PROPOSITIONS

Accompanying the Doctoral Dissertation:

TOPGENE:
An Artificial Intelligence Approach to a Design Process

By:
A. Zandi-Nia

to be presented with due permission
for public examination and debate
in the Aula,
Delft University of Technology
(Delft, The Netherlands),
on 25th of February, 1992, at 2 o'clock.

1- Many AI ideas, such as distributed problem solving, are appropriate and applicable to design discipline. Further studies can bridge some of the gaps between these two areas.

2- Experience with TOPGENE shows that heuristic rules are a satisfactory means for solving a large number of problems that are computationally intractable and for which there are no fast algorithms.

3- An automatic design generation system based on qualitative reasoning often produces a realistic design whose performance is hard to judge, while a system based on quantitative approach often delivers an optimized design whose performance is easy to judge, but which may not be realistic. The experience of TOPGENE shows that a design generation system based on the above two reasoning approaches has the advantages of both.

4- Q-analysis provides a means for explication of hidden information in existing data, discovery of interconnections between elements of a set with respect to a common attribute between them, and clustering of the elements into groups having specific ties with each other. These properties of Q-analysis have many implications in some processes of architectural design.

5- The notion of operation has a significant role in the behavior of a design. This concept is often omitted or mixed with the behavioral aspects in models of design processes.

6- Chemical engineering has used graph theoretical notions known as Topological Indices (TIs) for structure-properties correlations in chemical substances. Some TIs are extremely useful in development of precedent-based architectural design systems for indexing, categorizing, filtering, and selecting precedents of designs.

7- The experience of this work with the Hopfield model of neural networks shows that this area of artificial intelligence has potentials for solving some of the design optimization problems that are classically dealt with by analytical methods.

8- The rise of religious thinking and environmental activism in the world is a consequence of carelessness of contemporary social-economical views about the human values and nature.

9- Scientific theories are much less subject to misinterpretations and manipulations than religious principles. One reason is that only scholars are concerned with scientific theories, while religious principles are mostly dealt with by ordinary people.

10- History is said to be written for us to learn from the past and to stop repeating our past mistakes. Yet, practically, history seems written to teach us not to forget to repeat our mistakes.

11- TU Delft has extensive capital investment compared to many universities in the world. Yet, the use of these investments seems relatively very low.

12- TU's quarter system, and its 4-year undergraduate program is incompatible with most of the well known universities in the world, and harmful to its educational quality.

13- A partial truth is a lie.

TOPGENE:
An Artificial Intelligence Approach to a Design Process

Proefschrift

Ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus
prof. drs. P. A. Schenck,
in het openbaar te verdedigen ten overstaan van een commissie
aangewezen door het College van Dekanen
op 25 Februari 1992 te 14:00 uur door:

Abolfazl Zandi-nia
geboren op 1-7-1952, te Kashan, Iran, Master of Science

This thesis is approved by the promoters:

Prof. Dr. H. Koppelaar

Prof. A. Tzonis

ISBN 90-9004900-2

Members of the promotion committee:

Prof. Dr. Ir. E. Backer,
Faculty of Electrotechnic,
Delft University of Technology.

Prof. S.J. Doorman, M.Sc.,
Faculty of Philosophy,
Delft University of Technology.

Dr. Ir. E.J.H. Kerckhoffs,
Faculty of Technical Mathematics and Informatics,
Delft University of Technology.

Prof. Dr. H. Koppelaar,
Faculty of Technical Mathematics and Informatics,
Delft University of Technology.

Prof. Dr. D. Schodek,
Graduate School of Design,
Harvard University.

Prof. A. Tzonis,
Faculty of Architecture,
Delft University of Technology.

Prof. Dr. G. de Zeeuw,
Faculty of Social Sciences,
University of Amsterdam.

ACKNOWLEDGMENTS

In 1986, when I was granted an scholarship by the Government of Islamic Republic of Iran for continuing my education towards a doctorate degree I was hesitant for accepting the challenge. Several people encouraged me to take the forward step. Others stood by me or backed me up when I needed help. I would like to thank these friends and organizations which made my latest adventure to the land of science possible. I specially would like to recall:

Ministry of Higher Education of Islamic Republic of Iran, and A.T. University of Tehran for providing me the opportunity for improving my scientific quality, and for their financial support.

My parents for their everlasting support during these years.

My colleagues at A.T. University, Iran Argham Company, and Planning and Budgeting Organization in Tehran whom I owe a great deal for their continuous support.

Special thanks to my wife whom I had her support and encouragements before and during my study. Also for her patience and bearing the responsibility of our children during these years.

And finally my promoters in T.U. Delft.

I have to give most credit to Prof. Henk Koppelaar, my advisor. He is a man of wisdom and endurance who never runs out of ideas. Without his persistence I would have never be able to wrap up my research in a relatively short period of time. I sincerely wish him the best.

Prof. Tzonis is one of the main initiators for application of Artificial Intelligence (AI) techniques to building design in TU Delft. I was lucky to have his consultation and expert advise to pursue this work. My gratitude to him for his support during these years.

Also my appreciation to Prof. Doorman for reviewing drafts of my thesis and giving valuable advises on the structure and content of the work.

Dr. Kerckhoffs of TU Delft guided me through the wold of neural nets. Thanks.

CONTENTS

CHAPTER 1
INTRODUCTION

1.1 Problems

This work integrates Artificial Intelligence (AI) with mathematical techniques to partially model the architectural design process and to manipulate architectural design knowledge for solving the following class of design problems:

Problem-1:

Given the *structure* and *operation* of an architectural design product, and a set of social norms identifying its *expected (desired) behavior*:

- *Evaluate* the design with respect to the set of social norms.
- Develop an *analysis* and give a *diagnosis* of the design for its behavior.

Problem-2:

Given a set of activities, and a set of social norms identifying the *expected (desired) behavior* of an architectural design:

- Generate a design with activities assigned to its locations, whose *actual performance* is as close as possible to the expected performance.
- Develop an *analysis* and give *diagnosis* of the generated design.

Problem-1 is a design product evaluation problem, the second is a design generation problem. Both problems are based on the relationship between the *formal-structure* and *behavior* of a building mediated through its *operation*.

The terms *structure, function, behavior*, and *performance* are often used to denote different phenomena in different knowledge domains. These terms are treated in detail in the next chapter. However, for the sake of clarity,

brief definitions of these terms and other key-terms such as *analysis, evaluation, and diagnosis,* related to the architectural domain, the specific focus of my investigation, are given here:

Structure:

Structure refers to the physical organization of a building, and more specifically the pattern of connectivities embedded in it. I also will refer to the connectivity pattern of a building as its *location access graph, topology* of the locations, or simply its *topology* [Tzonis87] in rare circumstances.

Operation:

I define the *operation* of a building as the dynamic aspects of a building as a result of the arrangement or allocation of activities within the building. The allocation of activities in a building in a certain way has direct affect on the patterns of movements of people and objects within that building. The circulation flow between different location pairs in a building and interactions between groups of people are, therefore, operational aspects of a building system.

Behavior:

I define the term *behavior* in this work to denote the functional qualities of a building, measurable in costs and utilities that it provides for its users. The behavior of a building with respect to the social norms mentioned earlier is a function of its operation, and it is constrained by the physical organization of the building.

Norms:

The behavior of a system is usually divided into the *actual behavior* and the *expected behavior.* The expected behavior of a building is commonly expressed in terms of normative requirements. A *norm* is therefore the *anticipated* or *expected behavior* of a building with respect to a single point of view. Experiments have led to the identification of several categories of norms with respect to the performance of buildings [Tzonis87]. This work deals only with a selection of norms *privacy, community, circulation-cost,* and *intervening opportunity* that are associated with the social qualities of buildings.

Performance:
The actual behavior of a design with respect to a norm is sometimes referred to as *design objective, design goal, criterion of evaluation,* and *performance* of the design. I will mostly refer to the actual behavior of a building with respect to a norm as its *performance* in this work.

Analysis:
Analysis of a design is the process of deriving the values of its (*expected* or *actual*) *performance* attributes from its structural description [Rosenman-90].

Evaluation:
The process of comparing the performance (i.e. actual behavior) and the expected behavior of a design, and judging its performance is called *evaluation* [Rosenman90].

Diagnosis:
The process of identifying the causes of (mis)behavior of a system with respect to a norm is often referred to as its diagnosis [Rosenman90]. This term, here, refers to the process of identifying locations of a building having a behavioral value (e.g., disturbance degree, costs, utilities) with respect to social norms.

The relationship between the structure, operation, and behavior of buildings is illustrated in figure 1.1.

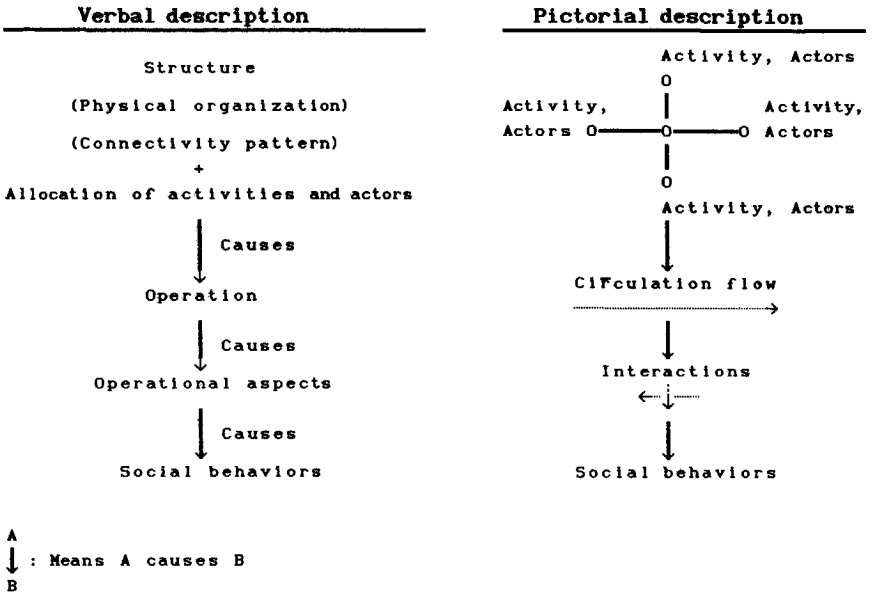


Figure 1.1: The structure, operation, and behavior relationships in a building

Architects deal with several dilemmas while facing a design. As opposed to more precise and simple kinds of artifact design, ambiguities, informalities, lack of information, and a vast number of constraints are characteristics of architectural design. These are characteristics attributed to design in general as well [Alexander66] [Aguero87]. The constraints of architectural design are often various, conflicting, and stemming from different sources. Orientation, light, access, and different performance requirements are examples of such constraints. In architectural practice, even a single set of design requirements might turn a design task into an impossible one. For example, the realization of connectivity requirements in a building might be an impossible goal to achieve. To avoid such situations, architects have to make trade-offs while designing. These trade-offs are often subjective and based on their intuitions. The introduction of more requirements into an architectural problem results in further increase in the complexity of its design process. Close examination of the problems, posed above, conveys several interesting characteristics that played an important rôle in

the development of the present system:

- There is a dynamic aspect associated with generating building designs as a result of the operation that they contain. Practically, the operation and behavior of a building are time dependent. I have suppressed the time element in this work for the sake of simplicity. I believe that the time factor, first, has only a linear affect on the computational complexity of our design problem, and second, it does not require an approach different than it is used in this work.
- There is ambiguity, lack of rigor and absence of information in the statement of above problems. The actual operation of a building in terms of degrees of interactions (flow) between its location pairs is not explicitly stated. This is important in quantitative evaluation of a design, and in generating a design product with respect to a set of social norms. In case of a design evaluation, the design product and the elements contributing to the operation of the design are known, while in case of a design generation the formal structure (connectivity pattern) of the building is unknown. Furthermore, no accurate degree of specificity is given on the level of performance requirements in both problems.
- There is a high degree of computational complexity associated with an architectural design at the abstract connectivity level. The number of possible designs increases exponentially with the number of locations in the design problem. The size[1] of the space of design solutions for n number of locations is $2^{n(n-1)/2}$. This size characterizes the design as an intractable NP-hard problem [Gary and Johnson79]. The introduction of domain constraints such as the planarity requirement into the search process reduces the size of the solution space to a limited degree but not enough to consider it as a polynomial type problem. This computational complexity was noticed by Berwick [72] before. The computational complexity of these and hundreds of similar problems [Gary and Johnson79] remains untouched, but advances in computer technology and development of new methods and techniques such as heuristic programming, and neural modeling,

[1]The number of possible graphs for n nodes is $2^{n(n-1)} / n!$. This number in the case of labeled graphs increases to $2^{n(n-1)/2}$ [Wilf86].

nowadays provide us with more efficient and effective ways of handling such problems than before.

- The design evaluation problem is computationally less complex than the design generation problem.
- Both problems are multi-dimensional, multi-objective decision-making problems in which each objective is expressed in terms of a normative requirement (expected behavior).
- There are norms that conflict with other norms. Conflicting norms are uncooperative in a sense that each attempts to achieve its own goal. Any attempt at optimizing such a norm has a reverse effect on the performance of a design with respect to other norms having conflict with it.
- Norms are incommeasurable and mutually independent of each other. Such a property common to these norms creates methodological and philosophical problems about the way in which a design can be optimized with respect to a combination of them simultaneously. Incommeasurability rules out comparison of the norms for making trade-off decisions in cases of conflict. Of course, one possibility to get around this problem is the assignment of weight factors between the norms. This is a strong assumption to make.

The main concern of this work is to report the implementation details of an operational system, called TOPGENE, for solving above design problems. The work also discusses a neural network approach for generating a sub-class of the design production problem, as well as proposing other AI based approaches for tackling these problems.

In the following sections after a review of motivations, objectives, scope, and context of this thesis, an introduction to the system is presented.

1.2 Motivation

Architectural design starts with a problem (a need), and ends with description of an artifact through a chain of complicated iterative processes involving problem analysis, abstract design, floor plan layout, etc. Finding methods for easing the architectural design process has been an objective of

researchers within the field. In the last four decades several attempts have been made to develop computer systems to attack architectural designs at various levels. Examples of these systems are space allocation systems aimed at generating floor plan layout of buildings from one or a limited number of points of view. Most of these systems were unsuccessful because of the underlying methods in their approach. Most of these methods were based on combinatorial optimization techniques [Tzonis85] [87]. Reliance of these methods on massive quantitative data and combinatorial calculations, and absence of architectural knowledge and qualitative data in these approaches, make them tedious, complicated, computationally expensive, and sometimes unrealistic in architectural practice. Inefficiency of these methods, and the need to devise new methods to cope with the complexities of architectural designs, demanded search for new solutions.

Advances in computer technology and software engineering, and the use of new problem solving methodologies have led to the use of computers in different scientific areas. In the realm of architectural design the use of computers has been limited and mostly CAD oriented. One reason seems to be the ill-structured [Newell69][2] and combinatorial nature of designs in general that make them unusually difficult candidates for automation [Simon-73] [Findler81]. The second reason is in relatively poor efforts in taking advantages of current progress in this direction.

Recent progress in the area of Artificial Intelligence has paved the way for research in the use of AI techniques and methodologies in architectural design. The main motivation behind this work has been to investigate and show possible application of AI approaches in automating a class of architectural design problems. These problems which are identified at the abstract (topological) level of architectural design are multi-dimensional, and some of them are computationally intractable. The approaches discussed in this work should go beyond the analytical, enumera-

[2]Newell [69] defines well-structured problems as having the following criteria:

- They are describable in terms of numerical variables, scalar and vector quantities.
- Their solutions are describable in terms of a well-defined objective function.
- An algorithm should exist for solving and stating them in numerical ter All other problems are considered as ill-structured.

tion techniques, and naturally are therefore more intelligent than brute force applications.

1.3 Research framework

The initial idea of this work was to design and implement a knowledge-based expert system for automatic generation of connectivity patterns of buildings exploiting domain knowledge of architectural design. However, during the work this approach appeared unfeasible for the reasons given in sections 1.3.2 and 1.8, while several other objectives exhibited themselves as appropriate for the effort. These objectives are discussed below.

1.3.1 Points of departure and research goals

The goal of this work from the design point of view is to provide a framework on how different disciplines such as mathematics, graph theoretical notions, and AI techniques can be employed for tackling specific architectural design problems some of which are computationally hard and intractable. From a computational point of view the primary objectives are:

- To discuss the combinatorial nature of architectural designs.
- To find the potential of AI techniques in tackling architectural problems in general. This includes the rôle of AI techniques in dealing with multi-dimensionality and multi-objectivity of architectural designs.
- To show the rôle of heuristics in reducing the complexity of search in design.
- Investigate the possibility of neural network approach in tackling design problems.
- Analytical comparison of designs generated for the same set of data but based on different approaches.

From the practical points of view the objectives of the work are:

- Design and development of a heuristic system capable of:
  - . Generating designs (connectivity-patterns) to architectural design problems with respects to a set of social norms presented above.

. Evaluating the quality of existing designs (architectural floor plans) with respect to a set of social norms.

- Design and development of a small scale and prototype neural network model capable of generating architectural designs (connectivity pattern of buildings) with respect to a set of social norms.

1.3.2 Desiderata

This work, as mentioned above, was targeted for a Knowledge-based system based on architectural design knowledge and a knowledge base of precedents of architectural designs at connectivity level. The initial studies of the problems posed revealed that, first, very limited expert knowledge of design is available, and second, these problems demand a deep reasoning strategy based on basic rules and laws of architectural design at micro levels rather than shallow expert knowledge. The dynamic aspect of our design problem, requires a deep reasoning process based on the mathematical analysis of the data, and externalization of new qualitative data for guiding the design process towards generating a design product. These facts resulted in giving a second thought in choosing the expert system approach. The system for this work is implemented in procedures. LISP was chosen as an appropriate language for representing the knowledge, modeling the mathematical techniques, and simulating the design processes.

A brief discussion on the differences between expert systems, and other systems, and their underlaying reasoning depth is given in section 1.8.

1.3.3 Caveat

This thesis reports on a research project, that has been conducted within the paradigms of AI. The results have been implemented into a system called TOPGENE and another small-scale neural network model. The fundamental issue in this research is, thus, development of new approaches outside of the enumeration techniques, to architectural design at topological level. The work also shows the power and the limitation of proposed approaches. Details of the work are given in the sequel; but, as for the sake of this introductory chapter, one should note that:

Much work is required to take a computer system from a research labora-

tory into the practical use. Crucial issues include: the appropriate scope of the program of the operation, integrating the system into routine operation, development of a user interface appropriate for both input data and the output data, etc. TOPGENE is tested and debugged to some extent; but the work does not present a fully polished system to be used by architects unless additional steps are taken.

TOPGENE, has a graphical interface for automatic input and display of patterns of buildings, and a limited help to navigate the user while working with the system. A polished version of the system should provide a more extensible help facility, and perhaps explanation facility for presenting the reasoning strategies behind the decisions made by the system while generating a design or evaluating existing designs.

Another important issue with respect to a working system is testing and tuning of the system with respect to real architectural problems. The system developed has been tested to a limited extent. However, more effort is needed in tuning the system with architectural environments, in order to claim the system to be operational for practical use.

1.4 Scope

An architectural design process takes place at various levels, each level comprising of several sub-processes. A rough sketch of levels of the architectural design is given below, while a more detailed review is postponed for chapter 2.

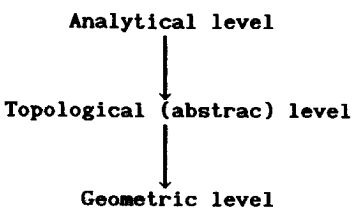


Figure 1.2 Levels of Architectural Design Process

TOPGENE addresses some issues involving design of the architectural floor plan at the second level, and in the presence of a set of specific social

norms. There exist several major classes of design norms in architectural design. This work restricts itself to the performance of a building at the presence of social norms: *privacy, community, circulation-cost, and intervening opportunity*.

The norms chosen in this work have common attributes that make them good candidates for a unique approach. All four norms, as are clarified in chapter 2, are sensitive to flows in a building. The potential flows between the location pairs in a building may be considered probabilistic and time dependent. This work, however, assumes constant and time independent flows. Consideration of flow as probabilistic and/or time dependent neither increases the computational complexity of the problem, nor the applicability of the method discussed in this work. TOPGENE's approach is also appropriate to other architectural problems, such as urban design, that deals with social norms.

TOPGENE, neither produces a design at the geometric level nor considers the metric distances at topological level of the architectural design. TOPGENE uses knowledge of design relevant to architectural design at the abstract connectivity-level. The system considers the interaction potentials between the location pairs in a building for generating a design or evaluating an existing design. The interaction potentials between the locations are calculated from the allocation of actors and the weight associated with them.

The scope of this research is, thus, limited to the topological level of the architectural design, and from now on, whether mentioned explicitly or not, the discussion is about designs only at this level.

To reiterate what was mentioned earlier, TOPGENE:

- Generates connectivity patterns of buildings with respect to a set of social norms.
- Evaluates the behavior of existing buildings with respect to a set of social norms, and produces analysis and diagnosis result for their behavior.

1.5 Context

This research is carried out in partial fulfillment of the Doctoral degree in Computer Science at Delft University of Technology. The work was carried

out in collaboration between the Department of Mathematics and Informatics and the Department of Architecture as part of the "Intelligent Architect Project" [Tzonis 75]. The objective of the project is to study and improve architectural design methods by studying and applying methods founded in the area of Artificial Intelligence. The ultimate goal of the project is development of an intelligent architectural system for automatic generation of architectural designs. The system should accept programmatic architectural problems and generate architectural floor plans based on the precedents of designs stored in a data base, and knowledge of the architectural design. The current subjects of the project for the study include:

- Pattern recognition.
- Discourse analysis.
- Expert Systems.
- Architectural data base.
- Architectural theory, and
- Architectural design methodology.

The result of this work will be integrated with other projects within the department towards the realization of the complete system.

1.6 The human analogy

Architects design (draw) floor plan layouts of buildings based on their traditional knowledge, experiences, and individual creativity. The process of architectural design is usually a long cognitive process with heavy dependency on subjective judgment and common-sense reasoning.

Generating a floor plan layout of a building requires architectural knowledge. The cognitive process that operates on architectural knowledge to create a plan is called architectural reasoning. Architects mentally work on a variety of levels and rely on their hierarchical knowledge to tackle architectural problems. An architect has to keep in mind different levels of information while working on a specific level of a design process. Such a correlation between knowledge at different levels of a design, while redundant in some specialized situations, still, comes to play an important rôle in most cases. There are spatial relations and connectivity problems. For example, in designing a house the program of requirements may dictate that:

everybody should have access to the living room, the kitchen most be next to the dining room, children's bedroom must be located next to their parent's bedroom, etc. There are also different performance requirements for different types of buildings. For example, prisons must have a high performance in terms of security and surveillance Libraries must be quiet and everybody using a library must share a comfortable and quiet study place. Obviously, there is not always a simple ready-made solution to every architectural problem. Architects usually start from the bottom up to come at a suitable solution for a design problem. Preparing the matrix of required adjacencies, generating alternative connectivity patterns under the architectural constraints dictated by the matrix of required adjacencies [Baybars80] [Hashimshony86], turning the matrix into a planar graph [Hashimshony80], turning a graph realization of a floor plan into a proper rectangular floor plan [Roth82] are examples of activities involved in an architectural design process.

An architect might face several alternatives in each stage of a design. Several solutions may be considered until one seems to fit best the problem. For example, at the abstract level several solutions about the connectivity of locations might exist for a single problem. In fact this multidisciplinary nature of architectural design justifies any attempt to look at architectural designs from different points of view. In this work the problem of generating connectivity patterns is viewed from the angle of social norms such as privacy, community, circulation-cost, and intervening opportunities.

In this thesis the objective is neither to simulate architects, nor to prove the psychological validity of any architectural theory, but to expand and improve design by discovering more intelligent ways of handling architectural problems. The approach should necessarily introduce an applicable and pragmatic system that benefits designers time / cost wise, and increases the reliability and the inter-subjectivity of their design products. However, this is the previous works of architects that provides the problems, provides the domain heuristics, guides to choose new methods, and helps to present new approach.

1.7 Introduction to TOPGENE

This research work has been implemented in a system called TOPGENE, an acronym for the "TOpological Pattern GENErator". An introduction to TOPGENE is given in this section. The implementation details of TOPGENE, the interrelationships between its different parts, and detail description of the underlaying methodologies are given in future chapters.

The architecture of TOPGENE

TOPGENE is capable of fulfilling three major tasks, based on the relationship between the structure and behavior of a building:

- Generating designs (connectivity patterns) with respect to a set of social norms.
- Evaluating existing designs with respect to a set of social norms.
- Analyzing and diagnosing existing designs with respect to a set of social norms.

TOPGENE consists of two major modules depicted in the following figure.

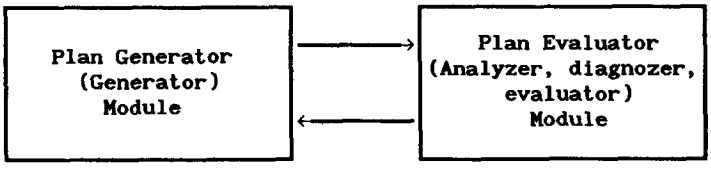


Figure 1.3: TOPGENE's modules

Design analysis and diagnosis is trivial compared to generating a design. A system for analysis and diagnosis of a design does not have to go beyond a model-driven approach. Such a system first has to analyze the input data, and make explicit the flow generation potential between the location pairs in a design, if such information is not given. Data analysis is also a major step in evaluating a design with respect to the social norms discussed. This is achieved by applying the Q-analysis [Atkin74a] [74b] [74c] [75] [77] method. In the next step, the system has to simulate the operational aspects of the design before analyzing its behavior and diagnosing its different locations for possible misbehavior with respect to the social norms. A

design analysis ends with modeling of the building behavior with respect to the circulation flow and interactions between the people moving within different locations of the building, measuring its actual behavior with respect to the social norms. A design evaluation is carried out after an analysis of a design and calculating its actual behavior. An evaluation is possible only if the expected behavior of a design is also given. This information is not a part of the input data in the evaluation problem discussed above. TOPGENE, acquires this information, by generating sub-optimal yardstick designs with respect to each social norm and uses their behavioral values as expected behavior for evaluation.

Architectural design at connectivity level by a model-driven approach is a search for a solution from among the space of possible solutions whose behavioral characteristics satisfy a set of social norms. I will show that this problem has a high degree of computational complexity and a large state-space of search. This implies that a model-driven approach is inefficient if not completely fruitless.

By viewing architectural design as a constraint-directed heuristic search, much of the knowledge for tackling this problem can be viewed as domain heuristics guiding the system towards evolution or rather configuration of a design product. This approach brings a high degree of efficiency in terms of computation time. The rest of this section tries to expose the components of TOPGENE and the methodology underlaying its implementation.

Variables

TOPGENE as a system capable of making inferences based on structure and behavior has to deal with variables one way or the other. The number of these variables is subject to change depending on the input problem and the values that are assigned to a subset of them. For example, in an evaluation problem the variable representing the connectivity pattern has a value, but this value is unknown if a design problem is posed to the system.

The important variables corresponding to different components of the TOPGENE are exhibited in figure 1.4.

Design aspects	Related variables
Structure: (Connectivity pattern).	Structure variables: - Locations. - Accesses between location pairs. - Paths between locations pairs. - Shortest paths between location pairs.
Operation:	Operation variables: - Actors in a location. - Potential flow between locations. - Flows crossing locations. - Flows crossing flows on locations.
Norms (Anticipated behavior)	Normative variables: - Community. - Privcay. - Circulation_cost. - Intervening_opportunity.
Performance (ACtual behavior)	Performance variables: - Community_utility degree. - privcay_cost degree. - Circulation-cost degree - Intervening_opportunity_utility degree.

Figure 1.4: Aspects of a design and their related variables

A design is always identified with a vast number of requirements (constraints) stemming from different sources. Architectural design is not an exception to this rule. The interactions between the requirements of a design are major contributors to the complexity of a design reasoning process. The design problem discussed in this thesis involves a number of constraints including the social norms, planarity, and branching degree of a location and several other related constraints. Two important set of constraints based on the domain knowledge are lists of the recommended and prohibited links in different building types. For example, in a hospital, from a doctor's room it is very likely prohibited to have direct access to the appointment room, although the analysis of data shows a high degree of interaction between these two locations. On the other hand, secretaries' rooms must have direct access to the doctors' rooms, although the analysis result indicates a lower priority for such accesses. The knowledge of prohibited and recommended links in typical buildings can serve to avoid the

system to make undesirable decisions while generating a design solution.

TOPGENE, although not a constraint satisfaction system, has to produce designs in a constraint environment. The major constraints considered by TOPGENE are exhibited in figure 1.5.

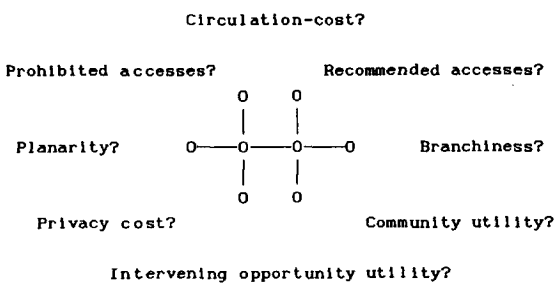


Figure 1.5: Design constraints considered by TOPGENE

Reasoning strategy

Computer systems based on AI techniques, or mathematical modeling, or analytical approaches, each emphasize different types of domain knowledge and reasoning strategies. Mathematical models describe physical laws governing a system. Artificial Intelligence deals with problems that demand computer programs based on domain knowledge, cognitive modeling, and which one way or the other emulate human intelligence. Both of these approaches are suited for reasoning about the behavior of a system based on its structure. Analytical approaches use differential equations to represent the structure of a system and use numerical or analytical methods to derive its behavioral values in numerical form. AI techniques, on the other hand, can give qualitative analysis of the behavior of systems based on their structural descriptions.

Common-sense causal reasoning is qualitative reasoning about the behavior of the mechanism of systems. The basic characteristic of common-sense reasoning is the absence of external memory or calculation aids [Kuipers84]. Qualitative reasoning as opposed to analytical reasoning uses qualitative description of a system's structure and uses heuristics to derive the behavioral description of the system in qualitative form. This later type of

reasoning is important in domains such as medicine, and in cases where precise numerical data characterizing states of a system are not available or are hard to attain. Qualitative descriptions of physical systems and the line of reasoning behind their behavior tend to be much weaker than quantitative reasoning based on quantitative data and analytical methods. This is evident from the reasoning of physicians. Physicians tend towards numerical data based on the analysis of, for example, blood samples, when their qualitative approach in diagnosing patients and predicting patients disease fails. Quantitative analysis, thus, can resolve qualitative ambiguities, and it can provide quantitative estimates values for parameters that cannot be measured or are hard to measure qualitatively [Kuipers84].

This work proposes an integrated approach based on AI techniques, mathematical modeling and domain heuristics for solving above problems. TOPGENE uses a heuristic approach in generating design, and analytical methods to quantify the operational and behavioral aspects of a building system. Such a mixed approach takes advantage of analytical methods and the efficiency of heuristics strategy, which is known as the basic aspect of common-sense reasoning and AI systems. The exclusion of either one from the approach is not without consequences. Without the heuristic power, one is shifted towards enumeration techniques and has to accept the burden of computational inefficiency. On the other hand, relying on pure heuristics, specially in a domain with analytical methods having crucial rôles in the quantitative analysis of the data, is not a wise approach. Without numerical analysis of a domain, a measure of goal attainment is hardly possible, or impossible. Quantitative data provides a means for constructing an evaluation function in a search procedure to compute distances of the intermediate states of the search with respect to a goal state [Nilsson80].

Another rôle of numerical data, at least with respect to the problems posed above, is tangibly to reflect the quality of design products with respect to the social points of view. This includes presentation of the analysis result of the design products in quantitative forms.

The existence of quantitative data, according to my belief, removes the sense of vagueness and subjectivity from a design, and presents its quality in a form that can be judged quickly with other competitive designs. Something that the present design products have been blamed for in other domains [Aguero87]. This holds for th architectural domain as well.

The integrated use of heuristics and analytical methods in TOPGENE is, thus, more powerful than the use of either one apart. Figure 1.6 displays a comparison of TOPGENE's reasoning method with pure numerical and pure qualitative methods.

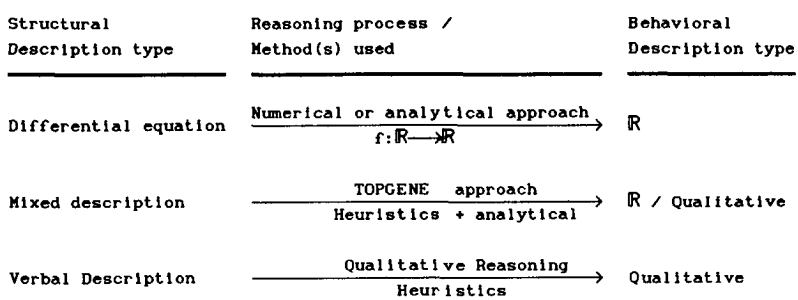


Figure 1.6: A comparison of TOPGENE's reasoning method with analytical (quantitative) and qualitative methods

Studies have been carried out around the causality in physical devices behavior [Iwasaki86]. Expert systems also have been built based on the structural-functional inter-dependencies of physical systems [Davis82] [Genesereth82] [Davis84] [Kuipers84]. TOPGENE is based on this concept. TOPGENE is built on a theory of architecture describing the mechanism of behaviors of a building as a result of its operation. TOPGENE uses the knowledge of the domain in the form of procedures and inference rules embedded in the procedures. The embedded knowledge in the system is based on the causal relations between the formal structure of a building in terms of the connectivities between its locations, and its performance or behavior, and to some level simulate the behavior of an expert trying to design or evaluate a building with respect to the social norms.

TOPGENE is, thus, capable of making inferences from the interrelation between the structural knowledge of a building (connectivity relations of locations) and its functional knowledge (i.e., its behavior) as a result of operation within the building. Causal relations are used to represent the reasoning of an expert while evaluating or designing a building, to capture functional-structural inter-dependencies, and to infer the effect of operations on the behavior of the building.

Knowledge Representation

Last section contrasted systems based on mathematical modeling with AI systems with respect to their reasoning methodologies. Another significant difference between these two categories of systems lies in their knowledge representation and knowledge manipulation approach. AI programs are characterized for their emphasis on heuristics knowledge [Campbell86]. AI systems also usually use integrated representation of domain knowledge. Several AI systems have been designed and implemented in various domains. Examples of AI systems are scattered around different fields. Knowledge Based Expert Systems (KBES) are a prominent example of these characteristics. Expert systems demand extensive knowledge about a domain for solving the domain problems. A sophisticated expert system might integrate different representation techniques to capture and use different levels of a domain knowledge. Domain knowledge can be categorized into two classes of knowledge. The first type of knowledge comprises the *factual knowledge* of the domain relevant to the problem solving, while the second type can be thought of as *causal knowledge* about relationships between different entities in the domain or the expert knowledge describing the problem solving process. Factual knowledge is usually encoded in *assertion* form [GoldWorks87], while the causal knowledge is represented in the form of *IF-THEN inference rules*. There are also AI programs based on the explicit representation of the knowledge of structure and behavior of systems. Davis's [82] [84] system reasons from the knowledge of structure and behavior in the domain of trouble-shooting digital electronic circuits. Genesereth's DART program [84] was built for diagnosing physical systems. These systems are similar to architectural building systems in the sense that there are logical relations between their structural and behavior.

TOPGENE's approach in integrating different reasoning strategies and different knowledge manipulation methods demanded different representation and manipulation techniques. The effectiveness of a technique depends on its fitness to a problem. TOPGENE uses an architectural design model based on first principles and facts of architectural design (domain heuristics) and includes causal relations between the structure and performance of a building as a result of its operation. TOPGENE also exploits mathematical descriptions of the structure and the operation of buildings and causal relations governing their behavior with respect to specific social norms.

The architectural knowledge used by TOPGENE is the knowledge of access in buildings and a limited expert knowledge of design at the abstract level of design processes. The first type of knowledge is the knowledge of prohibited and recommended accesses (connections) in a building can be taught to TOPGENE, and stored in a data base for repetitive use. The design knowledge is mostly in the form of general heuristic rules of architectural design for optimizing a building with respect to different norms.

The domain knowledge and causal relations in TOPGENE are embedded in procedures rather than explicit representation, encoding and storage in separate knowledge bases. Analysis and explanation of architectural design laws is ultimately based on the factual knowledge of structure and behavior of buildings and causal relations between them. The design knowledge in TOPGENE is stored both in procedures and inference rules embedded in procedures. The inference rules are capable of capturing the causal relations. Such rules encoded in IF-THEN rules in expert systems have the capability of making explicit the causality between a state of being (situation, event, etc.) and another situation to be reached or an action to be taken. The embedding of such rules in procedures hides the explicitness of causality, but has the advantage of gains in processing power [Winograd75].

Several types of knowledge can be identified and extracted from interview transcripts or other architectural knowledge sources such as architectural literature in this domain:

- Knowledge about the formal-structure of a building.
- Knowledge about the operation of a building.
- Behavioral knowledge of a building.
- Causal knowledge about the relations between the structural properties of a building (e.g., as branching degree, penetration etc.) and its behavior with respect to different social norms.
- Causal knowledge about the relation between operation and behavior of a building.
- Expert knowledge in the form of domain heuristics describing the process of architectural design. This knowledge can serve as a guideline for the search process.

TOPGENE, thus, uses:

- Graph representation techniques for representing the access proper-

ties of buildings. This includes *matrix*, *list* and *property-list* representations. Matrix representation of graphs is important to some sub-tasks such as an algorithm that calculates distances in partial designs. Access list and property list representations of graphs, on the other hand, are of use to LISP algorithms for path-finding and planarity testing, that cannot rely on the matrix representation of graphs.

- Matrix representation for representing associations between the activities (locations) and actors in a building. This representation is used by the Q-analysis method for inferring the circulation flow potential between location-pairs, and partially hierarchical clustering of them in the building.
- A knowledge base of recommended and prohibited accesses. This knowledge is taught to the system by the user for use during a design configuration.
- Causal knowledge relating the structure and behavior of buildings in the form of IF-THEN rules embedded within procedures, and
- Expertise knowledge of the design at the pre-metric level in the form of procedures.

The mathematical model of the system uses Q-analysis to recognize and organize the interrelationship (potential flow) between the location pairs as a result of the responsibilities of actors for certain activities. The calculation of flow is important to different activities such as: synthesis, analysis, evaluation, and allocation in design.

Methods

TOPGENE incorporates the following methods for its reasoning process:

- An iterative improvement bottom-up approach used in conjunction with hill-climbing and heuristic techniques to arrive at a design.
- An agenda mechanism and relaxation method to guide the system in improving partial designs with respect to the social norms.
- Q-analysis method for externalizing the flow-generation potentials between the location (i.e., activity) pairs of a building, and partially hierarchically clustering of the location-pairs with respect to their interaction potential degree.

- Graph theoretic notions for diagnosing the intermediate partial designs and improving (incrementing) their structure towards a complete design product.
- A fast algorithm for keeping track of distances in partial designs. This algorithm emerged through the needs of TOPGENE.
- Different graph algorithms such as planarity testing and path finding for manipulating partial designs during a design process.

Facilities

TOPGENE provides limited facilities for practical use. These facilities include:

- Limited help facilities for guiding the user of the system.
- Menu interface for efficient inputting of design data.
- Several data bases of existing designs and other design data for one time input and storing, and multiple use.
- An automatic heuristic graph-displayer for input of the existing designs for evaluation, also output of the generated designs.

Testing

Experiments were conducted to compare design solutions with varying styles for the same design problems. These experiments, to be discussed in chapter 6, confirm with the structural and behavioral knowledge of buildings, and with one's intuition. Tests were also conducted to compare designs generated by TOPGENE and a neural network implementation of a sub-set of design problems. These tests, partly presented in chapter 8, also show that the heuristic rules underlaying TOPGENE are sufficiently powerful in producing near-optimal designs with respect to the social norms.

1.8 Why not an expert system?

An expert system is primarily based on a large collection of rules describing interrelations between (important) concepts in a knowledge domain. Expert system technology, since its birth, has made an outstanding contribution to the discipline of Artificial Intelligence in general. XCON, MYCIN, and PRIDE are examples of successful expert systems in commercial use today. XCON is used for configuring VAX computers [Bachant84]. This system is

capable of validating the technical correctness (configurability) of customer orders and to guide engineers to assembly of these orders [Barker89]. MYCIN is an expert system in medical diagnosis that solves the problem of identifying an unknown bacteria based on input laboratory information [Buchanan84]. PRIDE [Mital85] is an engineering assistant system. However, experience [Parrello88], and history of expert system reveals that: this technique, like other techniques, is only fruitful in well-understood and appropriately chosen domains. In particular, this technology is suitable only for highly specialized and narrow areas, rich with a considerable amount of expert knowledge, and does not promise to work if breadth of the scope, level of expert knowledge, and careful feasibility study are not taken into consideration seriously [Bobrow86].

Bobrow [86] characterizes expert systems as programs that apply substantial knowledge of specific area of expertise to problem solving. He refers to the term "expert" as to imply both narrow specialization and competence. Expert systems are also characterized as being shallow in their chain of reasoning as opposed to deep systems that rely on the deeper rules and principles of a domain.

A system is characterized as shallow if its chain of reasoning is rather short. A deep reasoning system uses (long) chains of reasoning based on a knowledge comprising the underlying principles and theories of a domain [Bobrow86]. In general, however, there is not a guideline from the AI field on what makes a model deep, and on the appropriate depth of models for a specific task. A model is usually referred to as shallow, if it draws conclusion directly from the observed facts that characterizes a situation. Such models have the advantage of embedding the heuristics that experts usually use in performing their line of reasoning. Klein [87] uses a simple relational definition for distinguishing depths of models of expertise for a specific reasoning task as follows:

> "Consider two models of expertise M and M'. We will say that M' is deeper-than M if there exists some implicit knowledge in M which is explicitly represented or computed in M'". [Klein87], PP:559-562.

Advantages and disadvantages of both types of reasoning have been notified by Klein [87] as follows:

Shallow models:

- Are relatively easy to build, provided that the expert knowledge is either available or easy to acquire.
- Are relatively efficient in terms of performance. This is because of the fact that they usually select a solution to a problem rather than to constructing one.
- Are inflexible in a way that they cannot deal with unforeseen situations that are slightly different from those explicitly included in the system.
- Are difficult to maintain, since conceptually a single piece of knowledge may be unsystematically distributed across several objects in a knowledge base.
- Can only explain their line of reasoning processes in a limited extend. Such an explanation tends to be shallow and based on the trace of their chains of inference leading to conclusions.

In contrast **deep models** of expertise have the following advantages and disadvantages:

- They Correspond more closely to the notion of reasoning from first principles.
- They tend to be more robust than shallow models.
- They can handle problems not explicitly anticipated.
- They exhibit higher level of performance at the periphery of their knowledge.
- It is easier to verify the completeness of deep models.
- They are more useful in generating explanation because of possibility of elucidating the implicit reasoning steps involved in a process.
- They are usually slower than the shallow models.
- They are more complex than shallow models, since they require more sophisticated control structure than the shallow models.

The experience of this work supports the idea that prototyping is a much more flexible approach for complicated design tasks demanding deep reasoning strategies. Expert system technology is only suited for narrow areas of design, where considerable expert knowledge is available for application. For example, reducing the design problem posed in this work, to a problem of

design of a library, a prison, or an office, makes it a good candidate for expert system approach, specially if such a system relies on qualitative knowledge rather than quantitative data. On the other hand, it is hard to perceive that a qualitative system can provide a deep analysis of a design without being equipped with mathematical techniques such as used in this work. The conclusion is that, integration of a shallow reasoning process with a deep reasoning process based on mathematical models provides a better working system than a system relaying on either approach alone.

1.9 An Outline of this thesis

A brief description of topics discussed in proceeding chapters is given in this section.

Chapter2:

Chapter 2 begins with a short review of definitions and characteristics of design in general, followed by a brief coverage of different models of design. A background on architectural design automation is presented in section 2.4. A Classification of architectural design as a process of creating an artifact rather than generating a description of a design is given in section 2.5. Section 2.6 reviews phases of architectural design with a focus on design at the abstract non-geometrical level, because of its relevance to this work. A discussion on the relationship between the formal-structure and behavior of a building as a result of its operation is presented in section 2.7. The subsections of this section contain the formal definition of and the rôle of social norms: community, privacy, circulation-cost, and intervening opportunity in a design. Section 2.8 presents detailed versions of the problems discussed in this chapter. The rest of this chapter (sections: 2.9-2.11) examines the computational complexity of architectural design at different levels, the computational complexity of design problem discussed in this work, and the rôle of architectural domain heuristics in reducing such a complexity.

Chapter 3:

Chapter 3 is devoted to problem solving and knowledge Representation (KR). Section 3.1 gives a brief review of three distinguished problem solving

methods in Artificial Intelligence (AI): Heuristic Programming, Machine Learning, and Neural Modeling. Sections 3.2 and 3.3 review relevant problem solving paradigms and AI problem solving techniques. The rest of this chapter is devoted to the KR. Sections 3.4 and 3.5 present definition of KR, the rôle of KR in software engineering, and criteria for choosing a representation technique. A brief review on advantages and disadvantages of declarative and procedural representation of knowledge is given in section 3.6. Section 3.7 covers graphs as means for representations of objects and their relationships, and representations of different properties of graphs in computers. The last part of this chapter (section 3.8) is devoted to the representation of architectural domain information needed in this work.

Chapter 4:

This chapter contains a survey of means for diagnosing the structural complexity of systems having a topological property representable as a graph. This includes the connectivity property of a building. Diagnosing the structural complexity by means of Topological Indices (TIs) has a significant rôle in molecular design in chemistry. The idea may have several applications in diagnosing building behavior as well. Section 4.1 covers this subject. Examples of topological indices are given in appendix A.

Section 4.2 of this chapter presents a new theorem for keeping track of distances in incrementally growing graphs. The theorem exposed in this work exhibited itself in a search for an efficient algorithm for keeping track of distances in incrementally growing partial designs. The algorithm and a working example of it is presented in the subsections of this section.

Topological indices deal only with the statics of graphs. In the next section of this chapter (i.e., section 4.3) a mathematical method for diagnosing the operation of a building is reviewed. This method known as Q-analysis has significance in data abstraction, clustering of objects with respect to a common attribute within them, also exhibiting a holistic view on the relationship between these objects.

Chapter 5:

Chapter 5 describes issues involved in the implementation of TOPGENE as a system. Section 5.1 gives an overview of the architecture of TOPGENE. Sections 5.2 to 5.4 describe assumptions made before the implementation of

system, design types recognized by the system, and the representation issues involved. Section 5.5 describes levels of data-abstractions in TOPGENE. Section 5.6 covers TOPGENE's approach to problem solving. This section is a repetition of what was said in chapter 3. Section 5.7 covers heuristic strategies used by the system in easing the search efforts and reducing the computational complexity. Sections 5.8 and 5.9 discuss the conflict resolution strategies and the mechanism used by TOPGENE for resolving a conflict during a design process. Planarity, branchiness and knowledge of connectivities between different locations in buildings are some of the constraints of architectural design at the abstract level. These issues are reviewed in sections 5.10 to 5.12. Sections 5.13 to 5.19 give the implementation details of designs with respect to a single and combination of norms respectively. TOPGENE's approaches to design analysis, design diagnosis, and design evaluation are presented in section 5.20.

Chapter 6:

This chapter is fully devoted to the experiments carried out with TOPGENE. Three different design problems are selected and used as test cases. Data sets related to these problems are presented in section 6.1. Section 6.2 presents the partially hierarchical clusters of locations (activities) and streams of locations generated by TOPGENE for these problems. Sections 6.3 to 6.5 presents examples of designs generated with respect to single norms for our data-sets. Generated designs are followed by their diagnosis and analysis results when appropriate. Examples of designs generated with respect to multiple norms and an analysis of these designs are presented in section 6.6. Section 6.7 presents examples of evaluation of existing designs for the problems posed in section 6.1.

Chapter 7:

This chapter discusses a neural model of a specific type (i.e., linear-tree) of architectural design problem. This model only generates designs with respect to a single norm: community or privacy / circulation-cost. The idea in this chapter is borrowed from the Hopfield model of the neural network (NN) for the travelling salesman problem (TSP). Sections 7.1 and 7.3 discuss Hopfield's model. In section 7.4 I discuss the NN implementation of the linear-tree designs with respect to a single norm. Algorithms for generating

linear-tree patterns with respect to community and privacy / circulation-cost norms are presented in section 7.5. Experiments and test results with the NN are presented in section 7.6. A summary of the chapter is given in section 7.7.

Chapter 8:
In this chapter I discuss the testing of TOPGENE and future works. A comparison of specific sets of designs generated by the TOPGENE and the NN is presented in the beginning of this chapter. Sections 8.2 and 8.3 discuss the limitations of both TOPGENE and the NN, and suggest extensions for both systems.

Chapter 9:
This chapter is the concluding chapter. The contributions of this work, and summary of issues addressed in this thesis are reviewed in short in this chapter. In the last section of this chapter (section 9.3) I propose two other possibilities for tackling design problems as discussed in this work. The first suggestion is within the paradigm of automatic discovery and the second one is based on means-ends analysis.

1.10 Summary

This work is an attempt towards the automatic generation of architectural designs at the abstract topological level. The work was targeted for a knowledge based system. Close examination of the problem, the domain knowledge, and the knowledge based technology, showed that a deep reasoning approach based on mathematical modeling is more realistic and more fruitful than a pure Knowledge Based approach. The system developed on this basis is called TOPGENE standing for the TOpological Pattern GENErator. TOPGENE uses mathematical modeling and heuristic knowledge of the domain to generate connectivity patterns of buildings. Heuristics, as used by human problem solvers, reduce the computational complexity of a problem solving process, but may not succeed in producing the optimal solution. Mathematical methods on the other hand are more capable than the heuristics methods, but are often inefficient in some problem domain.

TOPGENE by integration of heuristics and mathematical modeling takes

advantage of both methods. Experiments with TOPGENE show that this approach deserves more attention.

CHAPTER 2
THE DESIGN PROBLEM

The aim of this chapter is to specify the domain context. The first part of the chapter (sections 2.1-2.2) introduces definitions, characteristics, and classification of design. The following section describes a model of design processes that TOPGENE implementation is bases on. Sections 2.4 and 2.6 give a background on architectural automation, a classification of architectural design, and different phases of an architecture design. The subsequent sections (2.7-2.8) provide formal definitions of social norms, also details of the design problems introduced in chapter one. The rest of this chapter is devoted to a brief introduction to the problem of computational complexity in general, and complexity of architectural design at different levels in particular.

2.1 Definition and Characteristics

Design has been recognized as an important activity for more than 4000 years [Gero90]. Yet, scientific research in this field does not date earlier than 1960 [Coyne et al 90].

Design as craftsmanship is the process of inventing new physical things (artifacts) that have order and form in response to a set of requirements [Alexander77], or the process of creating objects that fit best a context [Stiny81]. The context of a design is anything that imposes certain demands or constraints on the artifact, this includes [Mostow85]:

- Functional specifications of the design;
- Limitations on the resources; and
- The set of formal criteria defined over the design.

Eastman [81] describes design as creating the description of an artifact rather than its realization as a physical object:

> "Design is the specification of an artifact that both achieves desired performances and is realizable with high degrees of confidence." [Eastman81, p. 13]

Gero [90] characterizes design as a decision-making activity that involves exploration and learning, and take place in a constrained situation within a context that depends on the designer's perception.

Simon [88] defines design in a broad sense as a process that tries to change any undesirable situation into a desirable one. He describes design in the context of the problem solving:

> "Engineers are not the only professional designers. Every one designs who devises courses of action aimed at changing existing situation into preferred ones. The intellectual activity that produces material artifacts is no different fundamentally from the one that prescribes remedies for a sick patient or the one that devises a sales plan for a company or social welfare policy for state. Design, so construed, is the core of all professional training; it is the principal mark that distinguishes the professions from science. Schools of engineering, as well as schools of architecture, business, education, law and medicine, are all centrally concerned with the process of design." [Simon88, p.129]

Design as a problem solving activity is also characterized as a process in an ill-structured [Newel69][1] [Simon73][2] environment. Design problems are

---

[1] Newell [69] classifies problems into two categories of Well-structured and Ill-structured problems according to the following argument:

"Well-structured problems are to ill-structured problems as linear systems are to nonlinear systems, or as stable systems are to unstable systems, or as rational behavior is to non-rational behavior. In each case, it is not that the world has been neatly divided into two parts, each with a theory proper to it. Rather, one member of the pair is a very special case of all the rest of the world-uncharted, lacking uniform approach, inchoate, and incoherent." [Newel169, pp: 363-64]

[2] Simon [1973] defines ill-structured problems in terms of well-structured problems. He characterizes well structured problems by the following properties:

- They are representible (in terms of at least one problem space).
- There exist criteria and algorithms (for testing their solutions).
- Defining and representing all the legal and illegal moves (in their state-space of search) is possible.
- There exist enough information for solving them.
- They are computationally solvable in a reasonable time.

attributed as ill-structured because of their multi-dimensionality [Tzonis-75], fuzziness of the measure of their goal attainment, incompleteness and vagueness of their problem statement, and lack of general guidelines in choosing alternative course of actions in their search spaces [Findler81]. Characterizing a design problem as ill-structured also means that either it does not have a clear-cut solution, or there is not a specific process for finding such a solution. A design may start with specification of a goal that must be satisfied by an object, or start with testing an object if it matches some requirements. This implies that design may take a bottom-up approach, a top-down approach, or a combination of them, depending on their nature and complexity [Coyne et al90].

Design as a problem solving activity is goal oriented in search of description of an object that satisfies a set of performance requirements. In this respect design (in a naive way) is viewed as the reverse process of the scientific explanation. Scientific explanation tries to describe the behavior of objects through observation, while design is aimed at describing objects that satisfy a set of performance requirements [Coyne et al90] [Gero90].

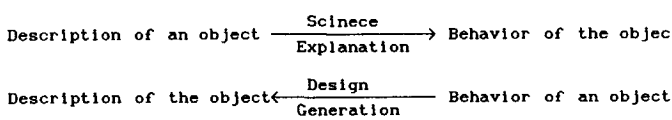


Figure 2.1: Design seen as the reversal of scientific explanation

Design often requires the integration of several disciplines for its accomplishment. The complexity of a design mostly arises from a combinatorial interactions between its requirements. Protocol analysis of three different design disciplines, architecture, mechanical engineering, and instructional design, has shown the following invariant characteristics within them [Goel89]:

- Design problems often exhibit lack of information, so they might suffer from freedom in statement.
- Feedback from the world is limited or delayed during a design process.

- Design objects are often judged as good, bad, better or worse, rather than right or wrong. Such attitude towards design objects entails the use of specialized evaluation functions and stopping rules on designs.
- The long, and expensive process of most designs demands a limit nested evaluation cycles or one final global evaluation.
- Designers are often forced to make and propagate commitments.
- The complexity of design problems often demands decomposition of them into several incomplete and "leaky" sub-problems.
- The solution decomposition of a design requires the mediation of goal and artifact by abstraction hierarchies.
- Design processes demand the use of representation systems and artificial languages for abstracting, filtering and processing of information.

Most of design processes usually take these sequences of actions [Harfmann-87] [Goel89]:

- Analysis and decomposition of the problem.
- Identification of the interconnections between components of the design.
- Search for sub-optimal partial solution for isolated sub-components, or with respect to isolated view points.
- Synthesis of (partial) solutions.
- Evaluation of the resulting design with respect to its functional or behavioral demands.

This process of subsequent sub-processes in design mostly requires backtrackings to previous steps and repetition of some sub-processes. A backtracking usually arises due to evaluation of result of a sub-process which demands some modification on the result. Such a design process is depicted in the following figure.

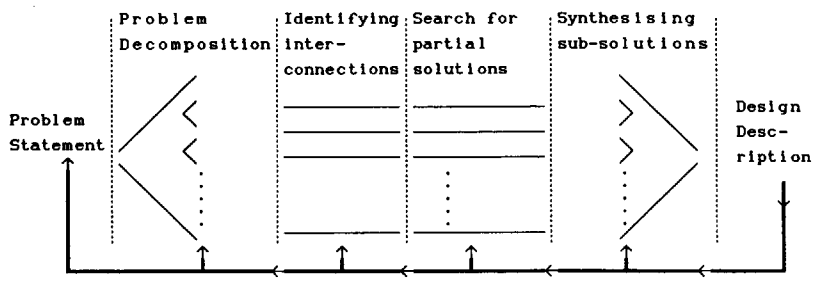


Figure 2.2: Sub-processes in a complex design

The difficulties in design of complex systems, such as architectural designs, arise when designers are trying to merge the sub-components or sub-optimal solutions to arrive at a global solution. Such a difficulty is largely because of the incommeasurability [Tzonis87] of design norms. This problem will be discussed in more detail in this chapter. Sriram [86] summarizes difficulties in design of large and complex systems as follows:

- The consequences of design are not apparent during the process of design. Many alternatives should be considered.
- A large number of constraints arising from multiple sources must be satisfied.
- Large design problems are often divided into several sub-tasks to be handled by individual experts. The interactions between these sub-tasks must be handled properly, and decisions by individuals should be justified. This is not an easy task when large number of decisions have to be made.
- Designers should be able to perceive the total picture of a design, as well as the locally optimum solutions for each sub-task.

2.2 Classification of designs

Simon [73] classifies design problems as ill-structured that can only be solved by division of them into smaller loosely-coupled well-structured sub-problems. This classification is alike to attributing designs as *Wicked problems* [Sriram87]. These are problems with no clear cut solutions.

Looking from a different perspective, design is considered as

formation-type problems, as opposed to most of the engineering problems which are in the *derivation-formation* category [Sriram87]. Derivation type problem solving is based on the hypothesis formation and identification of hypothesized solutions from a set of possible solutions. In the formation type problem solving, the problem solver only has knowledge of how to form a solution by applying appropriate problem solving techniques. This definition of design is most appropriate to a class of design (i.e., creative designs, defined below), rather than design in general.

A popular distinction between design types is their classification into *routine designs* and *non-routine designs* with non-routine designs further split-up into *innovative* and *creative designs* [Sriram87] [Coyne et al. 1990] [Gero90]. Sriram also defines *redesign* as another type of design process somewhere between innovative and routine designs. This classification of design, although may be differently defined from one author to another, but seems to enjoy a popular acceptance.

Designs also may be classified in terms of their computational complexity. Classification of design into routine, innovative, and creative design coincides with complexity degrees of design, with creative design as the most chaotic and complex type, and the routine design as the least complex type of design.

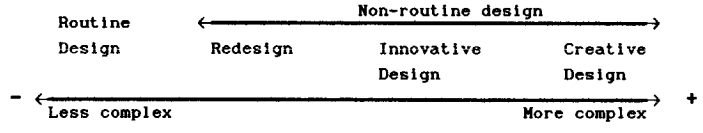


Figure 2.3: A classification of designs in terms of complexity

Routine design, the less complex and simplest type of designs, is a design process based on the assumption that both the knowledge of design and problem solving techniques are known in advance. Routine design is also characterized as a *prototype-instance refinement* [Gero90] process in which a priori plan of the design, and alternative components are known in advance [Sriram87]. The design process, here, consists of finding alternative sub-components that satisfies the new design requirements.

Redesign is the creation of new artifacts by modification of existing ones, so that it fulfills new functional demands [Sriram87]. Gero [90] calls this type of design routine design. Redesign, and routine design both proceed within a well-defined state-space of potential designs.

Innovative design, characterized as a *prototype-instance adaptation* [Gero90], is a design process in which the decomposition knowledge and variables (components) of the design are known, but the actual values of variables are in question. Innovative design, thus, proceeds in a well-defined state-space of solutions, and produce designs that are familiar in structure [Gero90].

Creative design, the most complex and chaotic type of designs, is a process that involves creativity and imagination. Designers face this type of design when neither an a priori plan (i.e., decomposition of the problem) for the solution of the design exists, nor existing design prototypes are applicable to the new problem. A creative design, thus, proceeds in an ill-structured environment and may produce a new design type.

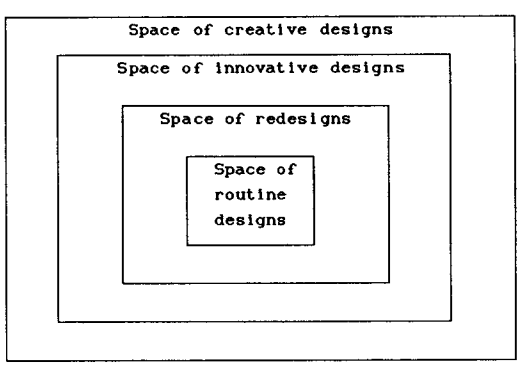


Figure 2.4: The spaces of different designs

2.3 A Model for design sub-processes

An abstract modeling of a design process helps understanding of their nature and aids in developing systems capable of generating designs. A set of design sub-processes may fit several models. Different design models have being proposed by researchers in the field. This section describes a design

model upon which the implementation of TOPGENE is based.

A design is characterized as a three-phase process consisting of an *analysis phase,* a *synthesis (or generation) phase,* and an *evaluation phase* [Coyne et al90]. The analysis phase involves understanding of the design problem and explicitly stating the design goals. This phase, also called *problem formulation phase,* is a problem decomposition phase that tries to analyze the design problem and find the relationships and interconnections between components of the design. The synthesis phase is the problem solving phase. This phase consists of the search for the design solution, albeit finding and combining of sub-solutions, or direct generation of a solution. The evaluation phase is the judging phase of the generated design. Evaluation is a process of comparing the actual behavior of a generated design with its expected behavior. Evaluation, in cases when a design does not meet the expected behavior, results in reformulation or revised analysis of the design problem and repetition of the three phases. This view of a design process (figure 2.5) is very much similar to the sub-processes in a complex design depicted in figure 2.2.

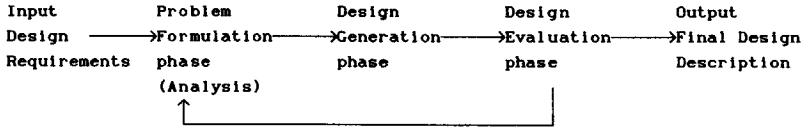


Figure 2.5: A general view of design consisting of three Phases

Gero [90] has proposed a model of design that is an extension to above view of a design process. Gero's model, based on the structure and behavior relationship in design objects, is incomplete in terms of most of the design processes. I will present an extension of Gero's model here. The prerequisite for understanding of this model is the definition of the terms: *structure, operation, behavior, analysis,* and *evaluation.* These terms, briefly defined in chapter 1, are treated in more detail and in relation with architectural design here.

Structure:

By *structure* one usually refers to the physical organization of a system in terms of its components and their underlaying connections. The components of a system may differ depending on the level in which they are considered. For this reason a system may be attributed with several structural properties each at a different level. For example, the structure of an electronic system at device level is the electronic boards and interconnections between them, while at the lower board level, its structure consists of electronic components in the board and the connections between them.

Structure, in this work, refers to the physical organization of a building, and more specifically the pattern of connectivities embedded in it. The connectivity pattern of a building may also be referred to as its *location access graph, topology* of the locations, or simply its *topology* [Tzonis87].

Operation:

An important missing part from Gero's model is the notion of *operation*. Operation has a significant rôle in the relationship between the structure and behavior of a system. Without an operation a behavior is not conceivable for a system. Operation is thus the property of a system, extrinsic to its structure, that causes it to behave in a certain way. The operation of a fork as a design object is the way it is handled. The operation of an electronic device is flow of electrons in the device that are initiated by its input. I perceive the logical switching of an electronic device as its operational aspects. Note that, while the physical organization of objects are constant, their operation may be a time dependent phenomenon that varies in time. For example, the operation of a fork changes as it is handled by different persons in different ways. The operation of an electronic device changes as its input varies.

Now, I define the operation of a building as the pattern of flow of people and objects within that building as prescribed by the interrelations and associations between them. The interactions between groups of people in a building and the pattern of movements between its different location-pairs are, therefore, operational aspects of that building system. The operation of a building, as it was defined above, is thus related to the arrangement of activities and actors within that building.

Behavior:

The *behavior* of a system is the conduct of the system as a result of the operation imposed on its structure. Davis [84], in describing his system developed for trouble shooting electronic devices, uses the term *behavior* to denote the way in which the input of a system (as a black box) is related to its output. He describes the operation of a system as a set of rules that maps its inputs to its outputs.

Here, I define this term to mean the qualities of a design with respect to different points of view, measurable in terms of costs or utilities that it provides for its users. The behavior, as it is defined here, is a multi-component type variable that reflects the points of view through which it is considered by its users or its designers. Furthermore, the behavior of a design at a particular time reflects its operation in that time. The behavior of a system is therefore under the influence of its operation and is constrained by its physical organization. The argument about the levels of physical organizations may be carried out here. As we attribute different levels of structure to a system, we can attribute a different behavior to each level as well. Furthermore, the behavioral aspects of buildings can be expressed both qualitatively and quantitatively. For example, the security degrees in different areas of a building can be stated qualitatively, while the privacy degree is expressible quantitatively in terms of the flow generation potentials between different location pairs.

Performance:

The behavior of a system is usually divided into the *actual behavior* and the *expected (anticipated) behavior.* The actual behavior of a design with respect to a norm is sometime referred to as a *design objective,* a *design goal,* a *criterion of evaluation,* and *performance* of the design. I will refer to the actual behavior of a building with respect to a point of view as its *performance* as well in this work.

Norms:

The expected behavior of a building is commonly expressed in terms of normative requirements. A *norm* is therefore the *anticipated* or *expected behavior* of a building with respect to a single point of view. Experiments have led to the identification of several categories of norms with respect to several

kinds of performance of buildings [Tzonis87]. This work deals only with a selection of norms *privacy, community, circulation-cost,* and *intervening opportunities* that are associated with the social qualities of buildings.

Function:
Each building is designed to serve a specific purpose. A house is designed for living, an office is designed for work, a school for educational purposes, and so on. The function of a system is defined as the purpose of a system for its behavior [Kuipers84]. In an example, in distinguishing the meaning of the terms behavior and function, Kuipers[84] explains that the function (purpose) of a safety valve in a boiler is to prevent the explosion, while its behavior is the level which it keeps the boiler at. The function of a system is obviously under the influence of its physical and operational organizations. This term is rarely used in this work.

Analysis:
Analysis of a design is the process of deriving the values of its (*expected* or *actual*) behavioral attributes from its structural description [Rosenman-90].

Evaluation:
The process of comparing the actual and the expected behavior of a design (with respect to a point of view) is called an evaluation [Rosenman90]. The goal of an evaluation is judging the performance (actual behavior) of a design [Rosenman90].

Diagnosis:
The process of identifying the causes of (mis-)behavior of a system with respect to a norm is often referred to as its diagnosis [Rosenman90]. The term, in this work, refers to the process of identifying locations of a building having a behavioral value (e.g., disturbance degree, costs, utilities) with respect to social norms.

The physical arrangements of locations of a building and the way they are accessible have impact on the operation and consequently on the performance of the building. The connectivity pattern of a building, the relative

distances between its locations, and the number of paths between these locations carry with themselves important behavioral information. They determine the relationships between locations, they imply the ways in which routes can run, along which people and objects can circulate through the building.

The allocation of activities in a building in a certain way also influences circulation flow between different location-pairs in a building. Circulation of people and objects in a building causes interactions that in turn results the building to behave in a certain way.

The structure, operation, and behavior of a building may be regarded as parts of its properties and its qualities, the relationships between them are depicted in the following figure.

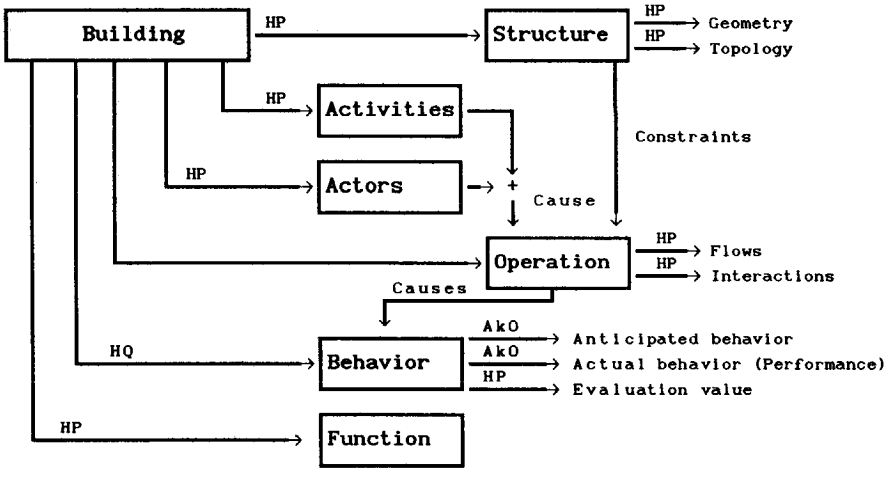


Figure 2.6: Causal relations between Structure, operation and performance of a building

Design as a process, based on the structural-behavioral relationship, is viewed as a chain of transformation sub-processes upon a set of entities. Design sub-processes include *formulation, generation, evaluation,* and *reformulation* [Gero90]. In the following discussions, ——→ means a transformation process, a ←——→ means a comparison, a sub-process is written on

top of an arrow, and the information that the sub-process needs is presented below the arrow.

A process that can hardly be considered as a design is a direct transformation between design requirements (i.e., D(R)) and the structure description (i.e., D(S)) of the design. This particular transformation which is achievable in the form of a routine design is as follows:

$$D(R) \xrightarrow{\text{Routine Design}} D(S)$$

A more realistic transformation between D(R) and D(S) can only take place through a chain of sub-processes based on consideration of operation (O) and expected behavior (i.e., Be) of the design.

$$Be \xrightarrow[O]{\text{Design Generation}} D(S)$$

Here, Be usually is estimated through a *formulation* (also called *specification*) sub-process. The objective of formulation is to explicate information from a design statement and to specify the expected behavior of the design object. The expected behavior of a design is a set of its qualities that are functions of the operation of a design.

$$D(R) \xrightarrow[O]{\text{Formulation (Analysis)}} Be$$

If we put together the last two sub-processes, we have:

$$D(R) \xrightarrow[O]{\text{Formulation}} Be \xrightarrow[O]{\text{Generation}} D(S)$$

A design, by nature, has attributes that exist in its performance space. The attributes of a design are either explicit and immediately obvious or not immediately apparent. In the second case attributes of a design are derived by application of analytical, or interpretive methods. For example, in a building design description, the branching degree of each location may be an explicit attribute, immediately apparent by visual examination of its connectivity pattern, while many of its other attributes, such as potential

interactions between its different location pairs is not an obvious attribute.

Any design description always maps into a unique point in its performance space, but the reverse is not necessarily always true. Several design descriptions may have the same performance, and thus, a performance value often maps to several design descriptions. This means that there may often be several design solutions to a particular design problem.

Derivation of performance attributes gives ability to assess the performance of a design with respect to certain criteria. The spaces of design description and of design performance, depending on the number of variables in the first one and the number of criteria in the second, may be multidimensional. However, in a given time, one might be only interested in certain attributes of a design. For example, given an existing design, one is always interested in knowing its actual performance relative to a desirable or expected performance, or having a design description, it is beneficial to know what is its performance with respect to a number of points of view?

The process of mapping a design description to values reflecting its performances criteria is called a *design analysis* [Gero90] [Rosenman90]. The purpose of design analysis is to derive its performance with respect to some norms once it is generated. Design analysis is carried out by applying a certain procedure. An analysis of architectural design with respect to social norms is a function of both the structural description and the operation of design.

$$\textit{Design description space } (D(S)) \xrightarrow[D(s) + O]{\text{Design analysis}} \textit{Performance space } (B_a)$$

The evaluation of a design is the comparison of its expected behavior with its actual behavior (performance) [Rosenman90]. Once an actual behavioral (B_a) value of a design description is known, then it may be compared to its anticipated (i.e., expected) behavioral value. An evaluation must provide sufficient information on design so as one can judge whether its actual performance value is acceptable or not [Rosenman90]

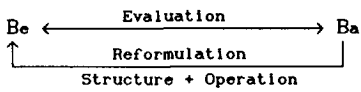
$$\textit{Expected behavior } (B_e) \xleftarrow[D(s) + O]{\text{Evaluation}} \textit{Actual behavior } (B_a)$$

Another inference process closely related to the analysis of a design is *diagnosis*. Diagnosis is the process of finding the causes of (mis-

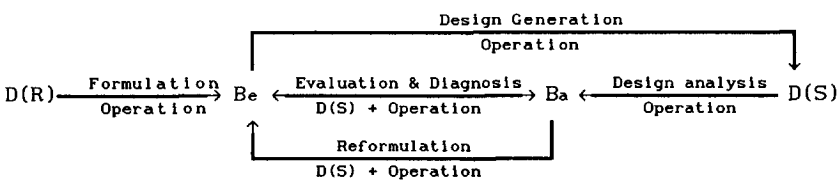
)behavior of a system. Diagnosis, thus, presupposes discrepancy between the expected and actual behaviors of a system [Struss88].

$$\textit{Expected behavior } (Be) \xleftarrow[D(s) + O]{\text{Diagnosis}} \textit{Actual behavior } (Ba)$$

Analysis, and evaluation of a design are followed by *reformulation*, if the evaluation of design shows that its performance (Ba) is unsatisfactory in some respect.



If we put the components of the process together we have the following model of a design. The implementation of TOPGENE is based on this model.



A $\xrightarrow[y]{x}$ B means that given A and y, process x produces B.

D(R): Design Requirements — Be: Expected behavior (norm)
D(S): Design (Structural) Description — Ba: Performance (Actual behavior)

Figure 2.7: A model for design sub-processes

2.4 Background on architectural design automation

There has been extensive research in bringing architectural design at a scientific level of thinking during last four decades. The motivation behind these efforts is based on several facts [Broadbent86] [Tzonis87]:

- Dissatisfaction about the improvement of architectural designs.
- Availability of new tools, such as computers.
- Emergence of new techniques such as simulation, operation research, and artificial intelligence.
- Successful application of above tools and techniques in other areas of science.

- Needs for transforming architectural design from a craftsmanship process to a more rational problem solving process.

Attempts in automating the architectural design process dates back to the invention of computers and emergence of operation research after the World War Two [Tzonis87]. In this period one can see the impact of operation research techniques in design [Coyne et al 90]. The first attempt towards the automatic generation of architectural floor plans is attributed to the work of Alexander and Chermayeff in late 1950's and early 1960s [Alexander-64]. Reviews of Alexander's works can be found in [Tzonis87] and [Coyne et al90]. The Chermayeff-Alexander paradigm inspired many researchers into development of new methods for generating architectural plans. These methods are classified under different names, such as *Space allocation techniques*, *Facility layout*, *Automated Spatial Synthesis,* and *Space Planning* [Eastman73] in architectural literature [Tzonis85] [87]. Space allocation methods saw architectural problems as quantitative combinatorial problems that could be expressed in terms of a matrix of interactions, and solved analytically. These methods used different types of constraints, such as relational (i.e., interaction requirements), locational (e.g., where to put a specific location), configurational (e.g., alignments, proportions, accesses) [Tzonis87], to reduce search efforts.

All space allocation techniques led to schematic partial architectural plans which were difficult to integrate into architectural design processes. The difficulties, inherited from the original paradigm, weakened the position of the techniques into a level of being considered as reductive in a design process, or of limited use in architectural practice [Tzonis87].

Two steps were taken in 1962 to improve the techniques: the development of more powerful algorithms and the introduction of interfaces between users and systems [Korf77] [Roch78]. The first effort improved the efficiency of the techniques in terms of calculation time, and the second allowed human intervention in search processes.

Galle [81] has categorized these systems in terms of their degree of automation as follows:

- Interactive systems for appraisal of users' designs.
- Step-wise automatic systems capable of generating intermediate partial solutions to be selected by manual control.

- Non-exhaustive automatic systems generating designs under a set of constraints.
- Exhaustive automatic systems generating designs under a set of constraints.
- Automatic systems generating optimal or quasi optimal designs under a set of constraints.

Still, space allocation techniques were seen as intermediary steps in the design process, and the implemented systems were used interactively by the architects for optimizing and evaluating design objectives one at a time, independent of other objectives. The introduction of more than one norm into these systems had the immediate consequence of incommeasurability.

An architectural design problem is essentially combinatorial and intractable. To appreciate the computational complexity of a design specially at topological level, one must notice that a problem with n elements has the potential for creating n!/(n-2)! interacting pairs and as many as 2^n interacting sub-systems. For example, a space planning program developed by Grason [68], took 23 minutes to find five solutions to a five-room problem stated in terms of adjacency and dimension requirements. It took 210 minutes for the same system to find only 1 solution for an eight-room problem, after which the program was forcefully stopped: Grason's reaction to exhaustive enumeration was that these approaches should be abondoned [Steadman76].

The trace of the exhaustive enumeration paradigm can be seen in the works of researchers even during 1970s and 1980s. For example, several works can be found in the area of exhaustive enumeration of floor-plans under space relations or dimensional constraints [Mitchell et al 76] [Steadman76] [Flemming78] [Gilleard78] [Galle81].

Flemming's [78] work described a procedure that could give solutions to space-allocation problems under the dimensional and relational constraints between the locations.

Mitchell et al [76] presented a set of algorithms for generating rectangular floor plans under adjacencies and dimension requirements. Their algorithm first generates exhaustively all distinct arrangements of locations under access requirements, and then arranges the rooms in an optimized fashion with respect to their covering surface.

LAYOUT, a system by Gilleard[78] enumerated all rectangular dissections based on the adjacency and size requirements. LAYOUT first generates all the adjacency graphs for a specific problem, and then the rectangular dissections based on the dual graphs of the adjacency graphs. Gilleard's algorithm, also based on exhaustive enumeration techniques, generates floor plans with rectangular space molecules based on user defined constraints such as the minimum and maximum area of each room.

Recent works in the area of knowledge-based expert systems, automatic discovery and neural networks have paved the way for development of new ideas and sophisticated systems in many scientific areas. Computers are considered as repositories of knowledge capable of performing intelligent tasks, competent in learning and discovering new ideas, and not as a calculating machine any more. Some new AI systems rely on a prior body of knowledge to solve problems. Something which was absent from previous systems. Examples of these AI systems are scattered around the scientific fields such as medicine [Shortliffe76] [Pople84], chemistry [Lindsay85], engineering [McDermott81] [Mittal85], etc.

AI systems employ heuristic methods and other AI techniques to go beyond the old paradigm of enumerating solutions. These techniques have proven to be successful in almost every area of science. Some of these systems are closely related to design methodologies in many respects. Their underlaying methods can be hired to attack design problems in many levels. There are many reasons to believe that these techniques might be equally useful in transforming the design process from a intuitive process into a more rational problem solving process. Such a transformation can help to increase the efficiency of design processes, also the quality of designs.

In spite of the widespread application of AI, it is believed that little improvements have been made in the area of architectural design [Tzonis85] [87] [Broadbent88] [Coyne88]. The existing body of knowledge provides sufficient ground for making new improvements in design methodology. The signs of new approaches towards design can be seen from the current research works in this direction.

Charles Eastman's paper "Automated Space Planning" [Eastman73], perhaps is one of the earliest contribution of AI to architectural design. Eastman developed a program called General Space Planner (GSP) that solved spatial arrangement tasks by taking the advantage of heuristics. GSP accepted a

space S, a set of Design Units (DUs), a set of Spatial-Relations S-R as constraints, and a set of operators for manipulating the locations of DUs within S. GSP was able to generate a space layout satisfying the set of constraint S-R.

Explanation Based Generalization (EBG) [Mitchell86] is a learning paradigm in AI whereby a learner learns from examples. Mostow and Bhatnagar [Mostow87] developed a system, called *Failsafe*, based on the EBG that could learn from its failures. Failsafe is in fact an intelligent space allocation system for generating simple floor-plan layouts of houses subject to dimension and adjacency constraints.

HI-RISE, is an example of knowledge-based expert system in building design [Maher85]. HI-RISE uses top-down refinement and constraint handling strategy to synthesis preliminary structural design of high rise buildings.

ALL-RISE developed by Sriram [87] is the extension of HI-RISE to include the structural design of all building types. This system is also a knowledge based system that uses top-down refinement and constraint-handling techniques to synthesize structural design of buildings. The architecture of ALL-RISE includes several knowledge sources and a blackboard mechanism [Rich83] [Hayes-Roth85] that allows communication between these several knowledge sources.

For detailed discussion on the history of architectural automation systems and references to the works carried out in this direction, readers are referred to Tzonis [87], and Galle [81].

2.5 Classification of architectural designs

In a broad sense, architects are believed to use four distinct approaches towards generating three-dimensional building forms: *Pragmatic design, Iconic design, Analogical design, and Canonic design* [Broadbent88]. This classification, which reflects the physical creation of the build forms rather than their descriptions, although does not reflect the scientific approaches towards design of new buildings, but in some cases coincides with particular models of design described by researchers.

Broadbent describes the *pragmatic design* as the way in which early men designed their shelters by choosing available materials, and putting them together in a way that seemed to work. Pragmatic design, as a trial and

error process, is common practice today as well, specially in situations when new materials are available and must be tested by use. Pragmatic design matches best the *redesign* described earlier.

Iconic design is defined as the repetition of typical existing designs which have been proved to work well for a particular climate or particular purpose. The cultural causes, the patterns of life, and the availability of certain materials have basic influence on persistence of iconic design approach [Broadbent88]. Iconic design is the practical model of the *routine design* described above.

Analogical design is another class of practical design mentioned by Broadbent [88]. Analogical problem solving plays an important rôle in every-day life. Human beings seem to use analogical methods in varying ways within their reasoning processes. From learning [Winston84] to teaching, and from reasoning [Winston80] [Winston83] to problem solving [Evans68] and finding a design solution to a complicated architectural problem [Broadbent88] analogy is a common practice. Analogy in building design can extend from abstract level to geometrical realization of the floor plan, and even to minor aspects such as decoration. A designer working at abstract level, for example, could use important aspect of prisons having rôle in their security performance, to design an office with a high security performance requirement. Another example could be given in micro level of architectural design, where the aesthetic aspects of nature direct designers in creating aesthetic forms. Analogy as a reasoning process is a practically proven method of problem solving in AI that could be well integrated with knowledge based design systems.

Canonical design is the process of creating form by setting up a two or three dimensional grid and trying to fit sub-modules of a design into the grid. This process ensures modularity and coordination in the outcome of a design process. The process starts by defining first the desired shape and dimensions of modules of the design in prospect. The basic dimensions of the grid will be determined according to dimensions of the modules. The obvious problem with this technique is the reconciliation of the grid, once it is decided, with design components on one hand, and fitting of the whole system with the environment on the other hand [Broadbent88].

2.6 The phases of an architectural design process

An architectural design problem usually demands organizing a given set of elements (activities) in space, under a given set of requirements. The process of architectural design can roughly be divided into two global phases, namely *the analytical phase*, and *the synthetic phase* [Broadbent88]. These phases may further be divided into sub-phases. For example, two distinguished sub-phases at the synthetic phase are the *pre-metric (topological)* and the *geometric phases* [Hashimshony86] [Tzonis87]. These two sub-phases are important both to the design processes, and because of methodological developments in recent years. These phases in architectural design are familiar to the general view of design, depicted in figure 2.5, consisting of three basic sub-processes. Research work on these two levels has been considered important and of equal value. The following sections give brief descriptions of these levels of architectural design.

2.6.1 Analytical (programmatic)

This phase of architectural design, also called *programmatic phase,* is identical with the *formulation phase* of a design process, discussed in section 2.3.

Traditionally, the basis of almost every architectural design process is a matrix of interactions that provides information about the interactions between different functional areas of a building [Reynolds80] [Hashimshony-86] [88]. The interaction information provides the basic ground for arriving at the adjacency and access possibilities of a design. This process is carried out in the analytical phase which involves analysis of the design to deduce the requirements of the design. Designers, in this phase, have to decompose and analyze design requirements, categorize them in terms of related concepts, define the elements of design and their characteristics, and deduce the program of requirements (POR) from design requirements. The ultimate aim of this phase is determination of the interaction information, and consequently the matrix of required adjacencies (MRA) which is an input to the topological phase. The required adjacency of a design reflects, to some degree, the expected behavior of the design. Figure 2.8 shows a matrix of interactions that reflects (potential) interaction degrees between the location pairs in a building, scaled within the range of 0 and 5. In this exam-

ple, a 5 indicates the highest interaction between corresponding locations pairs, the implication which is that these locations should have access or be located as close as possible. Similarly, a 0 indicates no interaction at all, that may be taken as very low priority in terms of access.

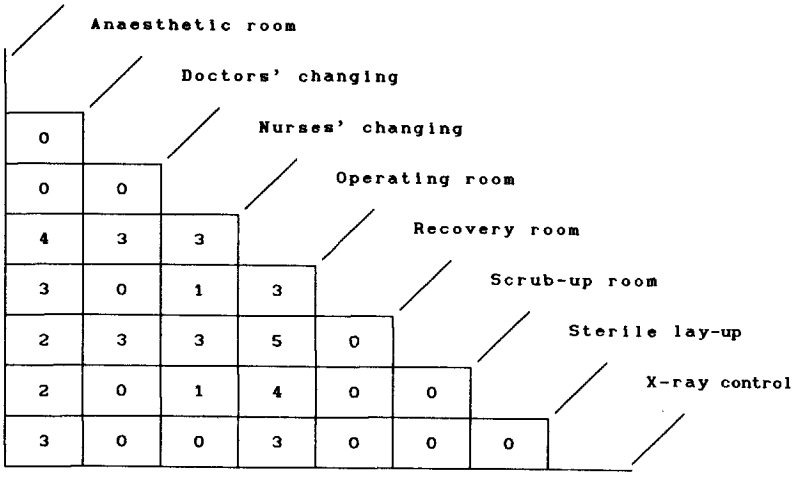


Figure 2.8: An interaction matrix exhibiting interaction potentials between location pairs in a design [Reynolds80]

The interaction matrix is used to arrive at the adjacency matrix or the access matrix. These matrices, which are symmetrical matrices hold information about adjacency and access of locations respectively. The polar and East-West directions, and exterior location are sometimes included in the MRA to emphasize the direction requirements or day-light requirements for some locations. In general an element a_{ij} of the matrix is either 1 or 0 depending on whether the adjacency (common wall) or direct connection (access) between locations i and j is recommended or not.

The entries to the adjacency matrix are sometimes weighted numbers representing the requirement degrees of adjacencies between the location pairs. In such cases the adjacency matrix should be further treated with some design considerations to obtain the more restricted adjacency matrix with binary entries. In such case definitions of interaction and adjacency matrices overlap with each other. For example Hashimshony[86] refers to the

interaction matrix as adjacency matrix, and to the binary adjacency matrix as the Feasible Connection Matrix (FCM). The reason is that often the empirical data reflecting the interaction potentials between the activities are not known prior to design solution. Designers, then, have to use intuition or rely on unprecise data from the program of requirements for guessing the interaction potentials between the activities. In such cases, instead of the real data providing basic ground for deciding on the adjacency and access properties of the design, intuition is used. These guesses and intuitions are usually reflected in the form of weighted entries in the MRA. MRA after further treated is converted into a binary matrix. Figure 2.9 depicts the MRA related to the interaction matrix depicted in figure 2.8. One should note that the adjacency is a necessary condition for realization of the accesses in an architectural floor plan. This implies that the adjacency matrix precedes the access matrix.

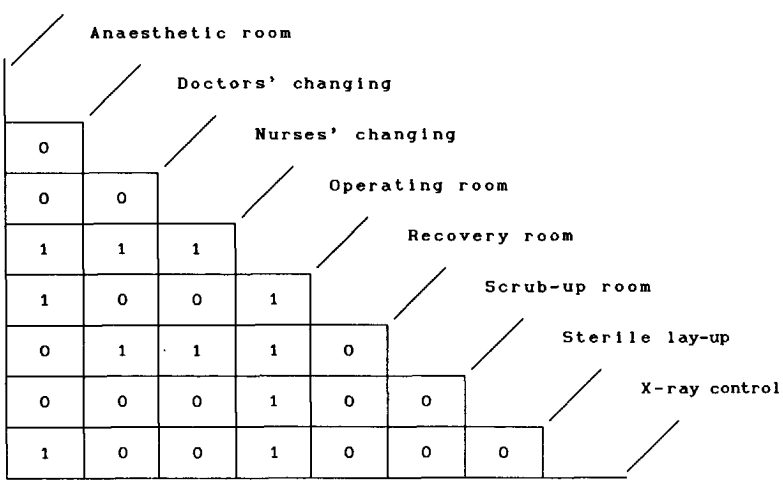


Figure 2.9: A matrix of required adjacencies (MRA)

2.6.2 Topological

The study of properties of objects which are invariant under *continuous transformations*, such as *twisting, pulling*, and *stretching*, belong to a branch of mathematic known as *topology*. The word topology is derived from two Greek words with the meaning *study* and *place*. Topology is defined as the

study of non-metric spatial relationships, such as connectedness and boundness [Flegg74]. Topological analysis in terms of building design is the study of the impact of relational organization of a building on its performance regardless of its other properties, such as sizes, physical shape, levels, etc. Continuous transformation destroys the geometric properties of objects but preserves the topological ones.

The importance of topology lies in the fact that the fundamental spatial concepts are not considered as metrical but relational. The most spatial qualities which can be treated topologically are: accessibility, adjacency, proximity, separation, enclosure, connectivity, visibility, acoustic, continuities, and properties such as localization of activities or objects [Tzonis87].

The topological relationship cannot be disregarded while designing buildings. Architects, when they are planning layouts, often speak of some kind of association between rooms or spaces that they manipulate in different arrangements. These associations have behavioral and operational implications in a design [Steadman71]. Topological arrangements of locations in a building, also abstract representation of space as graphs of topological properties gives a great degree of flexibility to the designers in the analyses of designs and finding of associations between its different elements. Another important aspect of this approach is its ability in formulation of architectural problems as mathematical models that could be solved by known techniques and computers.

The arrangement of locations in terms of adjacency or access is usually for various reasons. The most apparent is the access requirements. There is always a need for people to have direct access from one location to the other in a building. For example, in a factory material or goods have to be moved from one area to other areas. Adjacency also may provide natural lighting, ventilation, and viewing for a location, if such a location is adjacent to an outside space. On the other hand separation of locations may have its own reasons. For example, sound isolation or increasing the privacy of a room might be a good reason for keeping a room isolated from another.

The treatment of plan arrangement in terms of adjacency requirements at small scale, such as a house is perhaps more realistic than at a large scale such as an office building or factory. It is unrealistic to talk about adjacency and access for all locations in large size buildings. Instead we

can talk about the concept of nearness or proximity of rooms in them. In an office with possibly having hundreds of rooms or in a department store with more that hundreds of locations for displaying goods, or in a factory with a fairly long production line, the distance between the locations, which individuals must travel, and the arrangement of such locations in terms of nearness and proximity becomes more significant. These are the concepts that play rôle in optimization of design with respect to the norms discussed above.

The topological phase of architectural design is, thus, a conceptual phase which deals with schematics of a design in terms of its topological aspects such as connectivity (access) or adjacency between locations. In this phase, the programmatic requirements, translated into the matrix of required adjacencies, must be transformed into a layout graph of the design solution. The layout graph represents the elements of the programmatic phase and their interrelationships. Realization of such a graph is not always trivial. The ultimate adjacency or access graph of a building, at least, must be a planar graph so that it is realized at the geometric phase. This is not always the case. A graph corresponding to the MRA or the FCM is not always planar, and there may be several planar graphs corresponding to a non-planar one. An optimal planar graph should be sorted out of the generated planar graphs in this phase. Several researchers have proposed different methods for generation of an optimal graph in the topological phase [Baybars80] [Hashimshony86]. This phase is the invention of a design pattern which reflects the influence of constraints of the analytical phase. This work is concentrated on this phase with emphasis on social norms as the basic constraints in the formation of a building design.

2.6.3 Geometric

In this phase the layout graph is transformed into a two dimensional architectural floor plan. The transformation is carried out under a set of constraints. The constraints of the geometrical phase could be generalized as those which are defined in the programmatic phase and inherited in the adjacency graph, the functional performance of the design as it is viewed by a designer, and the forces of the contexts of the design such as physical and economical considerations. The result of this phase is the invention of

architectural floor plans which comply with those constraints. Design at geometric phase also may be seen as *pre-metric geometric* and and *metric geometric* sub-phases. The first sub-phase concerns the drawing) of rough sketches of the actual floor plans by designers, and the second sub-phase is when the metric constraint is imposed on the geometry of spaces.

The phases of the architectural design process, discussed above is depicted in figure 2.10.

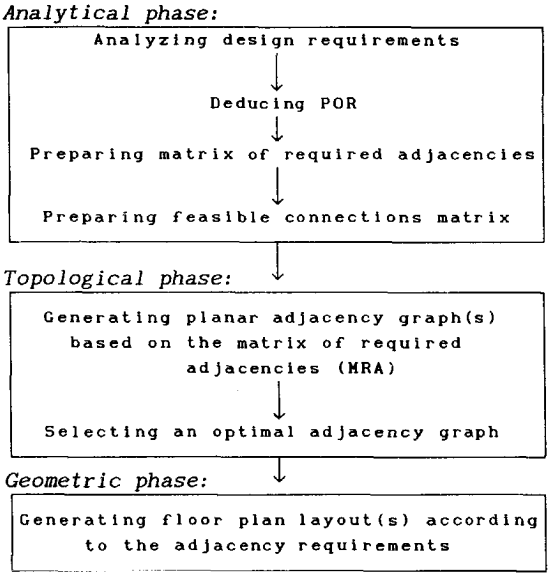


Figure 2.10: Phases of architectural design

2.7 The role of social norms in architectural design

As mentioned earlier, the expected behaviors of buildings are usually expressed in social norms. The social norms mentioned in this work are all under the influence and control of flows in a building. By flow, here the pattern of movements of people and objects in a building as a result of its operation is intended. The operation of a building is determined by the way in which the activities are allocated. If we assume that activities are bound to the locations in a building, and if some actors are responsible for more than one activity, then one can assume that these actors move between locations within the building and interact with each other. The movement of

the actors within the building, and their interactions contribute to the social behavior of a building. The theory can be explained by an example. We take a simple access pattern of a building and some arbitrary activities and actors responsible for them. By choosing a different building pattern and arbitrary assignment of activities to the locations (allocations) and observing the patterns of flow as a result of the operations imposed on different building types, we can analyze the impact of the structure and operation on the behavior of designs. We take a small design problem, with 4 locations as follows:

{A1 is an activity assigned to groups G1, G2}
{A2 is an activity assigned to the group G2}
{A3 is an activity assigned to groups G3, and G4}, and
{A4 is an activity assigned to groups G3, and G4}

Several designs exist for above problem, depending on the access pattern (structure) of the design, and also the way in which activities are allocated to them. To make life simple, we take only four design solutions, as depicted in figure 2.11, from among all possible solutions, and compare their performances.

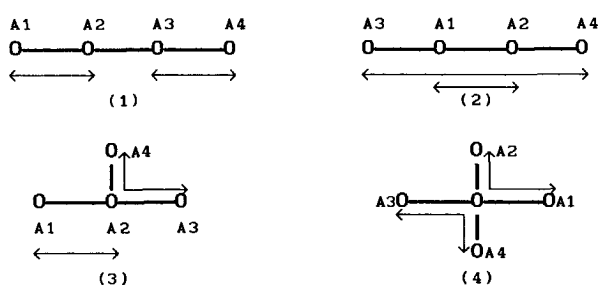


Figure 2.11: Four different solutions for a design problem

The arrows in the above figure show the flows between locations that share actors. Solutions (1) and (2) have the same structure, but have different operations, and consequently different behaviors. This is because of the way that activities are allocated to their locations. There are no interactions between the actors in solution (1). So, solution (1) has a behavior in accordance with the privacy norm, but poor in terms of the community norm.

In contrast to the solution (1), solution (2) has a relatively good performance in terms of the community norm, but it behaves poor in terms of the privacy norm. The sharing groups between locations that are identified by A3 and A4 disturb residents in locations labeled by A1 and A2 and create some sense of social amenity for the building, while moving from one activity assigned to them to the other. By the same tokens solution (3) has some advantage over (2) with respect to the privacy norm. The most interesting solution in terms of both norms is number (4). In this design, while all resident groups enjoy benefit of non-intervention, still there is a potential for new group formation at the central location that may be a corridor. This location absorbs all interactions, creating social amenity, and it is also a perfect place for allocating an intervening opportunity activity. These solutions also have different implications in terms of the circulation-cost. A topologically more compact solution, intuitively, has more benefit in terms of this norm relative to stretched solutions. This obviously is only true under relatively identical operations for all designs.

This example indicates the interrelationships between the structure and operation of a building, the effect of flow on the actual behavior of a building, and the rôle of social norms on a design process. The example, also showed the existence of conflicts between some norms. Another important characteristic for such a design problem is that the norms are incommeasurable, and there is no common factor in converting one to another. Such a characteristic of architectural design problems, limits choices of techniques that one may borrow from other disciplines in dealing with them. The following sub-sections treat the formal definitions of the social norms dealt with in this work.

2.7.1 Community norm

The norm community, also known as *social interaction opportunities* and *social amenity* [Tzonis87], concerns the degree of social interactions and associations between different groups within a building. The community utility of a building can be measured in terms of degrees of group formations within the building.

If we assume that two activities with identical groups are assigned to

more than two locations in a building, one may assume that these groups would travel between their assigned locations, and appear at intervening locations. As a result they will encounter other groups traveling or residing within those intervening locations and form new groups. For example, in figure 2.12, if the activities in locations L1 and L8 are assigned to the group-sets {G1 G2 G3}, and {G1 G2 G5} respectively, then one can assume that the groups G1 and G2 will travel between these two locations and will appear at intervening locations L2, L5, L6, and L7. Similarly the group G7 responsible for the activities in locations L3 and L4, will move between these two locations and will appear at location L5. As a result {G1, G2} forms a new group set {G1, G2, G7} with {G7} at L5, a new group {G1, G2, G4} with {G4} at L6, and a new group {G1, G2, G5} with G5 at L7. Formation of these new groups contributes to the social amenity of the building.

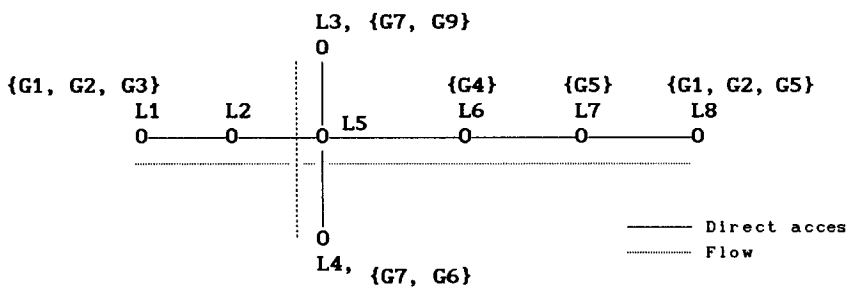


Figure 2.12: Patterns of flow in a building as a result of its structure-operation interrelationship

Social amenity in a building, as described in above example, is proportional to the degree of new group formations (i.e., interactions) in each location. The formation of new groups within a building may take place under three conditions: overlapping flows in a location, a flow crossing another flow in a location, or a flow crossing a resident location. The impact of new groups formed on the social quality of a building is obviously proportional to weight of new groups, which in turn is proportional to both the weight of flows between location-pairs and the weight of groups residing in locations.

Expression of the total community utility of a building with complex pattern in terms of the actual group formation in its different locations is not easy. A formal definition of this norm for directed and undirected flow

may be given as follows, respectively:

$$\text{Total } U_{co} = \sum_{i=1}^{n} \sum_{j=1}^{n} f_{ij}\, i_{ij}\, u_c, \text{ for } i \neq j$$

$$\text{Total } U_{co} = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} f_{ij}\, i_{ij}\, u_c$$

where, n is the number of locations, f_{ij} is the flow degree between locations i and j, i_{ij} is the index of potential group formations between two locations i and j, and u_c is the unit utility for group formations. u_c may vary for different group formations.

2.7.2 Privacy norm

Privacy may be considered as a cost associated with the undesirable interventions by other groups in a building. As a designer likes to aggregate benefits for the social amenity and new group formation within a building, he might as well provide quiescence for the groups working or residing in different locations. This is possible by minimizing the interventions of groups walking in the building, to the other locations. The privacy norm is uncooperative and in conflict with the community norm. This means that a designer trying to maximize both norms at the same time often faces unresolvable situations. A formal definition of the privacy norm under the influence of the undirected flow is as follows:

$$\text{Total } C_{pr} = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} f_{ij}\, i_{ij}\, c_p$$

where, n is the number of locations, f_{ij} is the flow degree between locations i and j. i_{ij} is the index of potential interventions for the resident groups between two locations i and j, and c_p is the unit cost for an intervention. c_p may be taken different for different group interventions

2.7.3 Circulation-cost norm

Circulation-cost is associated with the number of trips that groups of people make for different purposes within a building. In an office building, for example, the staff members carry out duties in a regulating way, and their relations to other staff members are more or less fixed. So, each staff member has some closer collaborators than other colleagues, and as a result the number of trips that people make between specific locations are more significant than other trips they might make. These trips mean waste of time and money for the office. This is what is called circulation-cost. It is wise for the architect to be concerned about this cost, and try to organize the activities in such a way that the distance between most interactive activities are minimized. Circulation-cost is directly proportional to the number of trips, distances between location pairs, the time spend for each trip, and the individual travelling cost per unit of time [Berwick71]. The cost per unit of time for individuals or specific groups of people could be taken as proportional to their salaries. The circulation-cost also can be categorized in terms of its different types. For example, in a hospital one can take into consideration the circulation cost for staff, visitors, trolly movements, service movements, etc. separately.

If distances between all pairs of locations of a building are captured in a distance matrix (d_{ij}), and the number of daily trips made by different groups of people between the same locations as in the distance matrix are stored in a matrix (t_{ij}), then the cost for all pairs of locations per day in term of distances can be calculated as:

$$\text{Total } C_{cc} = \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij}\, t_{ij}, \text{ for } i \neq j$$

Now, the actual cost in terms of money for a period of time can be obtained by multiplying the time each person spends for his / her trips during that period by his / her unit cost.

$$\text{Total } C_{cc} = \sum_{i=1}^{n} \sum_{j=1}^{n} (d_{ij} t_{ij} c_{ij})/s_{ij}, \text{ for } i \neq j$$

where, n is the number of locations, s_{ij} is the speed in meters per hour,

and c_{ij} is the unit cost per hour for a resident travelling between locations i and j.

Circulated-cost also may be calculated based on the topological distance rather than metric distance. In this case, the cost is proportional to the number of links between two locations, and the circulating flow between location-pairs in the building.

$$\text{Total } C_{cc} = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} d_{ij} f_{ij} c_{ij},$$

where, n is the number of locations, d_{ij} is the topological distance (number of links) between two locations i and j, f_{ij} is the flow between two locations i and j, and c_{ij} is the unit disutility per link between two points i and j. c_{ij} may be taken equal for all location pairs.

2.7.4 Intervening opportunity norm

Intervening opportunity is a norm concern with the location opportunities in a building. As people travel within a building, they might aggregate benefits by passing by location opportunities, such as a newspaper stand, a snack bar, an activity location, etc.

The total intervening opportunity utility of a building is equal to the benefits that individual groups gather by coming at an intervening opportunity point, while traveling in the building. Such utility is obviously proportional to the flow rate between location-pairs, and to the presence of intervening points between specific location-pairs.

We can define this norm in terms of the flow rate, the number of intervening opportunity points, and the unit utility as follows:

$$\text{Total } U_{io} = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} u_{ij}^{k} f_{ij},$$

where, n is the number of locations, u_{ij}^{k} is the unit utility associated with the intervening opportunity k for locations i and j. u_{ij}^{k} could take negative as well as positive values. f_{ij} is the flow between two points i and j.

2.8 Defining the problems

The interplay between the structure and behavior of buildings in terms of the social norms, leads us to the formulation of three types of design problems at topological level, two of which were briefly introduced in chapter one. The first problem concerns generating connectivity patterns of buildings optimized with respect to the social norms. The second problem demands evaluation of the behavior of existing designs (buildings) with respect to the same norms. These two problems are the only one considered in this work. The third problem, brought up only for the sake of completeness of discussion, is not fully manipulated in this thesis. This problem deals with the allocation of a set of activities to an existing building in a way that certain social behavior is achieved.

2.8.1 Generating Designs

The design problem is as follows:

Given:

- A set of activities.
- Actors responsible for each activity.
- A set of social norms expressing the expected behavior (i.e., B_e) of a design.

Find:

- A design for the given set of activities that has a good performance behavior (i.e., B_a) relative to the expected behavior (i.e., B_a).

An analytical view on this problem shows that very little information is available in the problem statement. For example, while activities and actors are given, the expected operation of the design is not known. Also, in the problem statement, the potential flow between location pairs may be identified by looking at the common actors between them, but the overall flow in the design may only be realized when design is known and the activities are allocated to its locations. Furthermore, there is lack of information on the level of the expected behavior. A normative requirement does not give much information on the degree of the required performance behavior of design.

This problem may take different forms depending on the number and types

of norms involved. Mathematically speaking, the presence of a norm in the statement of this problem does not increase its complexity in mathematical terms. Addition of norms, however, changes the handling complexity of a design. This problem, to be discussed later in this chapter, is an intractable problem with a high degree of complexity. Other important criteria of this problem are the incommeasurability of the norms and conflicts between some of them. These two characteristics are important and must be taken into consideration if one tries to optimize a design with respect to a combination of norms (i.e different view points). A traditional approach in architectural design towards this problem has been based on generating suboptimal solutions with respect to each norm irrespective of the others, and choosing a compromised solution, based on multi-criteria evaluation techniques, which seems to satisfy best all the norms [Tzonis87].

The relationship between data and variables of the design problem is depicted in figure 2.13.

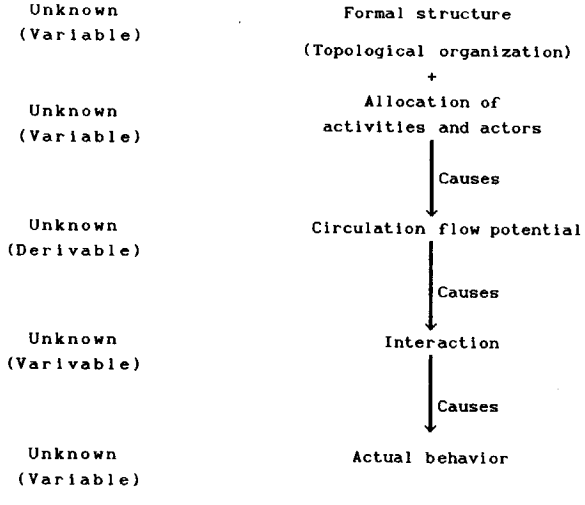


Figure 2.13: The interplay between data and variables in the problem of generating a design with respect to a set of social norms

2.8.2 Evaluating designs

The evaluation problem assumes that the formal structure (connectivity

pattern), the operation (activities and actors assigned to locations), and the expected behavior of the building are known, and the evaluation of the building with respect to a given set of social norms is required. Figure 2.14 shows a design under evaluation of four social norms (view points).

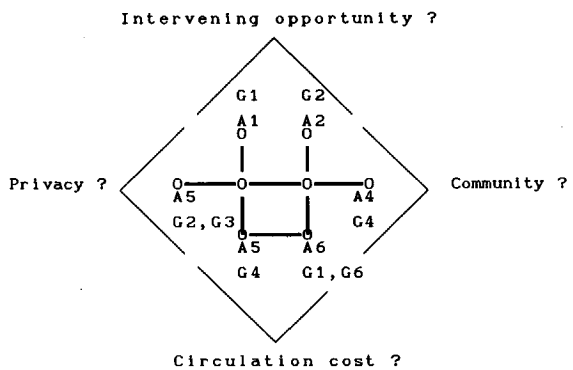


Figure 2:14: Viewing a design from perspectives of social norms

The evaluation problem may be stated as follows:

Given:

- A design (connectivity pattern) of a building.
- Activit(y) (ies) attached to each location of the building.
- Actors responsible for each activity.
- A set of norms expressing the expected behavior of the design.

Find:

- Evaluation of the building with respect to the given norms.

Noticeably, again several pieces of information are vaguely expressed in this problem. The operation of the building is not known and must be explicated by some means. Besides, the expected behavior, as in the case of the design problem is expressed in normative requirements. The norms do not carry any implicit quantitative or qualitative information in terms of the degrees of expected behaviors of the design.

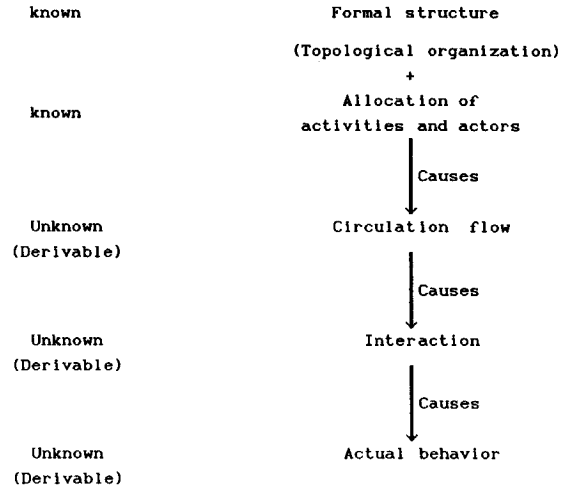


Figure 2.15: The interplay between data and variables in the problem of evaluating a design with respect to social norms

2.8.3 Activity and group allocation designs (fitness checking)

The third problem which fits our categories of design problems at topological level is the allocation problem. This problem requires the arrangement of a set of activities upon a building pattern under certain social behavior constraints. In this problem the structure of the building, activities, and actors responsible for each activity are given. The expected behaviors of the building, as in the case of previous problems, are expressed in normative requirements. The operation of the building is, thus, variable and directly depends on the way activities are allocated. Another unknown of the problem is the circulation flow potentials between the activity pairs. Circulation flow potential must derived from the given data. Here is a summary of the problem statement:

Given:

- A design in terms of a topological pattern.
- A set of activities.
- Actors responsible for each activity.
- Expected behavior in terms of social norms.

Find:

- Whether the activities fit the design well.
- Locate the activities on the pattern in such a way that satisfactory social behavior is achieved.

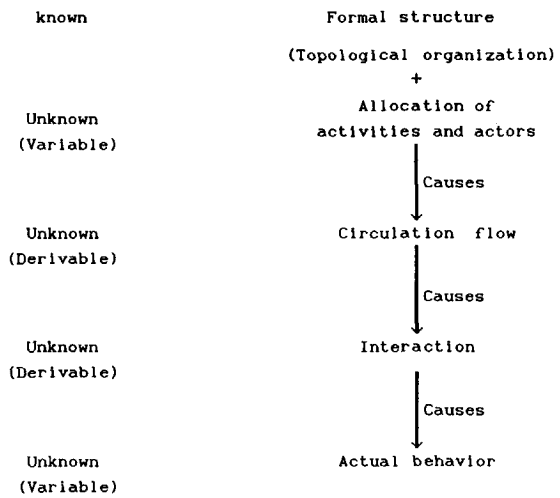


Figure 2.16: The interplay between data and variables in the problem of checking the fitness of a design with respect to social norms

2.9 On the complexity of architectural design

In this section we take a brief look at the complexity of architectural design at different levels, and complexity of the design problem discussed above. The concluding paragraphs discusses the rôle of heuristics in reducing the complexity of our design problem.

Complexity is an intrinsic nature of architectural design. Considering the architectural design as the act of arranging a set of locations (rooms, corridors, etc.) in such a way that to achieve a set of functional demands, the concept of computational complexity [Aho et al 74] [Gary & Johnson 79] [Wilf86] is an approach to show the difficulty of the subject.

Topological level

Practically, we can draw various unique graphs for the same set of nodes. The implication of this in terms of the connectivity property of a building

is that locations within a building can have access to each other in many different ways. The universe of connected graphs for a number of nodes, starts with the set of all possible trees over the nodes. A tree has at least n-1 links. If one is to enumerate all possible graphs of different sizes, then as the number of nodes increases, the number of possible connected graphs grows rapidly.

However, in the case of architectural design, a designer is interested only in simple planar graphs without loops or multiple edges. The number of possible graphs for a given set of nodes is dramatically reduced under the planarity constraint, but still grows fast for a number of nodes above six. The enumeration of all possible graphs for number of nodes up to 6 can be found in Steadman [76]. For example, for 2 nodes we have only a tree graph, for n=3 there are only 2 planar graphs (a tree and a cyclic graph), for n=4, there are 6 planar graphs (2 trees and 4 general graphs), for n=5 there are 20 planar graphs (3 distinct trees and 17 planar graphs), and for n=6 there are 98 planar graphs (6 trees and 92 general graphs). As I mentioned in chapter 1, architectural design at topological level not only requires taking care of the planarity of the design, but also is concerned with the behavior of the design with respect to different points of view. The behavior of a building with respect to social norms is directly related to its operation. This requires right allocation of the design with its activities (or location names). Labeling requirements increase the complexity of design at this level. The number of possible graphs for n nodes according to Wilf [86] is $2^{n(n-1)}/\ n!$. This number increases to $2^{n(n-1)/2}$ for the labeled graphs. These formulas show that enumeration of graphs in general has an exponential time complexity.

Figures 2.17 shows the number of planar, non planar, and labeled graphs for a limited number of nodes.

Transformation between topological and geometric levels

As in above case, the relationship between a graph of a building, and actual space arrangement is often one-to-many, and for every topological pattern there might be more than one floor plan arrangement. This is obvious if there is no constraint on the geometrical shape of the floor plan. However, to show the complexity, it is enough only to consider cases of square space arrangement and the rectangular dissections. These two cases for graphs of

up to 6 nodes, already enumerated by Steadman [76] [83], is reflected in figure 2.17. This figure shows that for a design of more than 6 nodes the number of possible space arrangements dramatically increases. One should note that the number of possible square arrangements and rectangular dissections corresponding to planar graphs are slightly less than what is reflected in figure 2.17. The reason is that in some cases there is not a floor-plan corresponding to a connectivity pattern.

Geometric level

Architectural design at this level is also a task with combinatorial characteristic, and there are many possible solutions to a given design problem at this level. The extreme case for this level is when we assume no metric constraints on the geometrical shapes and sizes of spaces in a design. This case, obviously, runs into infinite solution for any design. However, to check the level of complexity for real cases, we may look at the case of rectangular spaces, when a rather strong constraint exists on the geometry of spaces. The goal is to create combinations of patterns of rectangular spaces with a given set of square spaces. Furthermore, we restrict ourselves with the condition that the square cells are attached by sides, and holes are not permitted. For n=1, and n=2 We have only one possibility for each case. For n=3, there are two possibilities, For n=4, there are 5 possible arrangements. For n=5, there are 12 possibilities. The set of all possible combinations increases rapidly when n is more than 5. Haggett [67] claims that for seven cells there are over one hundred combinations, and for ten cells, over 4000 alternative combinations. This brings us to the issue of computational complexity in design problem solving.

2.10 On the complexity of our design problem, and the role of heuristics

Without a fast algorith for its solution, the design problem posed in this work is for all practical purposes an intractable problem. This problem, at first glance may look alike the problem of finding a tree or planar graph to fit a set of activities. Yet, the level of its complexity expands far beyond a combinatorial search for a tree or planar graph from among all possible ones. This is because of the fact that the norms involved in this design problem depend on the arrangements of the activities upon a given graph, and

not their arbitrary allocation. So, in fact one has to deal with labeled graphs for its solution. This means that, based on a combinatorial approach, one has to try all possible permutations of activities on every graph of universe of the planar graphs with specific number of nodes, in order to find the optimum solution with respect to a set of the norms. Berwick [71] has shown that a state modeling or integer programming of this problem ends with the permutation of activities on graphs, in such a way that it is equivalent to a generating all possible labeled graphs in search of a solution.

Wilf [86][3] claims that for n nodes there are $2^{\binom{n}{2}} = 2^{n(n-1)/2}$ labelled graphs and approximately $2^{\binom{n}{2}} / n!$ unlabeled graphs.

The number of possible planar, non-planar, and labelled graphs for a set of up to 9 nodes are included the following figure. This figure shows that our architectural design problem is an intractable NP-hard problem [Aho et al 74] [Gary & Johnson 79] [Kronsjo85] [Wilf86] [Nishizeki88].

Number of:									
Nodes (locations)	: 1	2	3	4	5	6	7	8	9
Graphs	: 1	2	4	11	34	156	1,044	12,344	308,168
Planar connected graphs	: 1	1	2	6	20	98	#700	#8,000	#200,000
Rectangular arrangements	: 1	1	2	5	12	?	>100		
Rectangular dissections	: 1	1	2	7	22	117	#700	$\#10^4$	#250,000
Labeled graphs	: 2^0	2^1	2^3	2^6	2^{10}	2^{15}	2^{21}	2^{28}	2^{36}

Figure 2.17: Growth of the solution spaces for design problems at different levels

One should note labeling of a pattern with the activities in certain way is merely for imposing a apecific operation on design. The flow potential between the location pairs are the important factor contributing to the operation of a building. Different operations affect in various ways the behaviors of a building with respect to different social norms. Some of

[3] A graph of n vertices has a maximum of $\binom{n}{2}$ edges. To construct a graph one has to decide which of these edges would be used, and one can make each of this $\binom{n}{2}$ decisions independently. For every way of putting an edge one may get a different graph. Therefore the number of labeled graphs on n vertices is $2^{\binom{n}{2}} = 2^{n(n-1)/2}$.

these norms, as discussed before, are conflicting and all are incommeasurable. These properties do not change the complexity degree of the problem in mathematical term, but increases its handling complexity. In other words, the size of the problem space is not changed under the presence of flows or norms. Yet, obviously the considerations for this design problem are more than just enumeration and labelling of graphs.

By inspecting definitions of norms, one can predict that a design with respect to the privacy and circulation-cost norms automatically favors condensed buildings with maximum connections between its location-pairs, while a design for the community norm tends towards a linear-tree type pattern. The reason is that a compact graph has a lower average distance than a less compact graph with the same number of nodes. Similarly, we can imagine that the community norm demands allocation of activities in a way they create maximum flow overlaps (interactions), while the other two norms favor minimum overlapping of the flows.

Figure 2.18 compares average distances (AD) between three different graphs with the same number of nodes (i.e. n=6).

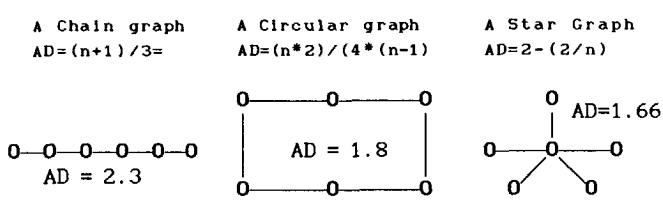


Figure 2.18: Graph compactness and average distance
[Tsonis87]

Similarly, we can see that the community and intervening opportunity norms demands allocation of activities in a way that they create maximum flow overlaps (interactions), while the other two norms act on the contrary. Altogether, one can conclude that, first the community and intervening opportunity norms favors linear-tree patterns, while the other two norms demand compact patterns; and second, maximum interactions on a pattern are only obtainable if most interactive locations (locations with maximum flow) are allocated as far as possible with less interactive activities between them. These intuitive conclusions may serve as heuristic rules, embedded in

an automatic system, for reducing the search efforts.

Trying to find a design that satisfies a combination of conflicting norms is not an easy task, since a step in optimizing a norm may have a reverse affect on another norm. A designer trying to optimize both norms will face difficulties. Besides the incomeasurability of norms do not allow relating a unit of a norm to another one for their comparison. This implies that practically a trade off between the norms is not possible, unless a strong assumption is made for such a purpose. For example, should the norm be compared to each other, one has to decide on the relations between the units of norms.

CHAPTER 3
PROBLEM SOLVING AND KNOWLEDGE REPRESENTATION TECHNIQUES

Chapter 2 identified levels of architectural designs, defined the social norms related to this work, and stated the problems concerned in this thesis.

The aim of this chapter is to review AI approaches to problem solving, and to present the problem solving and knowledge representation techniques that are either directly relevant to the implementation of TOPGENE, or appropriate to the architectural design processes.

3.1 Artificial Intelligence (AI), and its approach to problem solving

"Intelligence" is defined as to mean: "capacity to acquire and apply knowledge", but there is not a common definition over the term "Artificial Intelligence". Minsky [68] states that:

> "Artificial Intelligence is the science of making machines do things that would require intelligence if done by men". [Minsky68, preface]

The most common speculation in almost all definitions is the concept of "intelligent behavior" [Rich83] [Yazdani86]. Campbell [86] in an effort to bridge the gap between different definitions of AI, pictures AI as having three basic levels as follows:

- A process or model level,
- A heuristic level, and
- A computational level.

The top level consists of the computational models that have some inte lligent characteristics. Two types of model are recognized in AI [Campbell-86]: models that are related in some sense to human intelligence, and models that are not directly related to intelligence, but their behavior, as they are implemented in AI programs is at intelligent level. The first type of models, called theory generated models, are test beds for proving the legitimacy of the underlaying theories. For example Langley [87] claims that the ultimate goal of his work on scientific discovery is to provide a comprehensive theory of the processes of scientific discovery by men. His work is tightly coupled with weak heuristic methods.

Heuristic is attributed as the basic element of AI because of its recognition as a main tool of humans as a problem solver [Polya57]. AI uses heuristics usually at search level to reduce the complexity of search. Campbell argues that although it is true that most of the AI systems use heuristics as elements of intelligence, still, algorithms that fit one of the above three levels may be considered as to whether be in the domain of AI.

The computational level comprises problem solving techniques some of which may be common to other fields. Examples of AI techniques are representation and search techniques, some of which discussed later in this chapter.

With above arguments in mind, one may define AI in a broad sense as an area within the discipline of computer science that is concerned with the development of computational models, theories, methods, techniques, and systems that enable computers to do tasks that are judged to lie within the domain of intelligence.

3.1.1 Heuristic programming

> Heuristics! Patient rules of thumb, So often scorned:
> Sloppy! Dumb! Yet, slowly, common sense become. (ODE TO AI)

According to Webster's dictionary the adjective *"Heuristic"* means "serving to discover". "Heuristic" is derived from the Greek word *"eureka"* meaning *"I have found it!"*, and the word *"heuriskein"* meaning *"to discover"* or *"to find"*. Polya brought heuristic reasoning into focus in the last four decades [Lenat82] [Rich83]. Heuristics are defined as: The judgmental rules of thumb, educated guesses, intuitive judgments, readily accessible criteria,

methods, strategies, trick, principles, or simply common sense that help to choose the most promising course of action for achieving a goal or solving a problem by human beings or computers [Davis et al 82] [Lenat82] [Pearl84].

A program that uses heuristic rules is called a heuristic program. Heuristic rules improve the search process by guiding the problem solver in a quickest way towards a solution, but they do not necessarily guarantee the best solution. Heuristic techniques try to explore the paths in a state-space of a problem that has highest possibility of leading to goal-states. The problems attacked with a heuristic approach often have combinatorial nature, and take a long time to solve without heuristics. The arguments for heuristics are mostly based on the two factors:

- The combinatorial nature of many problems, that make a complete search through their state-space time consuming and redundant. This means that an obvious reason behind choosing a heuristic approach towards problem solving is efficiency.
- People rarely need or seek an absolute optimum solution but rather look for a satisfying solution [Simon81].

Heuristic rules are different in nature. Some are general purpose, applicable to a wide variety of problems, and some are multipurpose, suited for several problems. Other heuristic rules are specific purpose and limited to solve problems in a specific domain. General heuristics are properly attributed as *Weak methods*, as opposed to domain-dependant, task-specific ones, which are called *Strong Methods* [Langley87]. Domain-specific heuristics can be incorporated into a heuristic program in different ways. One way is to code heuristic information in forms of rules or procedures to solve the domain problems. This is the case when a direct method for solving the domain problem does not exist or is not efficient. This type of heuristic approach is named *strategic heuristic* [Georgeff83]. Heuristics also may be used in the form of *heuristic functions* [Rich83]. Here, heuristics play rôle in evaluating states of a search, and measuring the degree of success of corresponding problem solver. Such evaluation measures guide the problem solver for directing its search efforts towards the most promising paths. Some of the weak heuristic methods, such as *generate-and-test*, *hill-climbing*, and *means-ends analysis* that have been emerged from the study of problem solving will be discussed in the following sections. To tackle a

problem with heuristic search, three basic consideration must be made [Lenat82]:

- *Observability*, and possibility gathering of domain information.
- *Continuity* of environment to ensure the validity of heuristics, and
- *Stability* of environment changes over time to prolong the lifetime of a heuristic.

Empirical results show that heuristics originates from three sources [Lenat82]:

- *Specialization* of existing, more general heuristics. This is the process of adapting, binding, and matching of existing heuristics to a new more special situation.
- *Generalization* and expanding of existing domain dependant heuristics to suit a more general situation.
- *Analogy:* Creating new heuristics analogous to existing ones.

The heuristic AI approach towards problem solving has two main streams. A soft and a hard approach [Koppelaar90]. The soft approach relies on the existing body of knowledge that is either available or has to be prepared by knowledge elicitation. The existing body of knowledge is then stored in the computer programs for use. Examples of this approach are expert systems. Acquiring and incorporating all the required knowledge into a system is a time-consuming and error-prone process. Such a process requires a collaborative effort between at least two highly trained experts. A *Domain-expert* and a *Knowledge-engineer*. So, the main difficulty associated with the soft approach is the knowledge acquisition bottle neck. The main characteristic of this approach is that such programs have performance proportional to the amount of knowledge which has been stored within them.

The hard approach is based on the conjecture that if human beings can learn and discover things without prior knowledge or a limited amount of knowledge, so should do the computers. This approach tries to make the computer to perform in a higher level than its previously stored knowledge either by recreation of new knowledge or discovery of new facts. Examples of this approach are the *AM* system developed by Lenat [82] [Davis et al 82] and the *Bacon* system of Langley [87]. AM with the aid of a collection of 250 heuristics rules of thumb was able to form new mathematical concepts, and to

rediscover mathematical concepts [Davis et al 82]. Bacon was set as a test proof for automatic discovery, and for observing the process of discovery by human beings [Langley87]. Bacon rediscovered a group of previously discovered physical laws such as Proust's molecular gas constant law, Dalton's gases law, and Boyle-Gay state equation law of gases.

TOPGENE is a heuristic program that incorporates heuristics rules in its procedures to reduce the search effort. TOPGENE uses strategic heuristics during its search for a design. Heuristics used by TOPGENE are generalization of design rules either used by experts (architects) trying to optimize designs from points of view of social norms, or empirical rules relating the structure and behavior of a building with respect to the social norms. Examples of heuristics rules used by TOPGENE are given in chapter 5.

3.1.2 Machine Learning

Learning is any change or addition of new knowledge in the structure of a knowledge based system in order to improve its performance and efficiency upon a given task. Learning is classified into two basic categories of *Knowledge acquisition*, and *Skill-refinement* [Carbonell83c].

Basic forms of learning are recognized as: genetic learning, learning through direct teaching, learning by outside control and evaluation, learning by experience and observation, and learning by analogy [Carbonell83c].

Learning is also categorized in terms of the learning strategies. Several strategies have been distinguished in learning, among which are *rote-learning, instructional learning, learning by deduction, learning by induction, and learning by analogy* [Carbonell83c].

In rote learning, the basic information is accepted from the teacher, indexed and stored (memorized) by the learner. No transformation is involved in rote learning. Rote learning can further be subdivided into *learning by being programmed,* and *learning by memorization.* The first case is when a programmer program the knowledge into a computer. In the second case, which is the simplest type of learning, the knowledge or information is directly stored in a data-base. Learning by memorization does not involve new inference from stored knowledge.

TOPGENE is designed so that it can directly learn and memorize the access knowledge of building design in the form of recommended and prohibi-

ted links. The system may use this knowledge while generating new building patterns. This type of learning was categorized as rote-learning. An extension of TOPGENE may include a sophisticated type of learning such as inductive learning for accumulating knowledge of accesses in buildings based on precedents of architectural design.

3.1.3 Neural modeling

The third AI approach to problem solving is neural modeling. This approach, initiated in early 60's, suffered two decades of set-back because of criticisms by prominent researchers, such as Marvin Minsky, in the AI field,and the birth of symbolic knowledge manipulations, but has regained its acceleration in recent years.

One interesting aspects of natural neural networks is the non-local representation of information. The study of brain functioning shows that although its different parts are responsible for different activities, such as vision, speech and understanding, still, there is a large degree of non-localization in information storage in the brain. The argument is that, if a single neuron in human brains were responsible for representing an object, then, there would be several people in the world that would be normal except that they would not recognize that object. Instead human beings seem to suffer uniform degradation in functioning of their brains, specifically when they get old [Zeidenberg90].

Neural modeling tries to simulate the neural behavior of the human neural system. Human neural networks consist of roughly 10^{10} to 10^{12}neuron cells [Slagle71] [Zeidenberg90]. Neuron cells are of different complexity-types, and there are various connection-types in the brain for transfer of information. The actual interconnections and operating system of the human neural nets is not completely known. So, the correspondence between the nervous system and the current neural nets are considered only at analogical level [Durbin89]. Also, the realization of such a huge network of cells are not trivial, if not impossible!, at this moment. However, current progress in realization of parallelism in computers, and consequently the improvements in the processing speed have given new initiative to the neural network approach [Hoekstra90].

An artificial neural network has a large number of simple elements, or

artificial neurons (nerve cells) interconnected in a certain way. The main advantage and characteristic of neural networks are adaptability and learnability from experience. Neural network models of computation, whether hardwired or implemented in software, have the following elements and characteristics [Hoekstra90] [Zeidenberg90]:

- Elements (artificial neurons) are either active or not. The states of each element are expressed by a number, called *activation number*,
- There are usually two special sets of neurons, *input* and *output* neurons,
- Associated with each connection between the elements there is a *weight*, which corresponds to the strength of a synapse in a natural neuron,
- A *learning rule*, that describes the updating mechanism of the connection weights,
- An *activation rule*, which, based on the new connection weights and the past value of the activation number,describes the updating mechanism of the activation numbers, and
- A *propagation rule,* which describes the way activations are propagated in the network.

This work introduces the Hopfield neural network for solving a special case of the design problem introduced in the sequel. The Hopfield model is different from other networks in its underlaying principles and the mechanism of its behavior. An introduction of the Hopfield model, and details of the implementation of the architectural design process is given in chapter 8.

3.2 Relevant problem solving paradigms

"Problem solving" is a general term used to denote a set of actions for achieving a well-defined goal. Problem solving falls within one of the following paradigms [Carbonell83a]:

- The Newell and Simon's state-space of search paradigm,
- The McDermott and Wilensky's identifying and instantiating plan(s) of actions paradigms,
- The analogy paradigm, which considers problem solving as searching for previously solved similar problems, and transforming their

solutions into ones that are applicable to the new problems, and
- The heuristic paradigm, which emphasizes on the rôle of heuristics in searches for solutions to problems.

These problem solving approaches are not mutually exclusive, and a combination of them are often used. For example, analogical reasoning and heuristic programming both require search.

As problems are categorized under the classes of well-structured problems and ill-structured problems, problem solving techniques also may be categorized under the *Weak-methods* and *Powerful-methods.* Domain independent problem solving techniques are considered as weak methods which demand limited domain knowledge of their task environment and may lead to combinatorial explosion as the complexity of their tasks increase. On the other hand, problem solving strategies which employ domain knowledge as heuristic operators are called strong problem solving strategies [Sriram86].

TOPGENE is a heuristic program that uses domain heuristics to guide a search for a solution in the state-space of a design. The heuristic guided search in TOPGENE is planned in such a way that also resolves possible conflicts between social norms involved in a design process. The following sub-sections describe the state-space of search paradigms, and a metaphor used by TOPGENE for planning of the process of a design generation.

3.2.1 State-space of search

Problem solving, according to Newell and Simon is defined as the search for a solution through the *state-space*, also called *situation-space* [Jackson74], of the problem. Search, by its nature, is always guided by a set of rules in combination with an appropriate control strategy. The state-space of search is considered as the most popular problem-solving paradigm in AI for the following reasons, around which a considerable number of heuristic search is developed [Rich83]:

- Its correspondence to formal definition of problems as a process of converting a set of existing situations to a set of desired situations by application of some operators.
- Its power for defining a problem solving processes as a combination of known techniques with search techniques.

Search is the general technique of exploring state spaces in order to find some paths from the current state toward a goal state. A search always starts from an initial state toward a goal state by finding an appropriate path or paths. The state space is a set of all possible states in the problem solution, consisting of a set of *initial states S,* a set of *goal-states G,* and a set of *intermediate states* SI. An state space is analogous to a state transition diagram [Dougherty88]. The nodes in a state space represent the problem states and is analogous to the transition states. A link denote a relationship between two states or potential action for going from one state to the other and is analogous to an input function in a state transition diagram. The relationship between two states is defined by an operator. Operators can be considered as a set of functions F whose domain and range are the set of states of the problem [Nilsson71]. Operators, depending on the nature of the problems, can take different forms. F may be a simple function that maps a data structure, representing a state of the program, to the others; or, it may constitute computations that transforms a state description into other state descriptions, or a rewriting rule that transforms a situation into other situations [Nilsson71]. The operators are broadly classified into the following categories [Sriram87]:

- *General Operators:* Applicable to many problem domains.
- *Task-specific Operators:* Applicable to a specific domain.
- *Heuristic Operators:* Which might be domain dependent or domain independent.

Furthermore a solution path is a set of states beginning from a initial state and ending with a goal state. A problem solver is assumed to move from a state to another toward a goal state by applying an operator.
So, <SS, SI, F, SG> is a problem-space, such that:

- SS is a set $\{ss_1, ss_2, \ldots, ss_{i-1}, ss_i\}$ of start-states.
- SI is a set $\{si_1, si_2, \ldots, si_{j-1}, si_j\}$ of intermediate states.
- SG is a set of $\{sg_1, sg_2, \ldots, sg_{k-1}, sg_k\}$ goal states.
- F is a set of operators or functions required for changing the states of the problem-space.

A solution of a problem of searching a goal-state sg, starting from start-state ss is a sub-set of operators $\{f_1, f_2, .., f_n\} \subset F$, such that $sg = f_n$

$(f_{n-1}\ (f_{n-2}\ (\ \ldots(f_2\ (f_1\ (ss))))))$.

Furthermore, we can think of a set of goal paths for a particular solution k as: $\{SP_K\}$, for K = 1 to n, where, SP_k is a solution path k, or a partial solution path k. Here, $SP_k = \langle SS_k, SI_K, SG_k, F' \rangle$, where $F' \subset F$ is a set of transformation between states in SP_k.

As above definition shows, a search is very important in solving complex problems for which no direct techniques are applicable. Search techniques also allow embedment of direct methods into the search process when possible. The following steps must be taken for formulating a problem as a state space of search [Rich83].

- Define a space of all possible states of search.
- Specify the initial states.
- Specify the solution (Goal) states.
- Specify rules describing actions (Operations) for going from a state to another.

Furthermore the following issues must be considered while specifying the rules:

- Unstated assumptions in the problem description.
- Trade-offs between the generality and specificity of the rules.
- The amount of knowledge that should be pre-computed.

One should note that for a complex problem having a large state-space, often a part of the state-space is possible to be drawn, or mentioned implicitly. Similarly a problem solver trying to explore all states, or paths leading to goal-states of complicated problems can easily end-up with an unreasonable search-time. Such a problem solver has no choice but to explore most promising paths leading to solutions.

Figure 3.2 shows the state-space of a linear-tree type design problem with only four locations: L1, L2, L3, and L4.

3.2.2 Negotiation metaphor for distributed problem solving

Distributed problem solving is cooperative way of finding solutions to problems, by a number of decentralized but loosely coupled collection of problem

solvers [Davis83]. This type of problem solving is akin to viewing the process of design consisting of four global phases: *problem decomposition, sub-problem distribution, sub-problem solution, and sub-solution synthesis* [Harfmann87] described in chapter 2. A distributed problem solver presumably consists of a number of sub-problem solvers loosely coupled with each other. Davis defines the term *loosely coupled* as to mean that the individual problem solvers spend most of their time in computation rather than communication. This type of problem solving is of interest specially in cases such as a design in which there is a natural distribution of knowledge sources, and distribution of sub-tasks. Davis states his objective in developing the negotiation metaphor for problem solving as the creation of a "cooperative behavior between willing entities, rather than a compromise cooperation between potentially incompatible entities". Fundamental issues in managing distributed problem solving are [Davis83]:

- Global coordination of behavior of the system,
- A protocol for communication between the sub-systems, and
- Differed action in case of failure.

In this type of problem solving, any time that there is more than one active agent in the system, there are possibilities of mutual interference rather than mutual support between the actions of the agents. This can happen in numerous ways. There may be conflicts over the resources. One problem solver might unknowingly undo the result of the other, the same action might be repeated redundantly. etc. In general a collection of problem solvers might fail to act in a coordinated and supportive way, and as a team. This difficulty arises because each problem solver has a limited knowledge and a local view of the problem. If each problem solver had complete knowledge of the system, and consequently of the actions of the other competitors, then this would not happen. A local view of the problem, in one hand creates an easier atmosphere for the problem solver to think, and in the other hand requires a global coordination of the sub-processes. The basic motivation behind the idea of distribution is to allow sub-problem solvers to focus on specific tasks. So, if all problem solvers have universal knowledge of the problem, then this would be contrary to the basic notion of the distribution.

As the communication among any entities requires a pre-defined communi-

cation protocol, so it does the communication between a group of problem solving entities. A protocol defines what should be passed between two entities. A protocol has three basic layers [Davis83]:

- A content that identifies what should be communicated,
- A communication language, and
- A message format for negotiation.

Negotiation as defined by Davis is a type of discussion and exchange of information between interested parties in order to come to agreement upon a subject. A negotiation has three basic components:

- Two way exchange of information,
- Evaluation of information by parties of interest,
- Collective decision upon the matter.

There are two models of negotiation based problem solving: *Task-sharing* and *Result-sharing* [Davis83]. *Task-sharing* occurs in a situation in which a group of experts working on sub-tasks related to a global task, call each other when they need assistance on their sub-tasks or need for exchange of information because of lack of expertise. The alternative model to task sharing is the *result-sharing model* [Smith81] [Davis83]. In this type of approach the problem solvers assist each other through sharing of the partial (intermediate) solutions. As we will see below, this type of cooperation is more eminent in situations where incomplete knowledge of the partial solutions leads to the conflicting views at individual problem solvers.

The important components of a negotiation-based system are: A *task-manager* that is responsible for the distribution of tasks, and monitoring the executions, and several *contractors* that are the sub-problem solvers responsible for the actual execution of the sub-tasks. A contractor may in turn sub-divide a task and distribute them between other sub-contractors. A contractor in direct contact with the task-manager is called a *direct-contractor*.

TOPGENE is a negotiation based system akin to the result-sharing model. The TOPGENE's task-manager, by using an agenda of norms, iteratively passes the intermediate solutions to a number of direct contractors responsible for optimizing a partial design with respect to a specific norm. The final

solution is generated through cooperation between the individual problem solvers. In such a case passing the information about the partial solutions is vital to the continuation of the sub-problem solvers in improving the partial design towards a final global design. Negotiation is a two-way transfer of information between the task-manager and the contractors. The question of "What is to be done next?", "Who is responsible for it?", communicating the partial solutions with the sub-problem solvers, and the overall control of the process is handled by the task-manager. The messages passed between the task-manager and the sub-problem- solvers at each situation are: the partial solution, the processed locations, distance matrix corresponding to the partial solution, and partially hierarchical clusters of location-pairs.

This metaphor as it is laid out by Davis is for complicated tasks where a high degree of cooperation is required between a manager and problem solvers. A manager does not know before-hand which task to assign to which problem solver, unless it sends signals for all problem solvers (contractors) for a bid, and assign the task upon the readiness of a contractor. In addition the manager has to watch the problem solvers for their dead-lines. However, in the case of TOPGENE, the task manager knows before-hand, based on the norm on the top of the agenda, which contractor to call. TOPGENE's task-manager does not have to worry also about deadlines of the processes carried out by the task-executers when they are called for execution of a task. The task manager is notified by the task-executers when they fulfill their duty, or when they fail to do so because of difficulties that they may face.

The following figure shows TOPGENE's design generator modules, including its task-manager and its different task-executers. A detailed discussion on the implementation of the TOPGENE is postponed for chapter 5.

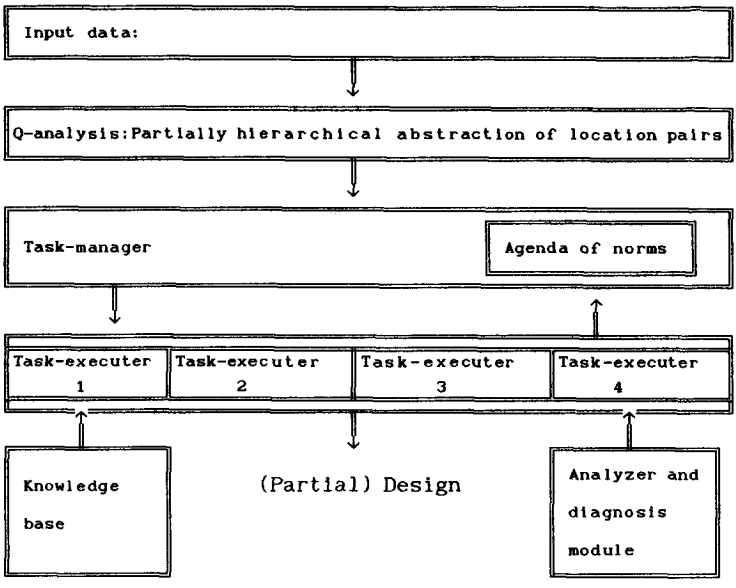


Figure 3.1: TOPGENE's design generator sub-modules

3.3 Relevant problem solving techniques

In this section problem solving techniques incorporated into the TOPGENE system and relevant to this work is discussed.

3.3.1 Generate-and-Test

A problem solver has to find one or all potential solutions to a problem. Tractable problems are trivial and soft to solve or to predict their solutions, but as intractable problem, such as a complex design, is far to difficult to predict its solution. A way of dealing with an intractable problem is to take a brute-force approach and devise a search procedure that generate all possible solutions for consideration. A system based on such an approach is called a *generative system*. The idea of generative systems can be traced back to Aristotle. Aristotle in his book *Politics,* used the following biological analogy to describe a generative system for design of a city [Mitchell75]:

> "If we were going to speak of the different species of animals, we should first of all determine the organs that are indispensable to every animal, as, for example, some organs of sense and instruments of receiving and digesting food, such as mouth and stomach, besides organs of locomotion. Assuming now that there are so many kinds of organs, but that there may be differences in them-I mean different kinds of mouths, and stomachs, and perceptive and locomotive organs- the possible combinations of these differences will necessarily furnish many varieties of animals...And when all the combinations are exhausted there will be as many sorts of animals as there are combinations of the necessary organs". [Politics, Section 1290]

Mitchell [75] gives examples of other thinkers describing generative systems throughout the history, some of whom dreamed of designing systems capable of generating the universal knowledge. Among which are the thirteenth century Spanish scholar Ramon Lull, who was thinking of generating knowledge by combinatorial art from a concentric spinning wheels inscribed with words, and German writer Kurd Lasswits who wanted to generate a library of universal knowledge by a generating system. The combinatorial idea was treated by other scholars such as the seventeenth century philosopher Leibnitz, and nineteenth century mathematician Charles Babbage. Leibnitz described the methods of invention and design by systematic generation of combinations.

A generating system was exhibited at the Cybernetic Serendipity exhibition in London in 1968 [Mitchell75]. This system consisted of a thesaurus of five short lists, and the following sentence frame with five empty slots:
I ... the ... in the ... all ... in the ... bang the ... has ....
The number of choices for each slots was as follows:

Slot number:	1	2	3	4	5	6	7
Number of choices:	5	7	8	6	8	7	13

The system was then capable of generating 1223040 (i.e., the multiplication of number of choices) different sentences by selecting 54 different words from a vocabulary and inserting into the slots of the sentence frame. The practice of combinatorial arts also is involved in classical architectural designs, when a design is approached by a systematic generation and testing of combination of elements [Mitchell75].

The Generate-and-Test (GT) problem solving technique stems from the idea of generative systems. GT in its pure form is the simplest but most inefficient of all problem solving techniques. This strategy first generates

all possible solutions (complete paths) in the search-space, and tests them for finding solutions that satisfies the goal requirements. GT can be devised in a way that generates a single solution at a time for test instead of generation of all possible solutions. GT is equivalent to enumeration techniques attributed as inefficient in design. Yet, this technique, as I will discuss it in later chapter, is perhaps a good choice if one is interested in discovery of a class of prototypical solutions for cataloging and repetitive use, for larger and more complex problems.

Figure 3.2 illustrates the search-space for a linear-tree design problem involving only four locations: L1, L2, L3, and L4. This design is expected to have a good performance in terms of the privacy norm. Furthermore, the following priority has been assumed for the location-pairs with respect to their flow generation potentials: (L1 L4), (L2 L3), and (L2 L4).

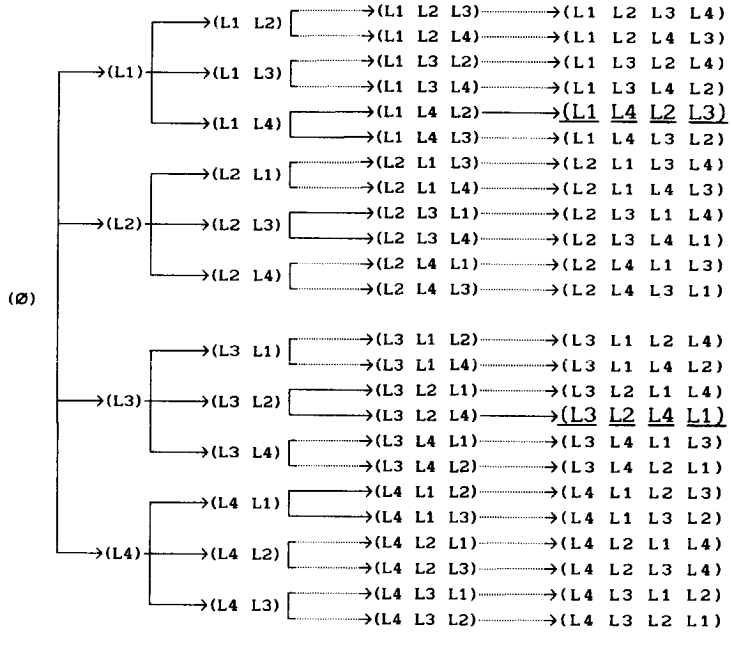


Figure 3.2: The state-space for a linear-tree type design with 4 locations

A pure GT strategy will produce a combination of all potential solutions for this problem, and then it will evaluate each to find the best solutions.

These solutions, underlined in the following figure, are: (L1 L4 L2 L3) and (L3 L2 L4 L1). Our design example shows that the complexity of its search ,under the G-T strategy, is n!. Where n is the number of locations. This means that this strategy is an inefficient strategy that quickly runs into trouble, specially for large size problems. The G-T strategy, as we will see in chapter 8, is extremely useful in cases, such as automatic discovery of solutions to small scale problems, when exhaustive enumeration of solutions is an important goal.

3.3.2 Heuristic-Generate-Test (H-G-T)

"By the pricking of my thumbs, something wicked this way comes."[MACBETH]

The example illustrated in figure 3.2 indicates the inefficiency of the pure GT technique, specially for large size problems. The number of final states in GT increases exponentially with increase in the size of a problem. To reduce the search effort in GT strategy, one possibility is the integration of heuristics with the search process. Early pruning of branches of a search tree limits the search process. For example, in the case of the design problem, posed in the last section, we can use knowledge of design in the form of heuristic rules during the search process. Here is a heuristic rule that rejects solutions which may not be optimal with respect to the privacy norm:

IF : The most interactive locations are not next to each other.
THEN : Reject the solution path.

This rule eliminates the unacceptable paths from the beginning, and some of them in the middle of the process, when they are found not attending this rule. The solid lines in figure 3.2 shows part of the state-space of search for the design problem that would be actually searched under the H-G-T strategy described above.

H-G-T may become similar to heuristic hill-climbing (H-H-C) technique used by TOPGENE in some cases. Hill-climbing is discussed in section 3.3.4.

3.3.3 Backtracking

When during a search process a choice made at a state leads to a dead-end situation, then, progress is impossible. In this situation, instead of starting from a start-state the problem solver may retract to a previous state and tries another choice. Such an action from the problem solver is called a backtracking. A backtracking is called *chronological backtracking*, if it is based on the time when the last wrong decision is made [Winston84]. A strategy based on backtracking, and specially chronological backtracking technique without the help of heuristics may be extremely inefficient specially in the case of intractable problems. For example, this strategy in the case of the design problem mentioned in the last section, simply may lead to discovery of all possible solutions, which is equivalent to a brute force approach.

In the case of previous example, a chronological backtracking with and without the help of heuristic rule will take the following courses of action, where $\longrightarrow$ represents a forward search and $\xrightarrow{b}$ denotes a backtracking. Furthermore it is assumed that the state-space of the problem in figure 3.2 is traversed from top to bottom and from left to right:

Chronological Backtracking without heuristics:

SS = (ø) $\longrightarrow$ (L1) $\longrightarrow$ (L1 L2) $\longrightarrow$ (L1 L2 L3) $\longrightarrow$ (L1 L2 L3 L4) $\xrightarrow{b}$
(L1 L2) $\longrightarrow$ (L1 L2 L4) $\longrightarrow$ (L1 L2 L4 L3) $\xrightarrow{b}$
(L1) $\longrightarrow$ (L1 L3) $\longrightarrow$ (L1 L3 L2) $\longrightarrow$ (L1 L3 L2 L4) $\xrightarrow{b}$
(L1 L3) $\longrightarrow$ (L1 L3 L4) $\longrightarrow$ (L1 L3 L4 L2) $\xrightarrow{b}$
(L1 L3) $\longrightarrow$ (L1 L3 L2) $\longrightarrow$ (L1 L3 L2 L4) $\xrightarrow{b}$
(L1) $\longrightarrow$ (L1 L4) $\longrightarrow$ (L1 L4 L2) $\longrightarrow$ SG = (L1 L4 L2 L3)

Backtracking with heuristics:

SS = (ø) $\longrightarrow$(L1)$\longrightarrow$ (L1 L2)$\xrightarrow{b}$ (L1)$\longrightarrow$ (L1 L3)$\xrightarrow{b}$(L1)$\longrightarrow$ (L1 L4)$\longrightarrow$(L1 L4 L2)$\longrightarrow$ SG = (L1 L4 L2 L3)

TOPGENE combines heuristics with the hill-climbing strategy without extensive backtracking, but two of its algorithms for generating streams of locations with respect to the norms privacy and community use look-ahead mechanisms that uses a limited backtracking. These algorithms are discussed in chapter 5.

A backtracking based approach in TOPGENE, because of the nature of our design problem, runs into full enumeration of all possible solutions. The main reason behind this claim is combinatorial relation between the loca-

tions pairs in a design which forces a backtracking strategy into the examination of all paths in the design search-space. This fact, leaves the heuristic approach as wise choice for tackling the design problem.

3.3.4 Hill-Climbing

Hill-Climbing (HC) [Pearl84] [Winston84], a modified version of H-G-T, is a simple search strategy based on local optimization techniques that tries to take the most promising course of action in each state of the search-space, by applying of a test function on the chosen states. This strategy is called Hill climbing because of its resemblance to the effort of a human in trying to climb a hill (Goal-state) in a foggy situation. H-C strategy, continuously expands a node in the search-space, test new generated nodes, and expands the most promising one, without keeping track of the previously investigated nodes. This means that this strategy is an irrevocable strategy [Pearl84] that does not allow backtracking into the previously investigated states. H-C strategy, without a good evaluation function guiding the search, has its own peculiar problems, some of which are as follows [Rich83]:

- A *local-optimum* situation in which a state pose itself better than its neighboring states, but in reality it is not better than some states farther away.
- A *horizon effect,* whereby all neighboring states show themselves the same, and a move towards a better situation is not possible.
- A *Ridge* situation in which an area better than a neighboring state is not accessible in a single move in any one direction.

Hill-climbing, however, is considered as a useful strategy when supported by a highly informative evaluation function that could prevent the problem solver from above problems, and it leads quickly towards a solution.

TOPGENE uses hill climbing in combination with domain heuristics in generating designs. The use of heuristics in TOPGENE eliminates needs for an evaluation function. TOPGENE, in each state of the search-space improves a partial solution (design) by choosing a next-state which seems appropriate to a social norm. The main criteria in choosing such a state in TOPGENE are potential flow degrees between the location-pairs in a design. The use of heuristics in TOPGENE, does not guarantee the optimality of designs genera-

ted. However, test results show that designs generated are at least in the neighborhood of an optimal solution.

The following is an example of heuristic rules used by TOPGENE to improve a partial design with respect to a social norm:

IF: - SP_k is a partial design,
- The current norm is COMMUNITY,
- (L_x L_y) is a unprocessed location-pair with maximum flow generation potential, such that: Location $L_x \in SP_k$.

THEN:- Improve SP_k by adding the link (L_y L_z) such that:L_z is the most eccentric pair with respect to L_x.

3.3.5 Means-ends analysis

Means-ends analysis is another weak heuristic problem solving method which is widely applied by human problem solvers particularly when specialized algorithms are not available for a problem [Langley87]. This method uses a type of test heuristic to detect specific differences between the start-state (SS) or the intermediate states (SI) and the goal state (SG) and evokes the moves that would reduce differences between the current state and the goal state [Winston77]. This strategy would apply to our design problem under the following conditions: Suppose that we have a number of activities for which a design is required with respect to a combination of norms. The goal of our design is to maximize the actual behavior of our design with respect the norms under consideration. In reality some norms are in conflict with each other, and all norms are incommensurable. Now, if we make strong assumptions with respect to their incommensurability. For example, if we presume numerical weights that relates norms to each other, then, we can define a globally optimum function over the combination of norms, and generate a satisfactory solution with respect to them as follows:

- Generate sub-optimal designs with respect to each norm separately.
- Analyze the behavior of all sub-optimal designs with respect to the social norms, and remember their behavioral analysis results.
- Use means-ends analysis in combination with a constraint-satisfaction system, described below, to generate a globally satisfactory design.

A final remark about the means-ends analysis is that this approach is very

much similar to the hill-climbing approach, in a way that both strategies try to attain a high quality goal-state by successive appraisal of the partial solutions. The only difference between these two strategies is that, during the process, the first strategy looks at both the intermediate and the goal-state, while the second strategy is only concerned with the intermediate states.

3.3.6 Hierarchical planning and top-down refinement

Hierarchical planning is a technique for converting complicated tasks into levels of abstract tasks that are usually simpler to work with, or to automate. Architects usually start from a abstract level, for example, by drawing a rough sketch that have no clear boundaries and dimensions toward a complete floor plan layout. Theoretically an architectural design may even start from a matrix of required adjacencies or a bubble diagram towards more sophisticated detailed levels.

Hierarchical planning also may involve knowledge abstraction. As the tasks could be hierarchically ordered into several levels of abstractions, so could be the knowledge. Knowledge abstraction takes different forms according to the nature of a task. One can, for example, consider design knowledge in terms of objects such as walls and spaces and processes that must be followed in arranging them. Or can consider is as a sequence of actions and processes that must be followed by a design problem solver. Another view on knowledge abstraction is hierarchical abstraction of data with respect to a specific criterion. Hierarchical planning, as it was described, is prerequisite for a Top-Down refinement process defined below.

Top-Down-Refinement is a problem solving process which starts from an unclear abstract level towards a more clear detailed (Down) level in a stepwise manner. This strategy may follow a model-driven or a data-driven approach. The first approach sees a problem as a conceptualization of several levels or layers of abstract sub-problems, to be solved from the top-most level towards the down-most detailed level. For example during a routine design, defined in the last chapter, TDR is seen at work by continuously refining an old design to arrive at a desired new design.

The data-driven approach is based on the hierarchical abstraction of data preceded by the hierarchical-planning strategy. Here, a Top-Down-

Refinement strategy would try to start at the most abstract level and based on the nature of the data to arrive at a detail solution. This process is obvious in theoretical design processes when designers start from adjacency requirements, and work out the problem towards the realization of a connectivity graph, the derivation of a planar graph, and consequently an actual floor plan lay-out.

TOPGENE uses both the hierarchical planning, and the Top-Down-Refinement in generating designs. The system, first, clusters location-pairs in a design with respect to their interaction potentials, in a partially hierarchical manner. In the next stem, the system uses a hill climbing strategy in combination with domain heuristics to generate a design with respect to a set of social norms.

The hierarchical abstraction of locations by TOPGENE also may be viewed as a Bottom-Up strategy towards a Hierarchical-Planning, followed by a design generation step, which is in fact a Top-Down-refinement process.

3.3.7 Deferred Commitment Principle

I refer to *deferred Commitment Principle* (LCP) as postponing the binding of a variable during a process because of lack of sufficient information, until more information is available. Deferred binding is often due to dependency of a variable in a process to the result of other sub-processes. Sriram [87] calls this technique as *the Least commitment principle.*

TOPGENE uses this principle during design in the following manner: assuming the current state of a design as PD (i.e., a partial design), a current location-pair to be processed as (L_x L_y), and the norm under the consideration as ni. (L_x L_y) must be added to PD provided that L_x or L_y have a logical relation with PD. If such a location-pair does not show a logical relation with PD, TOPGENE differs binding of (L_x L_y) to PD, and continues to bind other location-pairs with a logical relation with PD, until the logical gap between (L_x L_y) and PD is removed, after which (L_x L_y) is processed.

3.3.8 Constraint satisfaction

Design was defined as a goal oriented activity in a constraint situation. Constraint handling involves the following stages [Sriram87]:

- Constraint formulation, which is defining a new constraint on the value of a variable.
- Constraint propagation, which is the creation of new constraints by rules operating on the formulated constraints.
- Constraint satisfaction, which is the finding of appropriate value for variables that satisfying the constraints.

A constraint may take different forms and values, such as relational, numerical and non-numerical. We defined the constraints of a design as the requirements of design in general. For example, in the case of the design problem posed in this chapter, the requirements (Next-to L1 L4) and (Next-to L2 L3) are two non-numeric constraints that must be handled by the problem solver. TOPGENE as a problem solver has to lead its way towards a design solution through the paths constrained by the following requirements:

- Knowledge of recommended links.
- Knowledge of prohibited links.
- Planarity of a design solution.
- Branchiness degree of locations.
- Norms, indicating the social points of view.

Constraint-satisfaction (CS) systems use techniques such as local propaga tion, relaxation, propagating degrees of freedom, adding redundant view to the constraints, and graph transformation, for solving numerical problems dealing with constraints [Leler88].

TOPGENE tries to satisfy design requirements, discussed above, using the following strategies. These strategies in a way use a relaxation technique in dealing with requirements of a design.

- Planarity and branchiness degree of locations have priority over all other constraints.
- Recommended links have priority over all constraints except planarity.
- Prohibited links have priority over the normative requirements.
- The problem of conflicts between normative requirements is handled by agenda mechanism and relaxation technique.

3.4 Knowledge Representation (KR), its definition and role

The behavior of an intelligent entity depends on its knowledge about the environment [Genesereth87]. Such an entity has to have a representation of information prior to its manipulation. The rest of this chapter examines the rôle of representation, characteristics of a good representation system, and representation of architectural information relevant to this work.

A representation is the use of a set of syntactic and semantic conventions for describing a class of things, a set of objects, or some events [Winston84]. The syntax of a representation system is specified by a set of rules which describe the construction of expressions in the representation language from a set of symbols. The semantics specify the interpretation or meaning of expressions.

Knowledge representation (KR) is the invention of an artificial language for capturing the important aspects of a knowledge domain, and representing them in a modular and suitable form for manipulation by known techniques or intelligent machines. A representation system, with regards to above definition, then, consists of:

- A vocabulary or language for naming things.
- A set of operations that can be performed on things.
- A syntax (data structure or formalism) for capturing and encoding descriptions of things.

The word "representation" is understood both as structuring of knowledge, and also capturing of them in a data structure for computer manipulation. Considering the fact that the value of a representation is different in the eyes of men and the machines, one has to distinguish between the *human level* and the *machine level* of representation while comparing representation techniques. These two levels of representation are distinguished by Charniak and McDermott [Charniak85] as *abstract level* and *concrete level* of representation respectively. So, we can attribute three levels to any knowledge domain.

- A fact level which is the state of being.
- An abstract level representation, which is the capturing and representation of content of a knowledge domain into a set of expression for human use.
- A concrete level representation, which is the data structure chosen

for indexing and storage of the abstract level knowledge within the computers.

KR also may be looked at in a broader context for having the following rôles [Fisher87]:

- *Knowledge interpretation:* Representation has a kind of interpretive rôle. This rôle is more evident in representation of sensory information, specially when they are to be processed by computers.
- *Knowledge organization*: Representation helps structuring knowledge of a specific domain. Knowledge structuring helps better understanding of the nature of objects, events, and relationships between them.
- *Deduction power*: Representation methods, such as predicate logics, provide the basic ground for automating reasoning processes.
- *Modeling processes, and predicting the behavior of real world*: Representation of objects and events permits modeling of real world processes, *and* predict the behavior of the real world objects.

There is not a strict universal guideline for choosing a representation system for representing the knowledge of a specific domain. The choice of a representation depends completely on basic criteria of a domain such as structure of the data, and nature of its search space [Jackson86]. The data structure for representing knowledge in a computer is usually chosen in a way that the storage and manipulation of the knowledge are as efficient as possible. Furthermore, Since a representation technique might not be capable of capturing all aspects of a knowledge domain, usually a wide spectrum of representation techniques are needed in dealing with real world problems. This is evident from different representation schemes that are needed at different levels of an architectural design (figure 3.3).

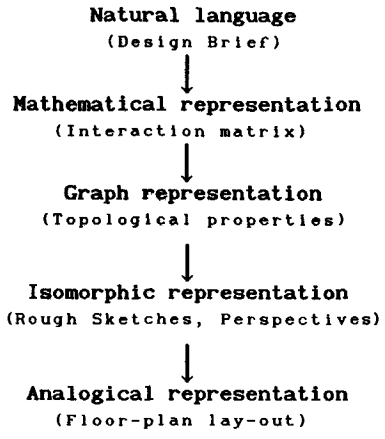


Figure 3.3: Representation systems used in different levels of architectural design

This work is concentrated on design at connectivity level. The most natural representation of connectivity properties of a building is graph. Graphs hire different techniques for its implementation at machine (concrete) level. In the rest of this chapter after reviewing theoretical concepts around the issue of knowledge representation, some of the existing representation schemes related to this work is reviewed. However because of the relevance of this work to the representation of objects and their relations as graphs, the main emphasis of this section is on graphs as a means of representation.

3.5 Criteria for a good representation method

Representation of knowledge is an important issue in the development of any computer system simulating real world events or imitating the behavior of real world objects. AI since its birth has paid special attention to this issue. Following, you can find some of the characteristics set for a good K.R. system. Some of these characteristics may overlap and some contradict others.

- *Explicitness:* One must be able to represent important aspects of knowledge domain as explicit as possible. Explicitness is considered

as must important feature of a knowledge representation method at least within the AI field[1].

- *Expressive adequacy:* also called *epistemological adequacy* [McCarthy-69], *completeness,* and *logical adequacy* [Jackson86], is the power of a representation language in exposing and capturing the important aspects of the things that one wants to represent. This characteristic is important in knowledge manipulation.
- *Reasoning adequacy*, also called *Heuristic power* and *Computability* [McCarthy69] [Rich83] [Jackson86] means that a good representation language must provide sufficient ground for reasoning processes, and problem solving in general.
- *Reasoning Efficiency [Rich83]:* A representation language must provide efficient ground for knowledge storage, knowledge retrieval, and knowledge manipulation. This criteria often contradict the expressiveness.
- *Uniqueness:* A representation system must not lead one to any ambiguities while encoding the knowledge. Charniak and McDermott [Charniak-85] distinguish two types of ambiguities, which must be avoided while encoding a knowledge domain into a representation system: The referential ambiguity and word-sense ambiguity. The first type of ambiguity arises when a referent in a sentence is not clear and leads to uncertainty as to which object it is referring to. This type of ambiguity is avoided by giving object of the same type and with similar names unique names. Such a unique name is called an *instance* or *token.* The word sense ambiguity arises due to multidimensional meaning of words in natural language. Word-sense ambiguity is removed from a representation language by choosing different term for different meaning of such words.
- *Stability:* A representation system must be stable in a sense that a small change in the domain knowledge does not cause a large change in its syntax.
- *Brevity and conciseness:* While one must be able to suppress and keep

[1] "What makes AI systems knowledge-based is not that somehow takes knowledge to write them, nor just that they behave as if they had knowledge, but rather that their architectures include explicit knowledge bases -more or less direct symbolic encoding of the knowledge in the system." [READ85]

details of the domain knowledge out of the sight and represent matters as efficiently as possible, but the detail must be accessible when they are necessary.

- *Consistency:* A good representation is expected to be consistent and not to lead to any contradictions in representing different aspects of a knowledge domain.
- *Transparency [Read85]:* A representation system must provide access to the implicit facts stored in a knowledge base.
- *Acquisitional Efficiency* [Rich83]: It must provide an easy (i.e., automatic) way of knowledge acquisition and insertion into the knowledge base.
- *Notational convenience* [Jackson86]. A good representation system must have an easy to write and read syntax, and an easy to understand semantics. This means that the resulting expressions from the language must be understandable by a human despite of their interpretive meaning to machines.

In practice, trying to find a representation scheme for a particular task that has an even balance between these characteristics is a difficult task. The choice of a representation is always under the control of nature of the knowledge domain, and the processes that must be applied to them.

TOPGENE, is an example of this situation. TOPGENE, as a system simulating the social behavior of buildings, takes advantage of several existing algorithms developed around graphs theoretical concepts. Each algorithm demands its own representation scheme. So, it was impossible to think about developing or choosing a single representation method capable of fulfilling all the system requirements. For this reason, TOPGENE uses several representation schemes for capturing and manipulation building information.

3.6 Declarative vs. Procedural representation

The choices of representation methods for representing knowledge at any level of abstraction are not too many. Current computers employ a handful of knowledge representation methods. Any book on knowledge representation will not discuss the matter beyond the categories such as: graphs, state-transition graphs, frames, procedures, predicate logic, if-then inference

rules, and a few more. Some of these methods, such as graph and isomorphic representations [Fisher87] are mostly used for representation of knowledge on abstract level. The representations of knowledge at concrete level (implementation level) often requires different representation methods than those at abstract level.

The approaches in representing knowledge for intelligent systems are categorized into two basic categories: The *declarative* method and *procedural* method. Examples of declarative methods are *frame representation* and *predicate logic* mentioned above. Procedural representation, the counterpart of declarative representations, is the embedment of knowledge in programming languages in the form of procedures. Knowledge involving performance of sophisticated tasks could be represented by procedures that describe accomplishment of those task in a step wise algorithmic manner.

The trade-off between the two representation techniques has been a controversial subject within AI community for sometimes. Some believe that there is not a clear distinction between these two types of representation, as it is not known sometimes whether a piece of knowledge is a program or a statement. Both approaches often seem of equal value for computer systems. Programming languages such as LISP and KRL have the benefit of combining both of these approaches [Rich83]. The following table shows different views on "What is knowledge?" [Winograd75]:

What is knowledge?

Proceduralists view: Papert, and Minsky	Programmist view: Hewitt, Bishop, and Steiger	Declarativist view: McCarthy
"Knowledge is: Knowing How.".	"Everything is: Program."	"Knowledge is: Knowing What.".

Figure 3.4: Different views on what is knowledge?

Both categories have some advantages over the other. Declarative knowledge representation is more modular and hence more flexible but process-wise more expensive than the procedural representation of knowledge, while procedural representation is less flexible but more efficient. Representation of facts or objects in frames or representation of beliefs in logic systems in decla-

rative form cannot accomplish any task without their manipulation by some type of procedures. Consequently one should consider both systems necessary for any real application. As Rich [83] states:

> "No system can survive exclusively on declarative knowledge, with no procedures for manipulating what it knows. ... A procedural representation of a piece of information is essentially a plan for the use of that information. Thus constructing a good procedural representation is similar to constructing any other type of plan. Because of this, work on procedural knowledge representation is closely interwoven with work on plan generation. " [Rich83, pp 240-241]

The advantages of both types of KR can be summarized as follows [Winograd75] [Rich83] [Genesereth87]: The advantages of declaratives:

- They are usually flexible and multipurpose. They may be used for various lines of reasoning.
- They are extendible beyond their status, by application of reasoning processes (i.e., derivation of additional knowledge).
- They may be changed easily, while a small change in procedural knowledge could have devastating consequences.
- They are economical, because of possibility of multiple use.
- They are understandable and learnable because of their simplicity, relative to procedural knowledge.
- They are accessible and easily communicable.
- They are easy to modify, or add to.

The benefits of procedurals:

- Modeling of a process in procedural form is usually easier than in declarative form.
- Second Order Knowledge, such as a plan of actions, are better represented by procedures.
- Heuristic knowledge could be easily integrated into procedures for deep deduction reasoning. Declarative systems do not allow such a flexibility.
- It is easier to represent knowledge that is not easily representible in declarative forms.
- A procedural knowledge is computationally less costly and more efficient than the declarative ones.

TOPGENE is a deep reasoning system minimally depending on declarative knowledge of architectural design. The main knowledge of architectural design related to TOPGENE is captured in procedures rather than knowledge bases. TOPGENE, however, uses a limited knowledge of design at connectivity level in declarative form, stored in knowledge bases. This is the knowledge of prohibited and recommended accesses in buildings. The connectivity property of buildings are best representible as graphs. TOPGENE uses several implementation of graphs as demanded by its algorithms. The following sections discuss graphs as a means of representing information.

3.7 Graphs as a means for declaring topological information

Graphs are convenient means for representing a set of objects and relationships between them. Graphs have wide variety of applications, and take different forms in representing information. Visually, graphs are represented in the form of nodes and links, with nodes representing objects and the links representing relations between them. Implementation of a graph for algorithmic operations by a human or a computer may take different forms. Several techniques have been developed for capturing basic structural properties of graphs. Following sections are devoted to these techniques.

3.7.1 Adjacency matrix

Matrices are convenient way of representing basic properties of graphs. A graph G with n vertices can be associated with an n by n (vertex) adjacency matrix A(G) [Christofides75]. An adjacency matrix captures the adjacency properties of the nodes of a graph[2]. An entry a_{ij} of A(G) is a binary value such that:

$$a_{ij} = \begin{cases} 1 & \text{If: } (i\ j) \in E(G), \text{and} i \neq j. \\ 0 & \text{Otherwise.} \end{cases}$$

[2]Geometrically, a graph is defined as a set of points (also called vertices or nodes) interconnected by a set of, not necessarily distinct, lines (also named links or edges). A graph G is denoted by: G=(V, E), or G = (G(V), G(E)), where, V and G(V) denote the set of vertices, and E and G(E) represent the set of edges in G.

If G is undirected, and contains no loops, then A(G) is symmetrical about the diagonal. Here, the transpose of the A(G) leaves it unchanged. That is to say $A(G) = A^T(G)$. If G is a weighted multi-graph (i.e., a graph with at least a node-pair with multiple edges), then, the entries of the A(G) are defined as:

$$a_{ij} = \begin{cases} k & \text{If and only if } (i\ j) \in E(G), \text{ and } (i\ j) \text{ is k weighted.} \\ 0 & \text{otherwise.} \end{cases}$$

$$a_{ii} = \begin{cases} h & \text{If there is a loop of a weight h at vertex i.} \\ 0 & \text{otherwise.} \end{cases}$$

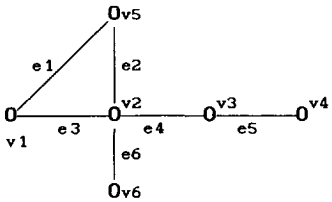
The adjacency matrix representation of a graph G reflects many structural aspects of a graph. For examples [Bucklay90]:

- A graph is connected if and only if one cannot re-label its vertices such that its adjacency matrix has the following form, where A_{11} and A_{22} are square.

$$A = \begin{bmatrix} A11 & 0 \\ 0 & A22 \end{bmatrix}$$

- If A^n is the n-th power of the adjacency matrix, then the $a_{ij}^{(n)}$ entry of A^n is the number of paths of length n from v_i to v_j.
- $d(v_i)$, the branching degree of v_i in G, is equal to the $a_{ii}^{(2)}$ entry of A^2.
- The distance between two distinct vertices v_i and v_j in G is the least integer for which $a_{ij}^{(n)} > 0$.

A graph can be represented in computers in array (matrix), linked list or simple list data structures. A graph with n vertices has n^2 storage complexity if it is represented by an array data structure. Here, if the graph is sparse and the number of edges are far less than n(n-1)/2, then , it is not economical to represent it as an adjacency matrix, and it will be better off if it is represented as an adjacency list, to be discussed later. However, to compare the representation of a graph in an array with other representation techniques, let's remember that if a graph is represented as an array data structure, then, a search for a link $(v_i, v_j) \in E$ takes one step, and a scanning of neighbors of a vertex v_i, despite of its degree $d(v_i)$, takes n steps to complete. The following figure depicts the graph and corresponding adjacency matrix



	v1	v2	v3	v4	v5	v6
v1	0	1	0	0	1	0
v2		0	1	0	1	1
v3			0	1	0	0
v4				0	0	0
v5					0	0
v6						0

Figure 3.5: A undirected graph and its corresponding vertex adjacency matrix

Adjacency matrix has important rôle in representing the connectivity and adjacency properties of floor plan layouts. Several exiting algorithms for calculating distances in a graph use this data structure. TOPGENE uses both the access and distance information for reasoning about structure and behavior of a building with respect to the social norms.

3.7.2 Edge adjacency matrix

The adjacency property relating edges of a graph G also may be captured in a m by m edge-adjacency matrix EA(G). Here, m is the number of edges in G.

$$ea_{ij}(G) = \begin{cases} 1 & \text{If: the edge ei is edjacent with enother edge ej.} \\ 0 & \text{Otherwise.} \end{cases}$$

3.7.3 Incident matrix

A graph G with n vertices and m edges can be represented as an n by m incidence matrix. The rows and columns of such a matrix, also called edge adjacency matrix, correspond to the vertices and edges of G, respectively. The entries of an incident matrix T(G) are defined as:

$$t_{ij}(G) = \begin{cases} 1 & \text{If the vertex vi has the edge ej incident with it.} \\ 0 & \text{Otherwise.} \end{cases}$$

The incidence matrix corresponding to graph of figure 3.5 is shown below:

	e1	e2	e3	e4	e5	e6
v1	1	0	1	0	0	0
v2	0	1	1	1	0	1
v3	0	0	0	1	1	0
v4	0	0	0	0	1	0
v5	1	1	0	0	0	0
v6	0	0	0	0	0	1

Figure 3.6: Incident matrix representing graph of figure 3.5

A directed graph also may be represented by an incident matrix. The entries to such a matrix will then be as follows:

$$t_{ij}(G) = \begin{cases} 1 & \text{If the vertex } v_i \text{ has the edge } e_j \text{ incident with it.} \\ -1 & \text{If the vertex } v_i \text{ has the edge } e_j \text{ incident from it.} \\ 0 & \text{Otherwise.} \end{cases}$$

The sum of the elements of T(G) representing an undirected graph G is equal to twice the number of edges in G. This, in turn, is equal to the sum of all vertex degrees in G.

$$\sum t_{ij} = \sum d(v_i) = 2m$$

The line graph of a graph G, denoted by L(G), is a graph derived from G in such a way that each vertex in L(G) represents an edge in G, and each edge connecting two vertices in L(G) denotes the adjacent edge in G. The number of vertices (V(L)) and the number of edges (E(L)) of a line graph L(G) of G can be calculated by the following relations [Trinajstic83a]:

$$V(L) = E(G),\ E(L) = \left(\frac{1}{2}\sum_{i=1}^{n}[d(V_i)]^2\right) - E(G)$$

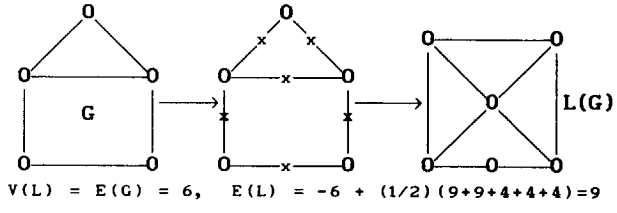


Figure 3.7: A graph G and its line-graph L(G)

The edge-adjacency matrix of a graph G is identical with the vertex-

adjacency matrix of the line graph L(G) of G. The reason is that the edges in G are replaced by vertices in L(G). Line graphs have application in capturing the wall property of building in architectural design.

3.7.4 A Dual representation

Adjacency and incidency are two views of a graph. The storage complexities for representing a graph of n vertices and m edges as an adjacency matrix or an incidence matrix are n*n, and n*m, respectively. If the number of nodes in a graph is relatively large and the representation of both views is demanded, then separate representations of them are redundant. In such a case, and if possible, the combination of two views and their representation in a single matrix reduces the space complexity almost in half. Figure 3.8 shows an n by m adjacency-incidence matrix representing the graph of figure 3.5. In this matrix a 0 entry represents a null relation, a 1 entry represents an adjacency relation, a 2 entry represents an incidency relation, and a 3 represents the adjacency and incidency relations both. This representation is only suitable for simple (i.e., Graphs without multiple edges) unweighted graphs, but with some modification the adjacency-access properties of other types of graphs also may be captured within the same matrix.

	v1	v2	v3	v4	v5	v6
	e1	e2	e3	e4	e5	e6
v1	2	1	2	0	1	0
v2	1	2	3	2	1	3
v3	0	1	0	3	2	0
v4	0	0	1	0	2	0
v5	3	3	0	0	0	0
v6	0	1	0	0	0	2

Figure 3.8: A Dual representation of graph G in figure 3.5

3.7.5 Adjacency lists

A graph also can be represented within computers as an adjacency list. Each adjacency list consists of a head element representing a vertex $v_i \in V$ and a number of succeeding tail elements representing its neighbors (i.e., $N(v_i)$). The space required for representing a graph G(E,V) with n vertices and m edges as an adjacency list is in the order of n + m. This seems more econo-

mical than the adjacency matrix if a graph is sparse. The space requirement for adjacency list representation of a graph G is then:

$$O(\sum_{v \in V} [1+d(v)]) = O(n+m), \text{ where } d(v) \text{ is the vertex degree of } v.$$

The scanning of the neighbors of a node of G represented by adjacency lists and search for a link, both requires d(v) steps. This means that in comparison with array representation, adjacency list is better in terms of search strategies. An adjacency list (and other representations) could be implemented in several ways. For example, if one has to implement graphs in procedural languages without any list processing power, then the logical relation between the graph nodes has to be explicitly taken care of by the programmer. The following figure illustrates the linked list representation of the graph G in figure 3.5. The arrows, in the following figure represent the address links (logical relations) between the neighbor vertices.

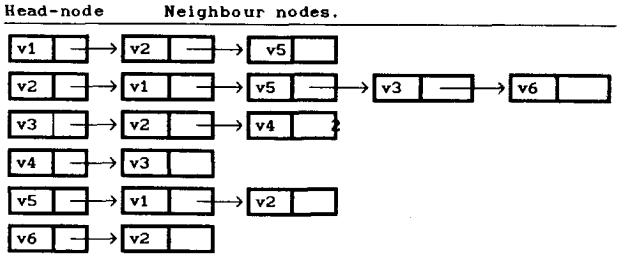


Figure 3.9: An adjacency list representation of graph G in figure 3.5

3.7.6 Association list and property list

The connectivity property of a graph may be captured in various implementation schemes, depending on the computer languages that are used. In LISP, for example, one can implement a graph as an *Association-list* or *Property-lists*. An association list, also called *a-list,* is a list consisting of pairs of associated elements. Each pair in an association list is a pair which its first element is called a *key* and the second element is called a *datum*. An association list has a name which serves as its addressing for manipulations. A function for constructing an association list, in common LISP language, has the following format [Steele84] [GoldWorks87] [Winston-89]: (SETF a-list-name '(<key datum>$^{+}$)), where, + sign means unlimited repe-

tition of the <key datum>.

An association list corresponding to the connectivity property of graph G in figure 3.5 is:

(SETF GRAPH-G6 '((v1 v5) (v1 v2) (v2 v5) (v2 v6) (v2 v3) (v3 v4)))

An item in an association list can be retrieved by applying the ASSOC function. ASSOC retrieves only the first pair with first item as its element. This function has the following format: (ASSOC item a-list). For example: (ASSOC 'v2 GRAPH-G) returns (v2 v5) as its value. However, since most of the algorithms on graphs work on the neighborhood property of graphs, one often needs to have a quick access to the list of neighbor vertices of a node. An association list representing connectivity property of graph G is depicted in figure 3.10:

```
(SETF GRAPH-G6'( (v1 (v2 v5))
                 (v2 (v1 v5 v3 v6))
                 (v3 (v2 v4))
                 (v4 (v3))
                 (v5 (v1 v2))
                 (v6 (v2))
               ) )
```

Figure 3.10: An association list representation of Graph G in figure 3.5

Now, the command (ASSOC 'v2 GRAPH-G) returns (v2 (v1 v5 v3 v6)), and the command: (SECOND (ASSOC 'v2 GRAPH-G)) gives (v1 v5 v3 v6), the neighboring node for v2.

An association list also can be updated by adding a new association pair to the list or by replacing an old pair with a new one. The function ACONS and RPLACD can do the addition and replacement job respectively.

(ACONS key datum a-list)

(ACONS 'v3 '(v1 v2 v4) GRAPH-G)

(RPLACD (ASSOC key assoc-list) new-value)

(RPLACD (ASSOC 'v1 GRAPH-G) '((v3 v6 v7))) ==> (v1 (v3 v6 v7))

Property list

Property list, also called *plist*, is another data structure capable of capturing graph properties, such as access and neighborhood. Property list is very much similar to association list. A difference between a property list and an association list is that: objects within a property list have a

unique identity, and when they are modified their old values are destroyed; while, an association list can be augmented with a new (key value) pair without destruction of an old pair identical with the new one. A function for searching and constructing property lists in common LISP have the following templates:

(GET <object-name> <property-name>)

(SETF (GET <object-name> <property-name>) <property value>)

Where, GET is the lisp primitive for returning the property of an object, and SETF is another primitive which in combination with GET allows to assign a particular property to an object. Figure 3.11 shows the representation of neighborhood properties of the graph G in figure 3.5.

```
(SETF    (GET 'v1  'neighbors) '(v2 v5)
         (GET 'v2  'neighbors) '(v1 v5 v3 v6)
         (GET 'v3  'neighbors) '(v2 v4)
         (GET 'v4  'neighbors) '(v3)
         (GET 'v5  'neighbors) '(v1 v2)
         (GET 'v6  'neighbors) '(v2)
)
```

Figure 3.11: A property list representation of Graph of figure 3.5

(GET 'v2 'neighbors) returns (v1 v5 v3 v6), the neighboring nodes of v2. For further detail on functions manipulating association list in common lISP language see [Steele84] and [Winston89].

TOPGENE, additional to the matrix representation, uses property list representation of topology of buildings. A representation was defined as a kind of data structure for algorithmic manipulation. It is hardly possible to find a single representation scheme that is expressive enough for capturing a variety of properties of an object, or highly adequate in providing for a number of algorithms a manipulating means. This was also the case for above representation schemes devised for capturing graphs properties. TOPGENE is a system that requires a relatively large number of graph algorithms, each with its own demand for a particular data structure. As a result, TOPGENE has to keep several representations of connectivity property of building simultaneously or at different levels of its reasoning processes. For example, adjacency matrices provide a basis for calculating topological distances in a building, but fails completely to serve as means for finding

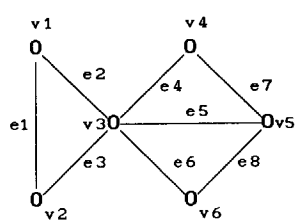
paths between the topological locations in a building. TOPGENE, thus, depending on its algorithmic demands, keeps vertex adjacency representation, property list representation, and association list representation of the connectivity property of a building at the same time or at different times as they are required.

3.7.7 Distance matrix

A distance matrix D(G) for a graph G is a symmetric n by n matrix, in which an entry $d_{ij}(G)$ in it represents length of the shortest path between the two vertices v_i and v_j in G. The length of a path is the number of edges on the path. All diagonal elements in D(G) are by definition zero. Such a matrix for an undirected graph is naturally symmetrical on diagonal.

$$d_{ij}(G)=\begin{cases} k & \text{Where, k is the number of edges on the track between } v_i \text{ and } v_j. \\ 0 & \text{If i=j.} \end{cases}$$

The following figure shows an undirected graph and its corresponding distance matrix.



	v1	v2	v3	v4	v5	v6
v1	0	1	1	2	2	2
v2		0	1	2	2	2
v3			0	1	1	1
v4				0	1	2
v5					0	1
v6						0

Figure 3.12: A graph and its distance matrix.

The following relation exists between the distance and adjacency matrices.

$$D(G) = \sum_{i=1} A_i(G) = A(G) + \sum_{i=2} A_i(G)$$

where, $\sum_{i=2}^{N} A(G)$ represent matrices containing the shortest paths between the second, third, ..., etc. neighbors [Trinajstic83a].

The distance matrix has applications in calculating well known topological indices, such as Randic's index in chemical topology, in generating access patterns of building, and evaluating buildings with respect to some

social norms. TOPGENE uses distance matrix to calculate and keep track of distances in partial designs.

3.7.8 Circuit matrix

The circuit (cycle) matrix C(G) of a graph G is a c by e matrix, where c is the number of circuits and e is the number of edges. The circuit matrix is defined as [Trinajstic83a]:

$$C_{ij}(G) = \begin{cases} 1 & \text{If the i-th cycle contains the j-th edge.} \\ 0 & \text{Otherwise.} \end{cases}$$

Figure 3.13 shows circuit matrix for the graph G in figure 3.13.

Cycles:		e1	e2	e3	e4	e5	e6	e7	e8
C1={e1, e2, e3}	C1	1	1	1	0	0	0	0	0
C2={e4, e5, e7}	C2	0	0	0	1	1	0	1	0
C3={e5, e6, e8}	C3	0	0	0	0	1	1	0	1
C4={e4, e6, e7, e8}	C4	0	0	0	1	0	1	1	1

Figure 3.13: The circuit matrix representation of the graph G in figure 3.13

A circuit matrix C(G) of a graph G, has the following orthogonality relation with the transpose of its incidence matrix T(G) [Trinajstic83a]. $C(G) * T^T(G) = 0$ (Modulo 2). That is to say, the elements of resulting matrix are either 0 or divisible by 2.

TOPGENE does not use a circuit matrix representation of topological property of a building, but this matrix may be of use in an extended version of TOPGENE that would include the allocation problem discussed in chapter 2. Such a matrix, then, may have rôle in holding information on the circular paths in a design.

3.8 Notes on representation of domain information

The richness of the graph theory, and possibility of representation of many topological properties of buildings in graphs, also existence of graph manipulation algorithms, poses it as a valuable choice for TOPGENE's implementation. The following sections covers notes on representation of buil-

dings' properties in graphs as is approached by TOPGENE.

3.8.1 Representation of activities and actors

In implementing a system capable of making inference based on the structure and behavior of a building, one has two choices in dealing with the allocation of activities and actors responsible for the activities in a building. To bound each actor to a location, and assume multiple presence of the activities in several locations, or to assign each activity to a location and assume multiple presence of actors in different locations. The first assumption is unrealistic, since in practice activities are bound to locations, and in reality locations in a building are for a purpose (i.e., activity) to serve.

Assuming that each location is assigned to an activity, then, actors have to attend the activity(ies) which they are assigned to. Now, if some actors are responsible for multiple activities, then, they are supposed to have to move between locations related to their activities. The movements of actors in a building create interactions and consequently cause a building to behave socially in certain ways.

TOPGENE assumes that the activities are bound to locations. In such a case, a location is identified by an activity. To be more specific, given an activity A_x, and actors G_i, G_j, G_k responsible for that activity. First, since the locations are bound to activities, so, there is no need for separate presentation of the locations and activities. A location, say L_x, is represented by the allocated activity A_x to that location. Now, the actors responsible for an activity, such as A_x, may be represented simply as a property list as follows:

(SETF 'A_x (GET 'actors '(G_i G_j G_k)))

3.8.2 Representation of topological information

Connectivity property

Architects present floor-plan layouts of buildings in two dimensional Euclidean space. This form of representation, called isomorphic representation [Fisher87], is what architects produce at the last stage of a design. The

reason is that, for a complex architectural design, a systematic approach starts at an abstract level, such as the topological level, and works out from top to bottom until arriving at a floor-plan layout. Graphs have been widely used for representing properties of buildings such as their adjacencies and accesses. An access graph represents the contiguity (existence of door or other means of access) of locations within a building, while an adjacency graph represents the boundaries of locations. The adjacency graph of a building is planar and has vertices corresponding to the locations and the links corresponding to their access or contiguities. The adjacency property of a floor plan represents the existence of boundaries between different locations of a building. The nodes and links, in such a graph, correspond to the locations and the common walls between them respectively.

Note that the access graph for a particular floor plan is a spanning sub-graph of the adjacency graph of corresponding floor plan. Such a graph is also generally a connected spanning sub-graph. The adjacency graphs are similar to what architects call "Functional" or "Linkage" diagram [Steadman-83]. Linkage diagram is used in the early stages of the design process by some designers for functional analysis of the building. The adjacency relationships between the locations of a building are significant, since when there is a sufficient wall sharing between two locations then the placement of doors and windows are possible. The following two figures shows a floor-plan and its corresponding access (connectivity) and adjacency graphs.

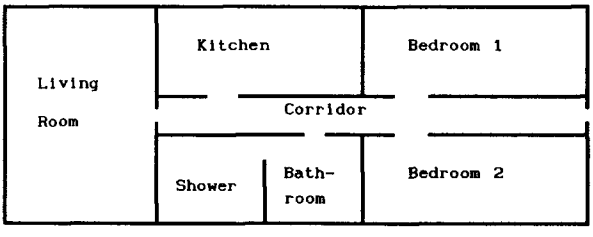


Figure 3.14: An architectural floor plan

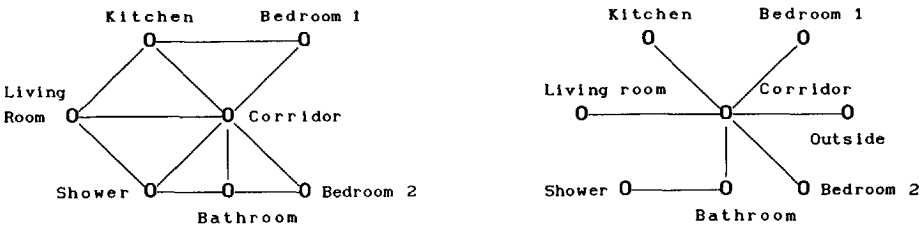


Figure 3.15: The adjacency and access graph corresponding to the floor plan layout of figure 3.14

Special cases of adjacency and access

There are special cases of access and adjacency relations which are realistic and actually exist in *some* buildings. The first case arises when a location completely encircles another location, and it has been partitioned in a way that it has access or has a common wall with itself. The graph of such a situation has a loop in the node corresponding to the room that has access to itself.

Another special case arises when two locations, such as A and B in above example, have more than one way of direct access to each other, or share more than one common wall. This case is realized only if location B is interpreted as one location. Such a case in graph notation can be represented by multiple edges, or case can be handled by taking the location B as consisting of two locations B and B'. These cases are illustrated in the following figure:

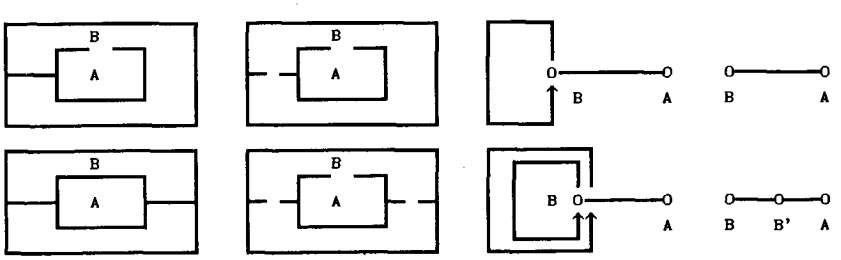


Figure 3.16: Two special cases of access/adjacency and their corresponding topological interpretations

TOPGENE only deals with the connectivity (access) property of buildings. The multiple loops and linkages between locations of a building do not have any implication in terms of the social behavior of a building because of its structure-operation relationship. For this reason, TOPGENE is careless about loops and multiple links in buildings.

Parcels of locations

A location is a space or place allocated to an activity or reserved for a purpose. A location might be a room, a long corridor, a hall, a place for a small object such as canteen, or an eating area. We can think of hierarchical levels of locations. For example, an eating room consisting of several sub-locations could be identified and taken together as a single location. TOPGENE does not attempt to subdivide a location into several sub-locations, or to aggregate a group of locations into a single location while generating a design. The number of locations a design is always identical with the number of declared activities. The exception is design with auxiliary locations such as corridors, in which case such locations are added to a design to produce prototypical designs such single-loaded or double-loaded designs. The decision on whether a group of locations should be considered as a single location or not, in the case of the evaluating an existing design is also with the user. A group of related locations (activities) may be presented as a single location. In such cases, actors responsible for the aggregated activities also must be aggregated and presented to the system.

Corridors

A corridor is a continuous, long, short, large, or small space in a building which serves as an accessory to the people residing, working or visiting a building. A corridor connects certain locations of a building and serves people as a link choice for making decision on where to go, and in passing through and reaching from one location of a building to the others. TOPGENE calls corridors and similar locations without any activity assigned to them the *auxiliary locations*. A rather large auxiliary location, such as a corridor, in reality, can be considered as a single location, but practically representing a long corridor with relatively large number of rooms having access to each other via that corridor as a single location can have serious implications and impact on the behavior of a building with respect

to the social norm, or evaluating an existing design. For example, in evaluation of a building with respect to a set of social norms , representation of a large corridor as a single location may result in the loss of the semantics, and as a result, false evaluation of the building. While, taking and representing parts of a corridor, having significance in terms of the linkage that they provide between different locations, as separate locations, eliminates the problem. Examples of this situation are the single-loaded and double-loaded type buildings, in which parts of a corridor in front of a residential location may be taken as a separate auxiliary location. TOPGENE leaves the choice of representation of auxiliary locations as an aggregated or divided locations with the user.

The following figures show a double-loaded design and two access patterns corresponding to this design. One with the corridor taken as a single location, and the other taken as a chain of successive locations.

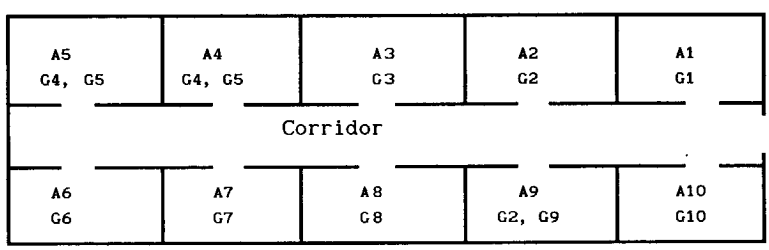


Figure 3.17: An architectural floor plan with double loaded corridor

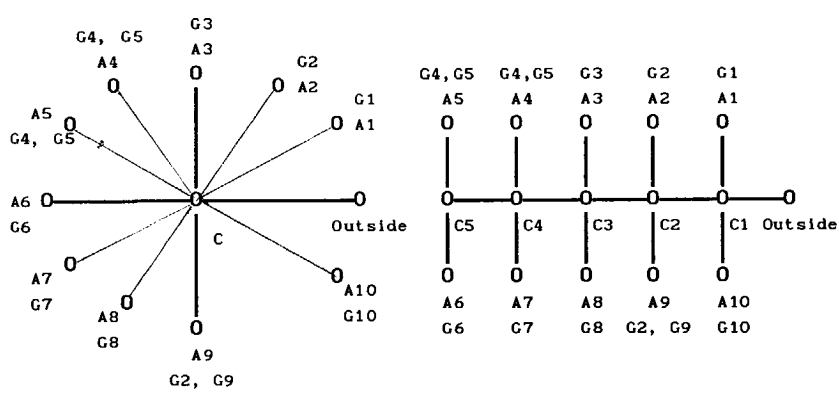


Figure 3.18: A star and a double-loaded interpretation of topology of floor plan lay-out depicted in figure 3.17

In these figures the As represent the activities and Gs the actors responsible the activities. The star-shape representation is, first, unrealistic in terms of architectural design since it might represent a building with a round corridor with rooms around it, and second, its implication in terms of the social aspect of the building is not the same as the other one. Since, if we compare these two designs and try to imagine the patterns of flow on them as a result of operation of the building, we can expect different behaviors from them. This can be examined by looking only at a few location-pairs and the interactions between them. For example, if we take two location-pairs (A2 A9) and (A4 A5), each of them shares the same actors. As a result we can expect interactions and flow between each location-pair. These patterns of flow in the case of the star type pattern cross each other at location C, and thus contribute to the social behavior of the building, while in the case of the doubley-loaded pattern the flows between these location-pairs may not cross each other. The second case, obviously, reflects a more practical design with a more realistic organization. An evaluation of the behavior of corresponding building based on the second pattern yield a more realistic result than the first one.

CHAPTER 4
DIAGNOSTICS BY COMPLEXITY MEASURES

"A man, viewed as a behaving system, is quite simple. The apparent complexity of his behaviour over time is largely a reflection of the comp lexity of the environment in which he find himself." [Simon81,p.65]

The interrelationship between physical *structures* and *behaviors* of systems is a fundamental concept in almost any area of science. In electronic network analysis, chemical engineering, and in architecture, the physical organization of a system has profound influence on its behavior. Such facts have urged researchers to creative work in developing theories and methods for diagnosing structural organization of systems and characterizing them for structural feature detection or structure-behavior correlations. The development of structural complexity was initially proposed in chemistry and biology [Bonchev87]. The study of complexity of chemical structures has great significance in recognition of structural-property correlation in chemical substances, and in design of new chemical substances. Complexity also is an important issue in other areas of science such as system theory and computer science.

Diagnosis and characterization of structural organization of buildings in terms of their connectivity or behavioral properties are important issues in design activities. Synthesis, evaluation, allocation were given as examples of activities involved in architectural design. Design as a problem solving activity may not proceed without some diagnostic means for probing the physical organization of designs and analysis of their data. For example, configuring connectivity pattern of a building requires a diagnostic means for probing the structure of forming partial (incomplete) designs as a

guide for choosing next courses of action. Similarly, an evaluation requires analysis and diagnosis of the structure and operation of a design for obtaining its actual and expected behaviors for comparison (i.e., evaluation). Another example is the allocation problem posed in chapter two. This problem, although open for an extension, seems to need a more intricate treatment compared with the first two problems. A structural analysis of the design and the analysis of the relations between the activities to be allocated is a prerequisite for proceeding the allocation activity.

To be more specific, problems introduced in this work involve the following complexity aspects:

Complexity issues in design evaluation:

- The complexity of input design:
  - . What are the flow generation potentials between the actors?
  - . Clustering of locations (activities) of design with respect to their flow generation potentials.
  - . What are topological distances in design?
  - . What are the (shortest) paths between different locations?

Complexity issues in design generation, and allocation problems:

- What are the flow generation potentials between the actors?
- Clustering of locations (activities) of design with respect to their flow generation potentials.
- The complexity of intermediate partial designs.
  - . What are the topological distances between different locations?
  - . What are the (shortest) paths between different locations?
  - . What are the most eccentric locations with respect to a location?
  - . What are the neighboring locations of a location?

Above complexity problems may be split in two significant categories. Those dealing with the *structure* of a building as a system of connected components, and those dealing with the *operation* of a building.

This chapter introduces a set of tools, and a method that deal with above complexity issues. The complexity tools are Topological Indices (TI). TIs are used in geographical science and chemical engineering to characterize topological features of systems of roads [Haggett69], and molecular structure of chemical compounds [Bonchev87]. To deal with the operational

complexity of buildings, this work takes advantage of the Q-analysis method [Atkin74]. This method enables diagnosing the relationship between a set of objects, and partially hierarchical classification of them with respect to their association degrees. Q-analysis in our case is a powerful method for detecting the flow generation potentials (dynamic aspects) associated with the activities assigned, or to be assigned, to locations a building.

TIs are complexity measures derivable from the adjacency and distance matrix of a graph. The calculation of distances in graphs involves algorithms. Reduction of computation time of algorithms is an aspect of complexity in computer science. To reduce the computational time for calculating distances in partial designs, this chapter also introduces a fast algorithm for this purpose.

4.1 Diagnosing structural complexities by means of Topological Indices (TIs)

A graph may represent the components (i.e., the nodes) of a system and relationship (i.e. the links) between them. Graphs have application in chemistry for depicting the connectivity structure of molecules. The vertices of a graph represent the individual atoms, and edges depict the valence bonds between pairs of atoms in chemistry. Such graphs are called *Chemical Graphs* or *Molecular Topology Graphs* [Trinajstic83a] [Balaban87] in chemical science. Molecular-structure complexity has been an important issue in Chemical science. In this domain, the problem was how to derive mathematically the chemical properties of compounds from their structure (molecular connectivity). One way to characterize the structure of a chemical compound is by means of complexity measures. Complexity measures characterizing chemical graphs are categorized under different names in literatures, the most common of which is Topological Indices (TI) [Balaban87] [Nicholson-87] [Tsai87]. TI's have found their ways in the areas such as network analysis, geographical science [Haggett69], and particularly chemistry [Trinajstic73a] [73b] [Kier76] for more than three decades. The structural complexity of a system may be viewed from different perspectives, urging for several complexity measures to reflect all complexity aspects of a system. Some applications of TIs in chemistry are:

- Characterization of chemical graphs, and chemical substances.
- Structural mapping of a chemical graph to an index.

- Quantitative Structure-Property Relationship (QSPR) [Tsai87] [Johnson87] [Trinajstic83b] .
- Predicting the structural complexity of non-existing molecular compounds [Rouvray83].
- Quantitative Molecular Similarity Analysis (QMSA) [Johnson87] [Rouvray83] :
  - . Molecular identification,
  - . Searching molecular databases for structurally similar compounds.
  - . Forming clusters of structurally related compounds,
  - . Ranking compounds with respect to their biological activities,
  - . Electing compounds for drug screening,
  - . Predicting molecular properties,
  - . Modeling drug receptor sites,
  - . Modeling and controlling corrosion,
  - . Predicting the degree of spread and harmfulness of pollutants in the environment,
  - . Estimating cancer-causing potentials of certain chemicals, and
  - . Developing beers with well-balanced taste,
- Quantitative Structure-Activity Relationship (QSAR) studies [Balaban87] [Hanson87].
  - . Ordering, grouping, comparing chemical structures,
  - . Exploring the structure activity relationships in chemical compounds,
  - . Quantitative structure-chemical properties correlations,
  - . Quantitative structure-biological properties correlations,
  - . Quantitative structure-boiling point correlations,
  - . Prediction of characteristics of unknown chemical compounds,
  - . Estimation of some properties of families of chemical compounds.

A TI is a numerical entity based on certain features of a graph that attempts, to some degree, to reflect a property of that graph. TIs are mainly based on local features of graphs. For example, molecular branching is a topological feature recognized for contributing in complexity of molecular graphs. At the heart of the structural analysis of a system, there is a procedure or an algorithm that converts the topological structure of that system into a single number (i.e., a TI). Such a number, then, characterizes the corresponding system in a way that correlates it with one of its experi-

mentally measured properties.

The standard procedure in devising structural-behavior (property) correlation based on topological indices is as follows:

1- Devising a TI capable of describing a structural feature of a set of objects under investigation.

2- Finding a quantitative relationship between the devised TI and a common behavioral aspect of the set of objects.

For example, the Wiener index, described in appendix-A, well correlates with a number of properties of a class of hydrocarbons, such as their boiling points, viscosity, and surface tension, and has a wide range of applications in predicting relative stabilities of unknown compounds that yet to be synthesized and in corrosion control [Rouvray83] [Adler87]. Hanson [87] uses three TIs, viz. the carbon number, the Wiener index, and the Balaban's distance-sum-connectivity index to establish a reasonable correlation with melting point of some chemical compounds. Similarly, in a building the branching degree of a location, well correlates with the interaction potential in that point. A location with branching degree one is an isolated point with potential for serving as a singleton.

In spite of wide application of TI in chemistry, there are problems associated with them. For example, a single TI is not a sufficient measure for detecting a range of complexity features of a class of systems. This means that a TI only gives a hint about the structure and the nature of a system and not the complete information as an incident or adjacency matrix. Besides, the correlation between structures of a class of objects and a TI is not always one to one. More than one structure might map into a single measure. This problem, known as the degeneracy problem [Bonchev87], is a major drawback in application of TIs in chemistry. The ideal is to devise a unique TI definition and an algorithm that could satisfactorily correlate each structure to a unique index. Fully non-degenerate TI can be devised to map chemical structures to unique indices, but till now such indices cannot correlate the structures of systems to their properties. On the other hand, TIs with little or no degeneracy, can be used for structure-property correlations of a system, but because of lack of one-to-one correspondence a structure cannot be retrieved from a data-base. The most challenging problem in chemical application of graph theory is thus, whether two approaches

above can ever converge into solutions which fulfill all these requirements [Balaban87].

To cope with these problems researches have been carried out. Some have proposed theoretical frameworks for trying to devise new indices. Kier [76] and Bonchev [87] has laid out a number of criteria for a good TI as follows:

- It must be *universal* (domain independent), and capable of characterizing any system representible as a graph.
- A good complexity measure should be specified within a *unique theoretical conception.* One should not try to devise an index based on combination of heterogeneous criteria of a system.
- It should take into account different structural complexity levels of a system and their *hierarchy*. This requires both a hierarchical classification of the component of a system, and a vector type measure instead of single number.
- It should *emphasis on the relations* between the components of a system rather than on their numbers. The number of components of a system does not necessarily reflect its organizational complexity. This requirement demands different weighting of size and relation parameters.
- A good TI should increase monotonically with increase in number of a complexity features.
- It should correspond to the *intuitive* idea of complexity.
- It should *differentiate* (not degenerate) the non-isomeric graphs.
- It must be as *simple* as (not too sophisticated) possible.
- It should be *practical* in analyzing the characteristics of a set of systems.

To minimize the degeneracy, some researchers have introduced combined indices. Randic's molecular identification numbers [Rouvray83], and Wiener's index [Rouvray83] are examples of these indices. A vector type index is another approach to reduce degeneracy, and increase distinguishing power of TIs. The hierarchical model of Bonchev [87] is an example of this approach. This model, which is almost universal, is not restricted to chemical graphs and could be applied to any systems representible in a graph. I believe, Bonchev model, introduced in the following section, is applicable to design of data-bases of precedents of architectural design, for indexing and

retrieval of floor-plans having certain properties.

4.1.1 A hierarchical Model for measuring the topological complexity

To decrease the degeneracy problem associated with the TIs, a hierarchical model for arriving at vector type indices have been proposed [Bonchev87a]. Figure 4.1 depicts a five levels model for arriving at a fairly non-degenerate indices. The first level is the graph size. The next level deals with the connectivity of the systems under investigation. This level is the next fundamental level after the size level. The third level considers the relationship between different classes of elements, if the system under consideration has such different elements. The fourth and fifth levels deal with specific metric and specific symmetry. The first three levels have general application, while the last two levels are applicable to chemical structures, where the notions of bound lengths and bound angles are important.

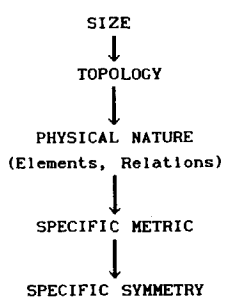


Figure 4.1: A hierarchical model for measuring the structural complexity of systems [Bonchev87a]

Topological level is the most important one in measuring the complexity of a system; so, it is natural to treat this level in greater depth. Bonchev [87a] proposes a separate hierarchical scheme for topological complexity. This is depicted in figure 4.2. Connectedness is primary a topological feature. The next level constitutes the graph types related to adjacency. This measure can differentiate classes of directed, undirected, and multi-graphs. Different fragments of a graph such as its cycles, branches,

bridges, and linear patterns are represented in the next level. Fragments of a graph also may be sorted out in terms of their complexities. *Cyclicity* is the most complex feature, with *bridging, branching,* and *linearity* as next complex features respectively. Symmetry is the fourth level in the hierarchy. This level is related to the graph with non-symmetrical and non-binary relations between their vertices. The fifth level is included for distinguishing non-isomeric graphs, if the first four levels are not sufficient for differentiation. This level is divided into two branches; a metric (distance) sub-level followed by a level corresponding to paths in a graph, and a sub-graphs level. The path sub-level can distinguish paths of different size in a graph, while the sub-graph level considers classification of sub-graphs of different size.

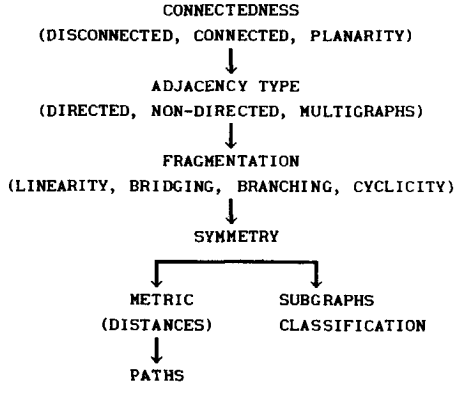


Figure 4.2: A hierarchical model for measuring the structural complexity of graphs [Bonchev87a]

In order to have the mathematical model reflect the hierarchy of the conceptual model, one must avoid to arrive at a single number for showing the structural complexity of a system. To achieve this, Bonchev proposes complexity vectors, with as many components as the hierarchies of the conceptual model dictate, for showing the complexity of a system. For example, for a graph G the complexity measure will look like this:

$$C(G) = (C_{size}, C_{topology}, C_{physical\text{-}nature}, C_{metric}, C_{symetry})$$

$$C_{TOP} = (C_{conn.}, C_{adj.}, C_{frag.}, C_{sym.})$$

$$C_{SYM} = (C_{metric}, C_{sub\ graphs})$$

The most important level with respect to the structural complexity of buildings in terms of their connectivity property, and topological properties of other systems in general, is the third level. The first two levels can contribute a little towards feature detection of a system. To enhance the structural feature detection, fragmentation contributes a great deal. Graph fragmentation means recognition of sub-graphs of different features and properties within a graph. Fragmentation processes additionally contribute to the recognition of graph complexities. For example *cycles* and *branches* have been recognized for a long time as features which increases the complexity of a graph. Bonchev [78a] has introduced two more notions, that are *bridges* and *lines*, for graphs. This breaks down a graph into four distinct fragments which together they make up the third hierarchical level.

In order to fulfill requirement two of the complexity measures, described earlier, it is essential that the fragmentation level and its components to be specified within the same adjacency conception of the second level. Specifically we want the summands of the entries for the fragments to be equal to the total adjacency A(G). The total adjacency of a graph then can be defined as the sum of contribution of fragment adjacencies:

$A(G) = A_{CY}(G) + A_{BN}(G) + A_{BR}(G) + A_{LI}(G) = 2m,$

where, m is the number of edges.

Or: $A(G) = a_{ij}^{CY} + a_{ij}^{BN} + a_{ij}^{BR} + a_{ij}^{LI} = m_{CY} + m_{BN} + m_{BR} + m_{LI}$

This formula for undirected graphs is: $A(G) = 2(m_{CY} + m_{BN} + m_{BR} + m_{LI})$

Furthermore:

- m_{CY} is equal to the cyclomatic number (i.e., $m_{cy} = m-n+1$).
- m_{BR} is the number of bridge edges, which are edges that connect two cycles, but they do not belong to any cycle.
- m_{LI} is the number of linear edges, which can be taken as the number of degree two vertices in the graph.
- m_{BN} is the number of branched edges, which is taken as the number of remaining edges after eliminating the cyclic, bridging, and linear ones.

To increase the discriminating power of the complexity measure, linear edges may be grouped according to the type of sub-graphs to which they belong [Bonchev78a].

Here: $m_{LI} = m^{CY}_{LI} + m^{BR}_{LI} + m^{SC}_{LI} + m^{AB}_{LI}$, where:

m^{CY}_{LI} = The number of linear edges in cycles.

m^{BR}_{LI} = The number of linear edges in bridges.

m^{SC}_{LI} = The number of linear edges in side chains.

m^{AB}_{LI} = The number of linear edges in acyclic branches.

Similarly branching can be broken down into acyclic and cyclic branching groups. Edges associated with cycles are *cyclic-branching*, while the *acyclic-branching* embrace the effect of branching in acyclic fragments, bridges, and side-chains of cyclic fragments.

$m_{BN} = m^{CY}_{BN} + m^{AC}_{BN}$, where:

m^{CY}_{BN} = The number of edges in cycles - $(m^{CY}_{LI} + m_{CY})$

$m^{AC}_{BN} = m^{AB}_{BN} + m^{BR}_{BN} + m^{SC}_{BN}$

The relationship between different fragments of a graph discussed above are depicted in the following figure.

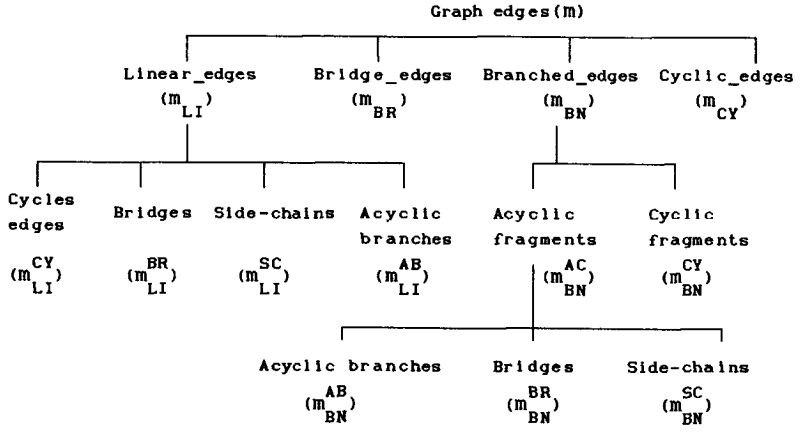
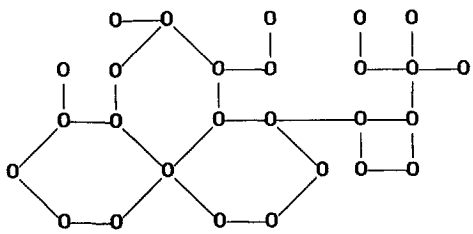


Figure 4.3: Classification of structural complexity in graphs



$m_{CY} = m-n+1=30-27+1=4$

$m_{LI} = 11 \quad (m_{LI}^{CY}=8,\ m_{LI}^{BN}=1,\ m_{LI}^{SC}=1,\ m_{LI}^{AB}=1)$

$m_{BR} = 1$

$m_{BN} = m-(m_{CY}+m_{LI}+m_{BR})=14$

Figure 4.4: A graph and its measures of structural complexity

The ten components of the total adjacency described above characterizes in detail the connectivity complexity of any graph. We can arrange these terms in hierarchical order with respect to their degree of importance in terms of the complexity. Intuitively, it is reasonable to assume that cycles are the most complex and lines are the simplest and least complex connectivity feature. Bridges which link cycles also may be taken as more complex than acyclic branches, but less complex with respect to cyclic branches. Similarly, we can associate a higher level to bridges than side chains and to side chains than acyclic ones in both branches and linear edges. Now the total hierarchy can be written as [Bonchev78a]:

$$C_{CY} > C_{BN}^{CY} > C_{BR} > C_{BN}^{BR} > C_{BN}^{SC} > C_{BN}^{AB} > C_{LI}^{CY} > C_{LI}^{BR} > C_{LI}^{SC} > C_{LI}^{AB}$$

Obviously, these hierarchical relationships will provide a higher discriminating power of the total adjacency of a graph. Now, this complexity measure can be considered as a ten component vector, with the following discriminating complexity rules attached to it, for comparing the complexities of two graphs G1 and G2:

$A(G)=(C_{CY}, C_{BN}^{CY}, C_{BR}, C_{BN}^{BR}, C_{BN}^{SC}, C_{BN}^{AB}, C_{LI}^{CY}, C_{LI}^{BR}, C_{LI}^{SC}, C_{LI}^{AB})$

IF A(G1) > A(G2), THEN C(G1) > C(G2) ELSE

IF A(G1) < A(G2) THEN C(G1) < C(G2) ELSE

IF C_{CY}(G1) > C_{CY}(G2) THEN C(G1) > C(G2) ELSE

IF C_{CY}(G1) < C_{CY}(G2) THEN C(G1) < C(G2) ELSE

IF ...

4.1.2 Application of TIs on the architectural domain

TIs, have limited power in structure-behavior or structure-property correlation, but can play a significant role in design activities such as design synthesis and activity allocation. Most of the TIs presented in appendix-A are capable of detecting structural features of graphs, and reflecting their complexity to a certain degree. Some of these TIs, thus, can serve during automated design processes for structural detection. The rest are purposely introduced for extending the minds for continuation of this work. Almost all TIs related to branchiness of graphs are able to support design decisions while synthesizing a design. For example, branching degree can be used as a constraint limiting the branching degree of locations. TIs also may be used as a numerical indicator for detecting the potential privacy of locations in a design. For example, locations with minimum degree branchiness (i.e., 1) are potentially private locations. Other TIs such as topological distances and eccentricity of locations are used by TOPGENE as a measure for heuristic optimization of designs with respect to social points of view such as privacy, circulation cost, and community. Here is a TOPGENE rule integrating the eccentricity and branchiness into the design process:

```
- LET HCLP be a hierarchical clusters of location pairs.
- LET (pair ∈ HCLP) be a location pair with the highest flow potential.
- Let PD be the current partial design.
  - COND: - (EQL current_norm_on_agenda 'COMMUNITY).
          - ((FIRST pair) ∈ processed_locs).
          - ((SECOND pair) ∉ processed_locs)).
    - LET ECC-LOCS be list of locations in PD ranked according to their eccen-
          tricity with respect to (FIRST pair).
    - DOLIST (LOC ECC-LOCS):
          - LET branch1 := max_branching_degree of LOC - current_branching_degree
                of LOC.
            - COND - NOT((LIST LOC (SECOND pair)) ∈ *NEG_LINKS*) & (branch1 > 0).
                   - (PUSH (SORT_PAIR (LIST LOC (SECOND pair))) *SOLUTION*).
                   - (UPDATE *DISTANCE_MATRIX* holding distances of the PD.
          - UPDATE_BRANCHINESS of LOC and (SECOND pair).
          - ADD (SECOND pair) to processed_locations.
          - Break the loop.
    - END-DOLIST.
```

TOPGENE uses a data driven approach to generate design based on the analysis of input data and heuristic rules. A model-driven approach would try to produce designs based on a knowledge base of precedents of designs. Such a system, would search the knowledge base for particular designs, select precedents, refine the selected designs, filter them, and perhaps would have

to modify a finally selected design before presenting it as a solution. Such a process has several issues that may be dealt with by TIs. For example, storing precedents of designs, demands a classification and indexing strategy. Classification of designs with respect to their connectivity properties, one way or the other, has to deal with the connectivity features of designs. Here, the use of TIs are inevitable. Indexing precedents of design solutions with respect to their most elementary structural features, such as size, average branching degree, compactness, etc. are something that may not be neglected, if efficient retrieval is required. Given a design problem, a basic TI characterizing its solution is its size. Clearly a design solution with hundreds of locations may not be expected to be appropriate for a design problem of a small house with a few locations. Furthermore, after selecting a number of design solutions, a system has to go through a deeper selection process for minimizing the number of choices made. Some of the TIs, discussed above, well correlate with specific properties of topological patterns of building, without a need for deep structural analysis of the pattern. A simple example is the compactness measure of a pattern, and the cyclomatic number. These measures are clear indication of complexity of a pattern, and can serve in quick detection and selection of patterns with specific properties. For example search for tree designs (e.g., single-loaded, double-loaded) must lead to quick rejection of designs with a positive cyclomatic number.

A better solution to search for specific designs is the use of a vector type TI such as the hierarchical model of Bonchev. This model can serve not only in selection but also produces detail structural characterization of a design. Various sub-features of a pattern reflected in this hierarchical model are valuable tools for detailed diagnosis of solutions patterns even for allocation problem. Some of the possibilities were mentioned before. A detailed discussion on the applicability of TIs is left further research work.

4.2 Diagnosing topological distances in a partial design

The problem of finding the shortest paths and their lengths (distances) in a graph is important in the applications of graph theory in scientific areas, such as: network analysis, transportation [Dial79], also generating design

solutions with respect to a set of social norms. TOPGENE, as mentioned, uses eccentricity index and topological distances of locations as diagnostic means for probing partial designs. Keeping track of distances in partial design is a time consuming process. In this section an efficient algorithm for this purpose is presented. This algorithm has a time complexity bounded by $(n-1)^2/4$ for readjusting distances between the nodes of a graph G that is incremented by a new link, where n is the order (i.e., number of vertices) of G. Existing algorithms, such as the power-matrix method [Flament63] [Haggett69], Dijkstra's algorithm, and Floyd's algorithms [Christofides75] [Gibbons85] [Buckley90] have a computational complexity in the order of n^3 for achieving the same goal. The power matrix method is based on the following theorems:

Theorem-1 [Festinger1949] [Flament63]: In a graph G, the number of paths of length r from a vertex v_i to another vertex v_j is given by the a^r_{ij} element of A^r, where A^r is the r-th power of the adjacency matrix corresponding to G. (See proof in [Flament63] and [Buckley90].

Theorem-2 [Flament63]: In a graph G with n vertices, the length of a shortest path is at most equal to (n-1).

Theorem-1 states that an adjacency matrix A representing a graph G can be powered to yield a series of matrices of order $A^2, A^3, \ldots, A^{n-1}, A^n$ with some interesting properties. In general an element a_{ij} of A^n represents the total number of n-steps paths between the vertex pair (v_i, v_j). For example, non-zero elements of A represent direct (one step) connections between the corresponding vertex pairs, a non-zero element of A^2 (A to the power of 2) represents the number of 2-step paths between corresponding vertices, and so on. This implies that we can use powered matrices of a graph for calculating the lengths of the shortest path between different node-pairs.

Based on above theorems and discussion followed we have the following algorithm for the calculation of the path-lengths in a graph [Flament63]:

1- Compute the successive powers of G, at most up to the power of r=n-1, when n is the order of G.

2- The length of the shortest path between two vertices v_i and v_j is

equal to the non-zero element of the smallest power matrix corresponding to the row i and column j.

For convenience, we can assume that:

- For any $r \leq n-1$ that $a^r_{ij} = 0$, $d(ij) = \infty$, and
- For all v_i, $d(v_i\ v_i)=0$.

If we keep the lengths of the shortest paths in a matrix D, then in order to denote the correspondence between this matrix and the power matrices, let's enhance D with a subscript p to correspond it to a powered matrix A at a given situation. For example, D_n will denote the correspondence of D with A^n. Here, D_1 will contain all connections that appeared in A^1; D_2 will include all the 2-step connections paths that did not exist in D_1, and so on. The process of finding the number of n-step connections (i.e., calculating A^n) and the length of the shortest path between pairs of locations (i.e., repairing D_n at each level n) can be continued until all non-diagonal elements of the shortest path matrix D is filled. At this level D_n contains distances the shortest path between all location-pairs. An example of this algorithm at work can be found in [Haggett69] page 35.

The information in these matrices have implications in architectural design. For example, the sum of the values in a row corresponding to a vertex v_i in A^n is the total number of connections from v_i to any location including v_i at maximum n-step in the graph. Similarly the sum of the values in a column corresponding to a vertex v_j is an indication of total number of routes coming to v_j at maximum n-step. Similarly for a graph representing a building, the power matrices A^2, A^3, ..., A^n suggest the access assurance of the building at different levels.

The shortest path matrix also may be used to compare relative accessibility of nodes in a graph. The rows in such a matrix show the shortest paths from location indexed on the left of the matrix to all other locations indexed at the top of the matrix. The largest value in a row is the Kônig *number* for location corresponding to that row. Kônig number is discussed in appendix-A.

4.2.1 A new efficient algorithm

To reduce the computational complexity of distance calculation by TOPGENE

during generating designs, I introduce a new algorithm for this purpose. The problem is to efficiently keep track of distances in an incrementally growing graph G (partial designs generated by TOPGENE) that is presumably in the order of 1 (i.e., has one node) at step t, and is in the order of n at step (t+n). TOPGENE needs to keep track of distances between the location pairs of the partial designs, each time they are incremented by the addition of a new location.

To keeping track of distances in an incrementally changing graph that presumably has 1 node at step t, and has n nodes at step t+n, based on applying existing algorithms, has a computational time proportional to $\sum^{n} i^3$ (i.e., the computational complexity is in the order of $O(n^3)$), while the algorithm introduced here has a time complexity bounded by $\sum^{n} (i-1)^2/4$ for a growing process of G from 1 node to n nodes. This algorithm is based on distance readjustment as opposed to the existing approaches, which are based on the distance recalculation. This algorithm keeps track of distances between all vertex pairs in G in a distance matrix, recalculates only those distances that have been influenced because of the growth of G, and readjusts the distance matrix. This is computationally cheaper than, for example, the power matrix method which recalculates distances in G, based on the access matrix, any time G is incremented by a new link. The time complexity of this algorithm for each distance readjustment in G is bounded by $(n-1)^2/4$. Now, if we assume that G is in the order of 1 at the initial step and is in the order of n at the final step, then the computation time adds up to $\sum^{n} (i-1)^2/4$ for a growing process of G from 1 node to n nodes.

4.2.1.1 Theoretical foundations

The key issue in reducing the time complexity of recomputing distances in an incrementally changing graph G is the method of detecting new tracks in G after adding a new link to it, and calculating only the lengths of these tracks. Such a method should be less complex and more efficient than the recalculation of all distances once the graph is incremented by a new link.

To provide a basis for such a method and its implementation, I first present a lemma (lemma-1) for detecting the newly created tracks and their corresponding distances in a graph that has been incremented by the addition

of a new link. This lemma is based on two previous theorems (theorems 1 & 2) that represent the properties of a track between two points in a graph.

Theorem-1 (Bratton's theorem)

If $\theta(v_k\ v_l)$ is a track between the two vertices v_k and v_l, then the path between any vertex pairs on $\theta(v_k\ v_l)$ also is a track. the proof is to be found in Flament [1963, page 30].

Theorem-2

The vertex v_x is located on a track $\theta(v_k\ v_l)$, if and only if $d(v_k\ v_x) + d(v_x\ v_l) = d(v_k\ v_l)$. The proof is to be found in Flament [1963, Page 30].

The implication of this theorem is that in an undirected graph G, the distance between two end points of the track $\theta(v_k\ v_l)$ is equal to the sum of distances between the end points of two tracks separated by v_x, a point on $\theta(v_k\ v_l)$ (i.e., $d(v_k\ v_l) = d(v_k\ v_x) + d(v_x\ v_l)$).

Lemma-1

Let G'(V', E') be a connected undirected graph with known distances between any vertex pair. If a new link $\{(v_i\ v_j) \ni i\#j\}$ with a weight w, and at least, a node already in G', is added to G', then G(V, E) is an incremented graph with new paths. A new path in G complying with one of the following conditions is a track:

$$p(v_i, v_j, \ldots, v_k) = \theta(v_i, v_j, \ldots, v_k)$$
$$\text{iff } d(v_j\ v_k) + w < d(v'_i\ v'_k),\ \forall k\#i,\ k\#j \qquad (1)$$

$$p(v_j, v_i, \ldots, v_l) = \theta(v_j, v_i, \ldots, v_l)$$
$$\text{iff } d(v_i\ v_l) + w < d(v'_j\ v'_l),\ \forall l\#i,\ l\#j \qquad (2)$$

$$p(v_k, \ldots, v_j, v_i, \ldots, v_l) = \theta(v_k, \ldots, v_j, v_i, \ldots, v_l)$$
$$\text{iff } d(v_k\ v_j) + w + (v_i\ v_l) < d(v'_k\ v'_l),\ \forall k\ \&\ l \text{ in (1) \& (2)} \qquad (3)$$

Furthermore, the distances of these new tracks are:

$$d(v_i\ v_k) := w + d(v_j\ v_k),\ \forall k\#i,\ k\#j \qquad (4)$$
$$d(v_j\ v_l) := w + d(v_i\ v_l),\ \forall l\#i,\ l\#j \qquad (5)$$
$$d(v_k\ v_l) := d(v_k\ v_j) + w + (v_i\ v_l),\ \forall k\ \&\ l \text{ in (4) \& (5)} \qquad (6)$$

Proof:

Necessary condition: Let $v_k \in G$ and $v_l \in G$ be two arbitrary vertices in an incremented graph G. $p(v_k\ v_l) \in G$ is a new path, if and only if it includes at least a new link different from the links in $p(v'_k\ v'_l) \in G'$. We know that G is identical with G' except for the new link $(v_i\ v_j)$, so, the necessary condition for $p(v_k\ v_l)$ being a new path is: $(v_i\ v_j) \in p(v_k\ v_l)$.

Sufficient condition: From the condition above it follows that a new path in G falls into one of the following three classes of paths:

$p(v_i\ v_k) = \{(v_i\ v_j)(v_j\ \ldots)\ \ldots\ (\ldots\ v_k)\},\ \forall k$ (7)

$p(v_j\ v_l) = \{(v_j\ v_i)(v_i\ \ldots)\ \ldots\ (\ldots\ v_l)\},\ \forall l$ (8)

$p(v_k\ v_l) = \{(v_k\ \ldots)\ \ldots\ (\ldots\ v_j)(v_j\ v_i)(v_i \ldots)\ \ldots\ (\ldots\ v_l)\},\ \forall k\ \&\ l$ (9)

Now, for distinguishing the tracks among the classes of paths mentioned above, we first take (7) and (8). The sufficient condition for any path in (7) and (8) to be a track is that the distance between its end vertices in G be shorter than their previous distances in G'. For example, $d(v_i\ v_k)$ must be less than $d(v'_i\ v'_k)$. The distance between v_i and v_k is not known yet, but we know that $d(v_i\ v_k) = d(v_i\ v_j) + d(v_j\ v_k)$. Hence, the sufficient condition for detecting the tracks from among the paths (7) and (8) are:

$d(v_i\ v_j) + d(v_j\ v_k) < d(v'_i\ v'_k),\ \forall k$, and

$d(v_j\ v_i) + d(v_i\ v_l) < d(v'_j\ v'_l),\ \forall l$

We know that: $d(v_j\ v_k) = d(v'_j\ v'_k)$, $d(v_i\ v_l) = d(v'_i\ v'_l)$, and $d(v_i\ v_j) = W$. So, first, above inequalities can be verified, and second, the new distance for any point pair holding above inequalities can be calculated as follows:

$d(v_i\ v_k) := W + d(v'_j\ v'_k),\ \forall k \neq i\&j$

$d(v_j\ v_l) := W + d(v'_i\ v'_l),\ \forall l \neq i\&j$

Now, the tracks among the paths described by (9) must be distinguished. We know from Bratton's theorem that a path p(Vk Vl) in (9) is a track, if its sub-tracks:

$\{(v_k\ \ldots)\ \ldots\ (\ldots\ v_i)\}$ (10),

and

$\{(v_j\ \ldots)\ \ldots\ (\ldots\ v_l)\}$ (11)

are tracks. This implies that the necessary condition for a path in (9) to be a track is that: its end points must be identical with end points of the

tracks recognized in (7) and (8). So, (10) and (11) are tracks if it was proved so in (7) and (8).

Now, the sufficient condition for the paths between these point pairs to be tracks is the inequality argument similar to the one for (7) and (8), i.e.: $p(v_k\ v_l) = \theta(v_k\ v_l)$, if:

$d(v_k\ v_j) + d(v_i\ v_j) + (v_i\ v_l) < d(v'_k\ v'_l)$, $\forall k \& l$ in (7) & (8)

And, the lengths of these new tracks are:

$d(v_k\ v_l) := d(v_k\ v_j) + w + (v_i\ v_l)$, $\forall k \& l$ in (7) & (8)

4.2.1.2 Algorithm

Based on the lemma-1, I present the following algorithm, with a computational complexity bounded by $(n-1)^2/4$, for keeping track of the distances in an incrementally changing graph G from a node to n nodes. G is assumed to be weighted and undirected. The algorithm with slight modification also may be applied to directed graphs.

When G is undirected, the distance matrix is symmetrical, and the storage requirement for storing distance information is $n^2/2$ for a graph of order n. The storage requirement for both distance and access information of an undirected-weighted graph is n^2. When G is both undirected and unweighted, then the storage requirement reduces to $n^2/2$, since distances equal to 1 imply direct links between the nodes as well.

The storage requirements for storing access and distance information of different graph types are reflected in the following table:

Graph type	Storage requirement
Undirected-weighted	n^2
Undirected-unweighted	$n^2/2$
Directed-weighted	$2n^2$
Directed-unweighted	n^2

The storage requirement for keeping the distance and access information in our case is then n^2. This is the size of a n by n matrix, which holds the distances in its upper triangle and access information in its lower triangle. The number n is the final order of G, so, the matrix is assumed to be fixed for a complete growing process of G. In order to reduce the storage

requirement, at least in the intermediate steps, it is possible to presume a growing matrix as well. This is achievable only at the expense of recopying such a matrix into a new one, each time G is incremented.

The time complexity bound $(n-1)^2/4$ for the algorithm is related to the distance readjustment corresponding to the condition (9). The worst case arises when G is a tree, and is incremented by a link that connects the roots of two sub-graphs (sub-trees) of the same weight in G. In this situation, and if the new link connects two nodes which are the only neighboring nodes of the *central-node*[1] in G, then the number of new tracks in (7) and (8) are exactly (n-1)/2. The comparison of these two sets of tracks for condition (9) has a time complexity in the order of $(n-1)^2/4$.

The algorithm is implemented by three procedures called ADD-LINK, ADJUST-DISTANCES-1 and ADJUST-DISTANCES-2. These procedures are presumed to have access to the distance and access information about the growing graph G(V, E) through an access-distance matrix. ADD-LINK is a control procedure that, upon receiving a new link $e \notin E$ with a weight w, adds the new link to G and controls if both nodes of the new links are in G, one of its nodes is in G, or none of its nodes are in G. ADD-LINK then calls either ADJUST-DISTANCES-1 or ADJUST-DISTANCES-2 procedure for distance readjustments of the new incremented graph.

The ADJUST-DISTANCES-1 recalculates (adjusts) the distances of the new tracks originating from a newly added node with respect to other nodes in G, and updates the distance matrix. The inputs index1 and index2 to this procedure are the indices of nodes of a recently added link to the access-distance matrix, with index1 corresponding to a new node, and index2 corresponding to an existing node in G.

The ADJUST-DISTANCES-2 readjusts distances of the new tracks created as a result of linking two existing nodes in G with each other. The inputs to this procedure are indices of the nodes of the new link to the access-distance matrix, and the weight associated with the new link.

These procedures are presented in a pseudo language similar to Pascal with some simple list processing notations. Comments start with a semicolon ";", and a node with an index i to the distance matrix is denoted by

[1] See appendix-A for definition.

node(i). The algorithm with slight modification can be applied to directed graphs as well.

The definitions of parameters and some of the variables used by this procedure are given below. Other variables are printed in italic font.

- *link* : the new link to be added to G.
- w : the weight associated with the new link.
- *nodes* : list of the current nodes in G.
- **ad_matrix** : the global variable, denoting the access-distance matrix

Procedure: ADD-LINK (link w nodes)

```
1- index1 := index of the node1 ∈ link to *ad_matrix*.
2- index2 := index of the node2 ∈ link to *ad_matrix*.
3- IF:   - (node1 ∉ nodes) & - (node2 ∉ nodes).
   THEN: - nodes := nodes + {node1 node2}.
         IF:   - node1 < node2.
         THEN: - *ad_matrix* (index1 index2) := w,
               - *ad_matrix* (index1 index2) := 1.
         ELSE: - *ad_matrix* (index1 index2) := 1,
               - *ad_matrix* (index1 index2) := w.
   ELSE IF:    - (node1 ∈ nodes) & (node2 ∈ nodes).
        THEN:  - Call procedure ADJUST_DISTANCES_2 (index1 index2 w).
   ELSE IF:    - Only one node, say node2 ∈ nodes.
        THEN:  - nodes := nodes + {node1}.
               IF:   - node1 < node2.
               THEN  - *ad_matrix* (index1 index2) := w,
                     - *ad_matrix* (index1 index2) := 1.
               ELSE  - *ad_matrix* (index1 index2) := 1,
                     - *ad_matrix* (index1 index2) := w.
4 - Call procedure ADJUST_DISTANCES_1 (index1 index2 w).
5 - END-ADD-LINK.
```

Procedure: ADJUST-DISTANCES-1 (index1 index2 w)

```
;; Get distances of all nodes relative to node(index2).
1- FOR row := 1 to index2
      PUSH *ad_matrix*(row index2) to distance_list
   END-FOR.
2- FOR col := (index2 + 1) to the last column of the *ad_matrix*:
      PUSH *ad_matrix*(index2 col) to distance_list
   END-FOR.
3- distance_list := (REVERSE distance_list)
   ;; Fix distance of the newly added node(index1) relative to other nodes.
   ;; Fix distances in the "index1"_th column of *ad_matrix*.
4- FOR row := 1 to index1:
      - distance := (POP distance_list)
      - IF:   - distance # 0⌟
        THEN: - *ad_matrix* (row index1) := distance + w.
   END-FOR.
```

```
   ;; Fix distances in the "index1"_th row of *ad_matrix*.
5- FOR col := index1 to the last column of the *ad_matrix*:
      - distance := (POP distance_list).
      - IF:    - distance # 0,
        THEN:  - *ad_matrix* (index1 col) := distance + w.
   END-FOR.
6- END-ADJUST-DISTANCE-1.


Procedure: ADJUST-DISTANCES-2 (index1 index2 w)
1- IF:   - index1 > index2.
   THEN: - swap index1 with index2
  ;; Store distance and access of the new link.
2- *ad_matrix* (index1 index2) := w.
3- *ad_matrix* (index2 index1) := 1
  ;; Get distances of all nodes relative node(index1).
  ;; Get distances in the "index1"_th column of *ad_matrix*.
4- FOR row := 1 to index1:
       PUSH *ad_matrix*(row index1) to list1.
   END-FOR.
  ;; Get distances in the "index1"_th row of *ad_matrix*.
5- FOR: col := (i + 1) to n (dimension of the *ad_matrix*).
       PUSH *ad_matrix*(i col) to list1.
   END-FOR.
  ;; Get distances of all nodes relative to node(index2).
  ;; Get distances in the "index2"_th column of *ad_matrix*.
6- FOR row := 1 to index2:
       PUSH *ad_matrix*(row index2) to list2.
   END-FOR.
  ;; Get distances in the "index2"_th row of *ad_matrix*.
7- FOR col := (index2 + 1) to n (dimension of the *ad_matrix*):
       PUSH *ad_matrix*(index2 col) to list2.
   END-FOR.
8- list1 := (REVERSE list1).
9- list2 := (REVERSE list2).
  ;; Adjust distances of the new tracks.
10- FOR row := 1 to index1:
       distance1 := (POP list1).
       distance2 := (POP list2).
       difference:= distance1 - distance2
       ;; Adjust distances, and save adjusted nodes.
       IF:   - difference > w.
       THEN  - *ad_matrix* (row index1) := distance2 + w.
             - PUSH row to set2.
       ELSE IF:   - difference < - (w).
            THEN  - *ad_matrix* (row index2) := distance1 + w.
                  - PUSH row to set1.
    END-FOR.
11- col := (1 + index1).
12- row := (1 + index1).
13- FOR n := 1 to ((index2 - index1) - 1):
       distance1  := (POP list1).
       distance2  := (POP list2).
       difference := distance1 - distance2.
```

```
        ;; Adjust distances, and remember the adjusted nodes.
        IF:  - difference > w.
        THEN:- *ad_matrix* (index1 col) := distance2 + w.
             - PUSH col to set2.
        ELSE IF:  - difference < -(w).
             THEN:- *ad_matrix* (row index2) := distance1 + w.
                  - PUSH row to set1.
        col := 1 + col.
        row := 1 + row.
    END-FOR.
14- FOR col := index2 to n (the size of *ad_matrix*).
        distance1  := (POP list1).
        distance2  := (POP list2).
        difference := distance1 - distance2.
        ;; Adjust distances, and remember the adjusted nodes.
        IF:   difference > w.
        THEN: *ad_matrix* (index1 col) := distance2 + w.
              PUSH col to set2.
        ELSE IF:  - difference < - (w).
             THEN:- *ad_matrix* (index2 col) := distance1 + w.
                  - PUSH col to set1.
     END-FOR.
;; Adjust distances of the new tracks θ(v_i v_j), where v_i ∈ set1 & v_j ∈ set2.
15- FOR indexi := (POP set1):
      FOR indexj := (POP set2):
        IF:   - index1 < indexi.
        THEN: - distancei := *ad_matrix* (index1 indexi).
        ELSE: - distancei := *ad_matrix* (indexi index1).
        IF:   - index2 < indexj.
        THEN: - distancej := *ad_matrix* (index2 indexj).
        ELSE: - distancej := *ad_matrix* (indexj index2).
        IF:   - indexi < indexj.
        THEN: - distance := *ad_matrix* (indexi indexj).
              IF:   - distance > (distancei + distancej + 1).
              THEN: - *ad_matrix* (indexi indexj) := distancei + distancej +
                      w
              ELSE: - distance := *ad_matrix* (indexj indexi).
              IF:   - distance > (distancei + distancej + w).
              THEN: - *ad_matrix* (indexj indexi) := distancei + distancej +
                      w
      END-FOR.
    END-FOR.
16- END-ADJUST-DISTANCE-2.
```

4.2.1.3 Example

To exemplify the algorithm above, we consider an incrementally changing graph G that is assumed to be undirected and unweighted, and has a maximum number of nodes equal to 18. The matrix for storing the access and distance information is called **ad_matrix**, with access and distances kept in the

upper triangle. A distance equal to "1" in the matrix represents a direct link (access), and a "∞" indicates a disconnection between the corresponding nodes. The distance from a node to itself is denoted by a 0.

To demonstrate the process, we take two cross sections of the process of distance adjustment for the graph G. One in case when the added link has one node already in G, and another in both nodes of the link are in G. In the first case the new link does not form new circuits, and there is only one node (the newly added node) for which the distances to other existing nodes have to be recalculated. In the second case, the addition of a link creates new circuits in G, which in turn causes the creation of new paths and new tracks in G.

To clarify matters, let's assume that G, at step t, has the configuration depicted in figure 4.5. This means that G has been incremented one link at a time, and based on the presented algorithm, the distances and access information has been kept up to date. Furthermore, we assume that the final order of G is known beforehand, so the **ad_matrix** is prefixed for G at final stage. The status of the **ad_matrix** at t is shown in figure 4.6.

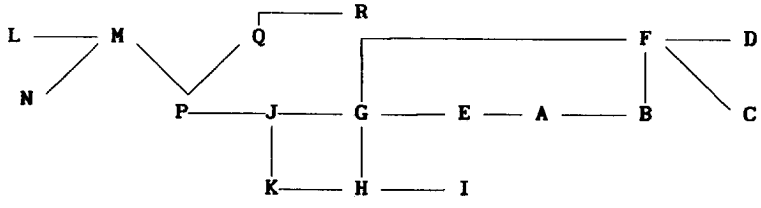


Figure 4.5: A graph G at step t

Now, if we add to G a new link (O P), at step t+1, then the new G will look like:

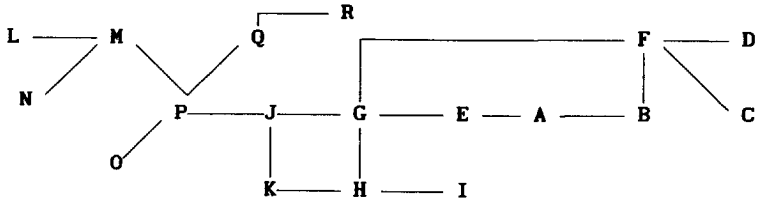


Figure 4.7: Graph G at step t+1

R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A	
0	1	2	*	4	3	4	4	3	6	5	4	5	5	6	6	6	6	R
	0	1	*	3	2	3	3	2	5	4	3	4	4	5	5	5	5	Q
		0	*	2	1	2	2	1	4	3	2	3	3	4	4	4	4	P
			0	*	*	*	*	*	*	*	*	*	*	*	*	*	*	O
				0	1	2	4	3	6	5	4	5	5	6	6	6	6	N
					0	1	3	2	5	4	3	4	4	5	5	5	5	M
						0	4	3	6	5	4	5	5	6	6	6	6	L
							0	1	2	1	2	3	3	4	4	4	4	K
								0	3	2	1	2	2	3	3	3	3	J
									0	1	2	3	3	4	4	4	4	I
										0	1	2	2	3	3	3	3	H
											0	1	1	2	2	2	2	G
												0	2	1	1	1	2	F
													0	3	3	2	1	E
														0	2	2	3	D
															0	2	3	C
																0	1	B
																	0	A

* Stands for infinity.

Figure 4.6: The access-distance matrix for the graph of figure 4.5

The addition of a new link (O P) to G, does not change the distances between the existing node pairs; but new tracks emerge from O to all existing nodes. The distances between this new node and all other nodes, thus, have to be added to the distance matrix. By looking at the upper portion of the access-distance matrix (figure 4.6), we see that the distances between O, the new node, and all other nodes are marked *, for infinite, standing for disconnection, and the distance of O to itself is marked 0. The calculation of distances of the tracks from O to other nodes and insertion of them into distance matrix correctly updates its status. This is trivial, since distances of O relative to any node in G are equal to the distance of P to the corresponding node plus the weight of the new link (P O). i.e.:

$d(O\ v_i) = d(P\ v_i) + w \qquad \{\forall i \ni v_i \neq O \ \&\ vi \neq P\}$

The distances of the new node O to all other nodes before and after the adjustment are:

```
                                  R Q P O N M L K J I H G F E D C B A
Distance-list of the node  P:  (2 1 0 ∞ 2 1 2 2 1 4 3 2 3 3 4 4 4 4)
Distance-list of the node  O:  (∞ ...                                ∞)

                                  R Q P O N M L K J I H G F E D C B A
Distance-list of the node  P:  (2 1 0 1 2 1 2 2 1 4 3 2 3 3 4 4 4 4)
Distance-list of the node  O:  (3 2 1 0 3 2 3 3 2 5 4 3 4 4 5 5 5 5)
```

Figure 4.8: The cross sections of AD-MATRIX corresponding to the nodes P and O before and after the addition of (O P) to G

The **ad_matrix**, after the adjustment will look like:

R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A	
0	1	**2**	3	4	3	4	4	3	6	5	4	5	5	6	6	6	6	R
	0	**1**	2	3	2	3	3	2	5	4	3	4	4	5	5	5	5	Q
		0	**1**	**2**	**1**	**2**	**2**	**1**	**4**	**3**	**2**	**3**	**3**	**4**	**4**	**4**	**4**	P
			0	3	2	3	3	2	5	4	3	4	4	5	5	5	5	O
				0	1	2	4	3	6	5	4	5	5	6	6	6	6	N
					0	1	3	2	5	4	3	4	4	5	5	5	5	M
						0	4	3	6	5	4	5	5	6	6	6	6	L
							0	1	2	1	2	3	3	4	4	4	4	K
								0	3	2	1	2	2	3	3	3	3	J
									0	1	2	3	3	4	4	4	4	I
										0	1	2	2	3	3	3	3	H
											0	1	1	2	2	2	2	G
												0	2	1	1	1	2	F
													0	3	3	2	1	E
														0	2	2	3	D
															0	2	3	C
																0	**1**	B
																	0	A

Figure 4.9: The access-distance matrix for the graph of figure 4.5

Now we add to G, at step t+2, a new link, say (P B), where p ∈ V(G) and B ∈ V(G).

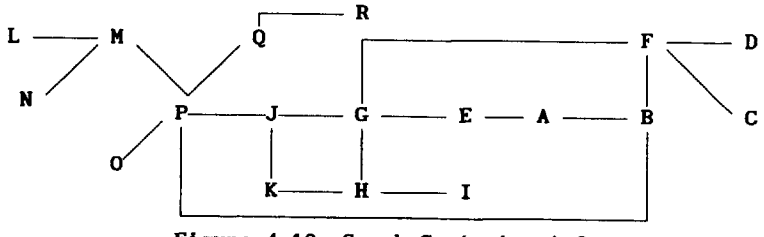


Figure 4.10: Graph G at step t+2

This time both nodes of the new link (P B) are in G. The edge (P B) then forms new circuits in G. As a result new tracks in the incremented graph G are present, and the distances corresponding to these new tracks are not valid any more. θ(P B) is the most obvious new track with a distance d(P B) equal 1, which is the weight of (P B). Other new tracks must be identified and their corresponding lengths must be adjusted according to the algorithm presented in the last section.

The cross section of the distance matrix corresponding to the nodes B and P, before and after the adjustment of the distance d(B P), is shown in figure 4.11.

	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A
Distance-list of the node **P:**	(2	1	0	1	2	1	2	2	1	4	3	2	3	3	4	4	4	4)
Distance-list of the node **B:**	(6	5	4	5	6	5	6	4	3	4	3	2	1	2	2	2	0	1)

	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A
Distance-list of the node **P:**	(2	1	0	1	2	1	2	2	1	4	3	2	3	3	4	4	1	4)
Distance-list of the node **B:**	(6	5	1	5	6	5	6	4	3	4	3	2	1	2	2	2	0	1)

Figure 4.11: The cross sections of the *ad-matrix* corresponding to the nodes P and B before and after the addition of (P B) to G

Following the procedure, in the next step we have to readjust distances of the new tracks originating from P and B, to detect and save the end-nodes of the tracks originating from P or B. Such nodes necessarily comply with one of the following inequalities.

$$d(B\ v_l) + w < d(P\ v_l),\ \forall l \qquad (12)$$

$$d(P\ v_k) + w < d(B\ v_k),\ \forall k \qquad (13)$$

And their new distances to P and B are:

$d(P\ v_l) := d(B\ v_l) + w$, $\forall l$ that comply with (12)

$d(v_k\ B) := d(v_k\ P) + w$, $\forall k$ that comply with (13)

Because the distance of a node to P or B complying with (12) and (13) is no longer valid, and the new track from such a node to the most distanced node (i.e., P or B) now crosses the other node (i.e. it includes the new link (P B)). Figure 4.12 depicts cross sections of the distance matrix corresponding to the nodes P and B, with distances that need to be readjusted underlined.

	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A
Distance-list of the node **P:**	(2	1	0	1	2	1	2	2	1	4	3	2	3	3	4	4	1	4)
Distance-list of the node **B:**	(6	5	1	5	6	5	6	4	3	4	3	2	1	2	2	2	0	1)

Figure 4.12: Distances (underlined) of the new tracks, originating from P and B, which require adjustment

Following the algorithm, the distances of P and B to the other nodes in G after the readjustment will be:

	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A
Distance-list of the node **P:**	(2	1	0	1	2	1	2	2	1	4	3	2	2	3	3	3	1	2)
Distance-list of the node **B:**	(3	2	1	2	3	2	3	3	2	4	3	2	1	2	2	2	0	1)

Figure 4.13: Distances of new tracks, originating from P and B, after the readjustment

The nodes corresponding to the end points of the tracks originating from B and P (i.e., the adjusted nodes relative to B and P) are collected in the following two lists respectively.

List_B: (J K L M N O Q R)
List_P: (A C D F)

Figure 4.14: End-points of the tracks originating from B and P

In the final step the distances between the list-B and list-P should be checked and readjusted, if necessary. The old distance of every node pair is checked, and if it is larger than the distance of the two nodes via the paths going through the new link (P B), then the new path crossing (P B) is the new shortest path for the corresponding node pair. To show the process, we take the node pair (F R).

The distance between these two nodes from the distance matrix of figure 4.6 is equal to 5 steps, but the length of the paths between these two via the link (P B) is equal to 4, sum of the lengths of the paths p(F B), p(R P), and p(P B). Hence, d(F R) need to be readjusted to 4. The process is the same for all combinations of node pairs corresponding to two node lists. The new distance matrix after the complete readjustment is:

R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A	
0	1	**2**	3	4	3	4	4	3	6	5	4	4	5	5	5	**3**	4	**R**
	0	**1**	2	3	2	3	3	2	5	4	3	3	4	4	4	**2**	3	**Q**
		0	**1**	**2**	**1**	**2**	**2**	**1**	**4**	**3**	**2**	**2**	**3**	**3**	**3**	**1**	**2**	**P**
			0	3	2	3	3	2	5	4	3	3	4	4	4	2	3	**O**
				0	1	2	4	3	6	5	4	4	5	5	5	3	4	**N**
					0	1	3	2	5	4	3	3	4	4	4	2	3	**M**
						0	4	3	6	5	4	4	5	5	5	3	4	**L**
							0	1	2	1	2	3	3	4	4	3	4	**K**
								0	3	2	1	2	2	3	3	2	3	**J**
									0	1	2	3	3	4	4	**4**	4	**I**
										0	1	2	2	3	3	**3**	3	**H**
											0	1	1	2	2	**2**	2	**G**
												0	2	1	1	**1**	2	**F**
													0	3	3	**2**	1	**E**
														0	2	**2**	3	**D**
															0	**2**	3	**C**
																0	**1**	**B**
																	0	**A**

Figure 4.15: The accesses-distances matrix after the complete readjustments

4.3 Diagnosing dynamics

TIs, discussed earlier in this chapter, are only capable of characterizing topological features of systems. In the study of many physical systems often a broader analysis of their complexity with respect to their components and the interrelationships between them is required. The complexity of a system is not always immediately evident from a large amount of information about the system without gross simplification of the data into an absorbable level. For example, given a building with a set of locations as a container of activities and actors responsible for them, many design activities demand deep understanding of dynamism of the building as a result its operation.

In dealing with such a problem Atkin [74a] [77] has developed a mathematical based method termed Q-analysis. Q-analysis provides a means for dealing with a large amount of data related to a set of objects in order:

- To identify the association degrees between the objects, and
- To partially hierarchically cluster them.

The mathematical considerations in Q-analysis, which is essentially combinatorial, center on the study of relations between finite sets. The overall

method provides a working algorithm which can process observational or theoretical data for identifying existence of relationship between a set of objects as components of a large system. The result of Q-analysis gives significant insight into the physical organization of a system as a collection of interrelated components, and extend the perceptions of investigators diagnosing such a system. Result of the Q-analysis may be used in conjunction with other knowledge of a system to diagnose an existing system, also to configure a system (design) with specific behavior. In this case, Q-analysis provides a powerful means for hierarchical abstraction of data before proceeding the process of design. The method has been successfully utilized in the investigation of the complexity of urban communities, medical diagnostics, art, university policies, etc. Atkin [Atkin74a] [74b] [74c] [75] [77] explicates the method and reviews many of its applications[2].

In this section a review of the method, and algorithms involved are presented. In the last section the method is shown in work by applying it on a design example related to this work. The original algorithm is only applicable to set of objects having binary relation. I have improvement the algorithm to widen its applicability to objects having non-binary relations and to increase its classification power in some respect. I will discuss this matters in sequel.

4.3.1 Q-analysis

Q-analysis is capable of analysis and presenting a holistic view on the relationships between components of a system taken as elements of a set. A set is a collection of finite objects characterized by two basic notions:

- The *Set-elements*: the set of member objects.
- The *Set-membership*: a proposition, which gives the membership degree between the elements of the set.

The mathematical expression, representing the relation between elements of a set, has been called *Simplicial complex* (or complex of simplices) [Atkin74]. A simplicial complex has an associated geometry which formally represents

[2] For detailed treatment of the subject See also the special issue of the International Journal of Man-Machine Studies, 1976.

the structure or organization of the simplex. Atkin has shown [Atkin74] [85] the application of set theory in the study of the structure of relations between sets of entities in social systems. Here, the intention is to show how the idea can be applied in the analysis of the relations between a set of locations in a building as a container of activities. Such a relation is identified based on the association degrees between location pairs in the building, as a result of the sharing actors between them. The interpretation of these associations leads to the identification of interaction potential between these location pairs, and partially hierarchical cluster them with respect to their association degrees.

Before to pursue the subject, some basic definitions and relations must be laid down. Assuming two finite sets:

$X = \{y_i,\ i=1 \text{ to } m\}$

$Y = \{x_j,\ j=1 \text{ to } n\}$

If a relation ρ exists between the elements of these two sets (e.g., y_i and x_j), this relationship is denoted by one of the following notations:

$\rho \subset Y * X$, or $y_i\ \rho\ x_j$, or $(y_i, x_j) \in \rho$

Similarly, a relation ρ^{-1}, the inverse of ρ, exists between the sets X and Y. This inverse relationship is denoted by $(x_j, y_i) \in \rho^{-1}$, whenever $(y_i, x_j) \in \rho$.

A relationship ρ between two sets can be represented by an incidence matrix $A = (\rho_{ij})$. An elements ρ_{ij} of this matrix, depending on the nature of the relation could be either a binary number or a weighted integer number. In the first case we have:

IF: $(y_i, x_j) \in \rho$

THEN: $\rho_{ij} = 1$

ELSE: $\rho_{ij} = 0$

A *weighted relation* matrix can be converted to several binary relation matrices through slicing processes. This process tests the values of specific rows, columns, or elements of the matrix against the value of a threshold parameter θ, and sets them to 0 or 1, respectively, depending whether the value is $\leq \theta$ or $> \theta$ [Atkin74].

If a particular y_i is related to just a member of x, then this is called *zero-order simplex*, written as 0-simplex. Similarly if y_i is related to two members of x, it is called a 1-simplex. In general y_i is called *p-simplex* if it is related to (p+1)-subset of X. Such a simplex is written as:

$y_1=<x_1, x_2, \ldots, x_{p+1}>$ or $y_1=\sigma_p$. Any subset of this (p+1)-subset also is a simplex which is said to be a *face* of the σ_p. The collection of simplexes identified by Y is called a *simplicial-complex K*, denoted by $K_Y(X; \rho)$. $K_Y(X; \rho)$ implies that xs provide the vertices, and ys the names or labels of the simplexes.

The largest value of q for which $\sigma_p \in K$ is then called the *Dimension* of K written as n = Dim. K. So, the dimension of K is equal to max q-value in K (x).

By applying the inverse relation ρ^{-1}, we get the conjugate simplicial complex $K_X(Y; \rho^{-1})$ in which case ys provide the vertices and xs the name of the simplexes.

Geometrical representation of a simplex

Geometrically a 0-simplex can be depicted as a point, a 1-simplex as a line, a 2-simplex as a triangle, and so on. We can say that a p-simplex $\sigma_p=<x_1, x_2,\ldots, x_{p+1}>$ may generally be thought of as a convex polyhedron with p+1 vertices corresponding to $x_1, x_2, \ldots, x_{p+1}$. The simplicial complex $K_Y(X)$, then, is geometrically represented by a collection of polyhedra which are possibly connected to each other by sharing faces. Such a geometrical representation gives better understanding of how the simplexes are interconnected to each other. In fact, it is this geometrical representation which has justified the vague term *structure* for describing the relation ρ between two sets [Atkin74]. The following figure shows geometrical representation of a simplex $y_1=<x_2, x_3, x_4, x_6, x_8, x_{10}>$

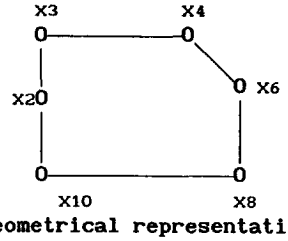


Figure 4.16: Geometrical representation of a simplex $y_1=<x_2, x_3, x_4, x_6, x_8, x_{10}>$

Q_connectivity and Chains of q_connection

Two simplices are said to be q-connected or q-near (i.e., associated), if

they share q+1 vertices. If we assume that each pair of simplices σ_p and σ_r in K is connected by a common face, then all simplices of K are said to be connected to each other directly by a common face or by a chain of q-connection or q-connectivity, when q is the least of connections between two simplexes within the chain. It is obvious that the maximum length of the chain of connectivities in a simplicial complex with n simplexes is n-1. The following figure shows the geometrical representation of 2 simplexes $y_i=<x_2, x_3, x_4, x_6, x_8, x_{10}>$ and $y_i=<x_1, x_3, x_4, x_6, x_8, x_9, x_{10}>$ which are 4-connected to each other.

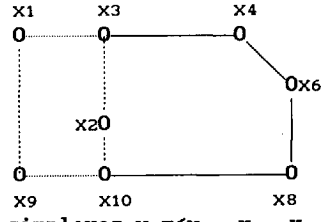


Figure 4.17: Two simplexes $y_i=<x_2, x_3, x_4, x_6, x_8, x_{10}>$ and $y_j=<x_1, x_3, x_4, x_6, x_8, x_9, x_{10}>$ 4_connected to each other

Now, if we assume another simplex, say $y_k=<x_1, x_9, x_{11}, x_{13}>$, this simplex is 1-connected to y_j because of the sharing points x_1 and x_9, while it lacks common face with y_i; so, y_k is said to be -1 connected to y_i. But, since y_j is 4-connected to y_i in one hand, and 1-connected to y_k in other hand, then there is a chain of 1-connectivity (i.e., degree 1 association) between y_i, y_j, and y_k. The connectivities between these three simplexes are depicted in the following figure.

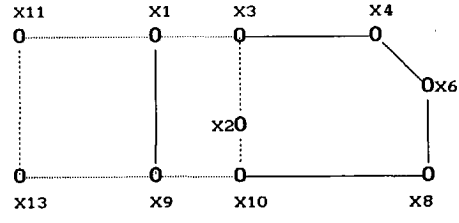


Figure 4.18: Three simplexes $y_i = <x_2, x_3, x_4, x_6, x_8, x_{10}>$, $y_j = <x_1, x_3, x_4, x_6, x_8, x_9, x_{10}>$, and $y_k = <x_1, x_9, x_{11}, x_{13}>$ with a chain of 1_connection

The structure vector

The process of identifying all distinct simplices which are q-associated with each other for all values of q from 0 to n (i.e., dimension of K) is called Q-analysis. If we denote the number of components of K which are q-connected by Q_q, then we analyze K by finding all the values of: Q_0, Q_1, Q_2, ..., Q_{n-1}, Q_n, where n is the dimension of k.

These values form a vector Q called *structure-vector* [Atkin74], which is denoted by $Q = (Q_0, Q_1, Q_2, \ldots, Q_{N-1}, Q_N)$. The position of each Q_i in Q represents the q- value, when Qo is the leftmost, and Q_n is the rightmost item, and each Q_i represents the number of components at the q-level of connectedness which is the number of q-connected simplices. $Q_i > 1$ is an indication that the complex K has Q_i parts in that level, which can be considered as the union of Q_0 disjoint complexes.

4.3.2 Q-analysis algorithm

The goal of Q-analysis, thus, is to find the shared faces between all pairs of simplices $y_j \in Y$ in $K_Y(X; \rho)$. If we denote our incidence matrix as A and its transpose by A^{-1}, then the product of a vector a_i by a_j is equal to the number of common vertices shared by simplexes a_i and a_j, hence the q-value of the face common to these two simplexes is obtained by deducting 1 from their vector product. This implies that the q-value of $K_Y(X; \rho)$ is obtainable by first evaluating the matrix product $A*A^{-1}$, and subtracting 1 from every element. Similarly q-value of $K_Y(X; \rho)$ is calculated by deducting 1 from every elements of $A^{-1}A$.

The complete process can be described in algorithmic terms as follows:

1- Form a m by n matrix A of (degrees of) associations between the x_s and y_s, with rows corresponding to number of y_s (i.e., m), and columns corresponding to the number of x_s (i.e. n).

2- If A is a weighted matrix, then apply a threshold value (slicing parameter) to convert A into a binary matrix.

3- Form $A*A^t$, a m by m matrix representing the number of shared points between all y_s in $K_Y(X; \sigma)$

4- Evaluate $A*A^t - \phi$, where ϕ is a m by m matrix of 1s.

5- Analyze the resulting connectivity (association) matrix, and identify hierarchical clusters of x_s with chains of association.

6- Form the structure vector.

A similar process can identify the sharing faces between all pairs of xs in $K_x(y;\ \sigma^{-1})$ and corresponding association table. In this case, step 3 and step 4 are as follows:

3- Forms A^t*A, a n by n matrix representing the number of shared points between all xs.

4- Evaluate $A^t*A-\phi$, where ϕ is a n by n matrix of 1s.

4.3.3 Application of Q-analysis on design data

This section presents examples of application of Q-analysis on design data related to this work. Three design activities were distinguished in chapter 2:

- A design evaluation,
- A design generation, and
- Activities allocation.

In dealing with above problems, one is always in need of the analysis of design data, and discovery of the following information:

- What is the complexity of associations between the actors?
- What are the degrees of associations (flow generation potentials) between the activities (locations)?
- Which locations are related with each other, and in what degrees?
- What are the hierarchical clusters of locations (activities) with respect to their flow generation potentials?

To show the capability of Q-analysis in providing above information, a set of data related a small design problem suffice the purpose. We may assume a set of activities {A1 A2 A3 A4 A5 A6 A7 A8 A9 A10}, and a set of actors: {G1 G2 G3 G4 G5 G6 G7 G8 G9 G10} with the same number of elements, with the following relationships holding between them. Where, ρ^{-1} is the inverse of ρ:

ρ: *"The actor: ... is responsible for the activity ..."*

"The actor: ... is present at the location: ..."

ρ^{-1}: *"The activity (location): ... is assigned to (contains) the actor: "*

The following table shows the activities and actors responsible for each activity.

Activity	Actors (Groups) responsible for an activity
A1	G1, G2, G6
A2	G2, G4
A3	G3, G1, G4
A4	G4, G1
A5	G5, G7, G4, G9
A6	G5, G7, G9, G10
A7	G6, G8, G9
A8	G6, G7, G8, G9
A9	G7, G10
A10	G7, G8, G9, G10

Figure 4.19: Activities and the actors responsible for each activity

Furthermore we assume the following weights associated with the actors (groups).

Actor (Group)	weight
G1	5
G2	5
G3	4
G4	6
G5	5
G6	5
G7	8
G8	5
G9	4
G10	7

Figure 4.20: Weights associated with the actors

Now, we have to obtain the structure of the two simplicial complexes $K_A(G; \rho)$ and $K_G(A; \rho^{-1})$ corresponding to the first and the second relationship. In the first case, the activities provide the names (labels) for our simplices (polyhedra) and the Gs provide the vertices of the polyhedra which ρ creates out of the two sets. The position of As and Gs are reversed in the second case. This means that the vertices of our polyhedra are named by As, where Gs provide the labels for the simplices.

In order to achieve such a goal, the first step is to set up an incident matrix of relationship between elements of the two sets. This

matrix should contain binary values of 0 and 1, where a 1 in a position of the matrix implies a positive tie between corresponding row (actor) and column (activity). But, there are weights associated with the actors, so, there are different degrees of associations between the activities and the actors. The weighted association table for activities and actors in our data set is as follows:

	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10
G1	5	0	5	5	0	0	0	0	0	0
G2	5	5	0	0	0	0	0	0	0	0
G3	0	0	4	0	0	0	0	0	0	0
G4	0	6	6	6	6	0	0	0	0	0
G5	0	0	0	0	5	5	0	0	0	0
G6	5	0	0	0	0	0	5	5	0	0
G7	0	0	0	0	8	8	0	8	8	8
G8	0	0	0	0	0	0	5	5	0	5
G9	0	0	0	0	4	4	4	4	0	4
G10	0	0	0	0	0	7	0	0	7	7

Figure 4.21: The weighted relation matrix for the set of actors and activities

According to the Q-analysis algorithm, if we have a weighted relation matrix, then we have to choose a threshold for converting the table into a binary one. But, the weights are important in identification of degrees of associations between the objects. For example, in the case of evaluation of a design, weights are important in identifying the potential flow degrees between the location-pairs, which in turn has significant influence on the evaluation result of a design with respect to a set of social norms. The Q-analysis ignores these weights. The following is an incident matrix representing the relationship ρ between the two sets. This table is prepared by taking the slicing parameter value $\theta \geq 0$. This means that any value > 0 is taken as a positive relation and other values as 0.

	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10
G1	1	0	1	1	0	0	0	0	0	0
G2	1	1	0	0	0	0	0	0	0	0
G3	0	0	1	0	0	0	0	0	0	0
G4	0	1	1	1	1	0	0	0	0	0
G5	0	0	0	0	1	1	0	0	0	0
G6	1	0	0	0	0	0	1	1	0	0
G7	0	0	0	0	1	1	0	1	1	1
G8	0	0	0	0	0	0	1	1	0	1
G9	0	0	0	0	1	1	1	1	0	1
G10	0	0	0	0	0	1	0	0	1	1

Figure 4.22: Incident matrix based on the threshold $\theta > 0$, representing the relationship ρ between the set of actors and activities

To get the connectivity picture of our complex $K_G(A;\ \rho)$ or $K_A(G;\ \rho^{-1})$ we continue with the Q-analysis algorithm, and get the following connectivity table for the complex $K_G(A;\ \rho)$.

	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10
G1	2	0	0	1	-	0	-	-	-	-
G2		1	-	0	-	0	-	-	-	-
G3			0	0	-	-	-	-	-	-
G4				3	0	-	0	-	0	-
G5					1	-	1	-	1	0
G6						2	0	1	1	-
G7							4	1	3	2
G8								2	2	0
G9									4	1
G10										2

Figure 4.23: Connectivity table representing connectivities in the simplicial complex: $K_G(A;\ \rho)$

Now, we can extract q-connectivity values together with corresponding components from the above connectivity table, and order them in terms of their q-values in the following partially hierarchical table. Q_q-values in this table represents the number of q-connected clusters of actors.

q_value	Q_q_value	Components.
4	Q4=2	{G7} {G9}
3	Q3=2	{G4} {G7 G9}
2	Q2=4	{G10 G9,G8,G7} {G6} {G4} {G1}
1	Q1=3	{G10, G9, G8, G7, G6, G5} {G4,G1} {G2}
0	Q0=1	{ALL}

Figure 4.24: Q-analysis table, displaying the relationship between the actors

From above table we have the structure vector Q = {2, 2, 4, 3, 1} that gives a holistic picture of the relationship between all actors responsible for the activities. We can attribute the following analytical information to the result of the Q-analysis:

1- The Q vector shows that, we have one subset of actors at least 0-related to each other, 3 subsets at least 1-related to each other, 4 subsets at least 2-related, and so on.

2- There are two groups, G9 and G7, which are 4-simplexes (i.e., q-value=4). Each of these groups are responsible for four activities. The structure vector also suggests that these two groups are the most active groups, not in relation to the other groups but within themselves. Another implication of the structure vector is that G7 and G9 do not have great significance in terms association with other groups, at Q4 level. No other groups has any implication with respect to design activities, at Q4 level.

3- At Q3 level, G7 and G9 become associated, and there is another isolated group G4. These three groups do not have interaction with the others at this level. G7 and G9 have some significance with respect to design activities. Were we assumed that actors are bound to locations of designs, then {G7 G9} had the implication that G7 and G9 are 3 associated, or having a potential of generating a flow of degree 3. A design process under the influence of privacy norm would try to locate them as close as possible, while a similar process under the influence of the community norm would attempt on the contrary.

4- In the next level (q-value=2) we have 3 isolated groups: {G6}, {G4}, {G1}, and a group-set {G10, G9, G8, G7}. G8 and G10 join {G7 G9} at this level to create a new cluster with a chain of association of degree 2. The same argument as above is applicable to this group as a whole, in terms of the design activities. The fact that Q-analysis only identifies chain of associations between objects, must be given a cautious taught, specially with respect to its role in design evaluation. A chain of association does not provide enough information about the degrees of association between all group-pairs in a cluster. So it is not valuable for a design evaluation.

5- At q=1, more groups are glued together. At this level there is only

one isolated group G2, and two group sets {G10, G9, G8, G7, G6, G5} and {G4, G1}. An argument the same as above is valid for these group sets.

6- At the next level (q-value = 0), all activities are connected to each other, thus we have a 0-simplex of all activities.

Above analysis, as was mentioned earlier, was under the assumption that actors are identified with the locations. In practice one is interested in identifying a activity with a location. In such a case the simplicial complex $K_A(G; \rho^{-1})$ serves the purpose better. We can obtain this complex by the operation $\Lambda^t\Lambda$ to arrive at the following table:

	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10
A1	2	0	0	0	-	-	0	0	-	-
A2		1	0	0	0	-	-	-	-	-
A3			2	1	0	-	-	-	-	-
A4				1	0	-	-	-	-	-
A5					3	2	0	1	0	1
A6						3	0	1	1	2
A7							2	2	-	1
A8								3	0	2
A9									1	1
A10										3

Figure 4.25: Connectivity table for the simplicial complex: $K_A(G; \rho^{-1})$

The q-connectivities in $K_A(G; \rho)$ extracted from $\Lambda^t\Lambda$ (above table) is as follows:

q_value	Q_q_value	Components.
3	Q3=4	{A10} {A8} {A6} {A5}
2	Q2=3	{A10 A8 A7 A6 A5} {A3} {A1}
1	Q1=4	{A10 A9 A8 A6 A5} {A4 A3} {A2} {A1}
0	Q0=1	{ALL}

Figure 4.26: Q-analysis table displaying the connectivity structure within the activities

From above, we have the structure vector Q={3,5,4,1}. The Q-analysis on above structure shows that:

1- There are four activities (i.e., A10, A8, A6, and A5) each separately a 3-simplex (i.e., q-value=3). This is as a result of responsi-

bility of four distinguished group-sets for each of these activities. The Q-analysis at this level does not provide any information important in design activities.

2- At level q-value = 2, all the activities appeared at Q_3 become associated and form a cluster. This association is, again, as a result of a chain of association between pairs of activities. The implication of the cluster of activities identified at this levels is straight forward. Here, the same as arguments carried out with respect the complex $K_G(A; \rho)$. The identified clusters with a high degree of association have in turn a high potential degree of interaction. A design with respect to the privacy norm would try to locate these locations as close as possible (e.g., locate them adjacent with respect to each other). Again, the pair-wise associations between these activities, important in design evaluation and activity allocation, are not known. At this level there are also two isolated activities each with a degree 2 association with itself.

3- At the q-value=1, beside the identified set at the earlier level, A3 and A4 also become associated. There is still no relation between the activities A3 and A4 at this level, and between them and the cluster of activities identified at level 2.

4- The elements of the set are not connected until we reach the level of q=0. If it would happen that even in q=0 the value of Q_0 was, for example 2, then, this would mean that our set of activities was divided in two distinct disconnected sets, and that there was no association between them at all. Such case has some social implication in terms of a building design. For example, a design with a privacy behavior requires allocation of the activities in these two set in two groups of separate locations. Such strategy would prevent the unnecessary interactions between the groups in two sets moving around. A design with a community behavior requirement, on the contrary, demands mixing of these two groups for increasing the social amenity of the building.

4.3.4 A modified algorithm for Q-analysis

The examples, given above, and their corresponding analysis clearly indica-

ted the analytical power of Q-analysis in explicating hidden information about objects and their relations, also partially hierarchical cluster them. However, two major shortcomings were evident in this method at least with respect to design activities:

- The influence of weight factors on the association degrees was neglected.
- The association degree between a location-pair was not known.

Both of these information have major rôle in all design activities discussed in this work. For examples, without knowing direct relationships between the location pairs a reliable evaluation of a design is not possible. The same information is needed in allocation of activities on locations of a design under the influence of the social norms. The information provided by the Q-analysis is obviously valuable to design activities, but are not enough. The identification of direct association-degree between the location-pairs can have much more significant value than the chain of association-degree between them. Such an information has significant impact on choosing a course of action during a design process or evaluation of a design with respect to the social norms. In fact, identification of a chain of relation between a set of objects, does not guarantee the existence of a relation between all element-pairs in the set at all.

The following revised algorithm eliminates both of the deficiencies mentioned in the Q-analysis.

for $K_y(x; \sigma)$:

1- Form a m by n matrix A, representing degrees of associations between the x_s and y_s, with row corresponding to the y_s (i.e., m), and n corresponding to the x_s (i.e., n).

2- Form $A \alpha A^t$, a m by m matrix representing the association degrees between all y_s, where α is the following operation for each row of A and a column of A^{-1}:

$$\alpha = \sum_j (a_{ij} * a^t_{ji})^{1/2}$$

3- Analyze the resulting connectivity (association) matrix, and identify clusters of pairs of x_s with identical association degree.

A similar process can identify the sharing faces between all pairs of x_s in

$K_x(y; \sigma^{-1})$ and corresponding association table. In this case, step 2 is as follows:

2- Form $A^t \alpha A$, a n by n matrix representing the total degrees of association between all x_s, where α is the following operation for each row of A^t and a column of A.

$$\alpha = \sum_j (a^t_{ij} * a_{ji})^{1/2}$$

The influence of the weights, as displayed by the algorithm, is enforced by multiplying the components the two matrices, and then taking the square root of the result after each multiplication.

Application of above algorithm on the complex $K_A(G, \rho)$ yields this connectivity table.

	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10
A1	15	5	5	5	0	0	5	5	0	0
A2		11	6	6	6	0	0	0	0	0
A3			15	11	6	0	0	0	0	0
A4				11	6	0	0	0	0	0
A5					23	17	4	12	8	12
A6						24	4	12	15	19
A7							4	14	0	9
A8								22	8	17
A9									15	15
A10										24

Figure 4.27: Connectivity table for the simplicial complex: $K_A(G; \rho^{-1})$

Now, by an appropriate scanning of the above connectivity table, we can identify location pairs with identical degree of association, and collect them into separate groups, to be called *partially hierarchical clusters*. The following table shows partially hierarchical cluster of activity pairs for the simplicial complex $K_A(G; \rho)$.

Association degree	Partially hierarchical clusters of location
19	{(A6 A10)}
17	{(A5 A6)(A8 A10)}
15	{(A6 A9)(A9 A10)}
14	{(A7 A8)}
12	{(A5 A8)(A6 A8)}
11	{(A3 A4)}
9	{(A7 A10)}
8	{(A5 A9)(A8 A9)}
7	{(A9 A10)}
6	{(A2 A3)(A2 A4)(A2 A5)(A3 A5)(A4 A5)}
5	{(A1 A2)(A1 A3)(A1 A4)(A1 A7)(A1 A8)}
4	{(A5 A7)(A6 A7)}

Figure 4.28: A partially hierarchical clusters of locations for the simplicial complex $K_A(G; \rho^{-1})$

The element of each cluster in above table represents the activity pairs having the same degree of association. For example there are two location pairs, that are (A6 A9) and (A9 A10), each with an association degree of 15. Note that collapsing these two pairs together yield the set {A6 A9 A10} with three element having a chain of 15 connections. This table is named "partially hierarchical" since it presents elements of the set at different levels of associations. This table conveys a much more clear picture of the connectivities between the activities, than the connectivity table produced by the original Q-analysis algorithm. In addition, the influence of the actors weights also has been accounted for this result. The implication of this table, for design activities, is more straight forward than the previous tables. The direct associations between the activity pairs, provides sufficient information for all design activities discussed in this work. A detailed discussion will be given in the following chapter.

CHAPTER 5
THE IMPLEMENTATION OF TOPGENE

Chapter 3 and chapter 4 discussed major techniques and methods involved in the implementation of TOPGENE. This chapter discusses the implementation details of TOPGENE working system capable of generating, analyzing, and evaluating designs. The chapter, among other subjects, discusses the architecture of TOPGENE, the rôle of method and techniques, and the algorithms involved in the implementation of the system.

5.1 The architecture

TOPGENE is designed for generating topological patterns of buildings with respect to a set of social norms. These norms, defined in chapter 2, are *community, privacy, circulation-cost, and intervening opportunity* norms. The system also can evaluate existing buildings with respect to the norms.

TOPGENE has two basic modules corresponding to the two above mentioned design sub-processes. A design generator module, and a design evaluator module. The first module is responsible for generating designs (topological patterns of buildings) with respect to a single social norm or a combination of them. The evaluator module can analyze generated designs in terms of their expected behaviors with respect to the social norms, and evaluating existing designs with respect to the same norms.

TOPGENE's modules are interrelated. The design generator module calls the evaluator module for analysis, diagnosis, and calculating the performance values of the generated designs. The evaluator module, on the other

hand uses the design generator module to generate yardstick solutions for comparing their behavior with the input designs (i.e., evaluating the input design).

The relationship between the plan generator and plan evaluator module is depicted in the following figures.

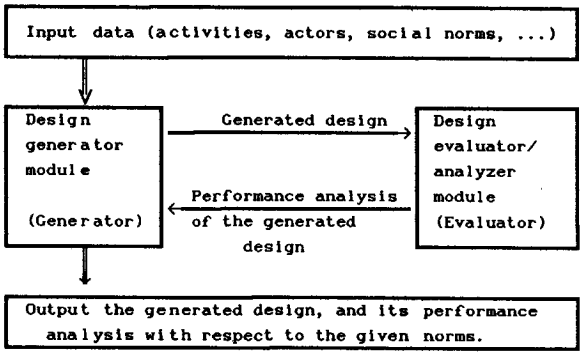


Figure 5.1: The Generator-Evaluator relationship

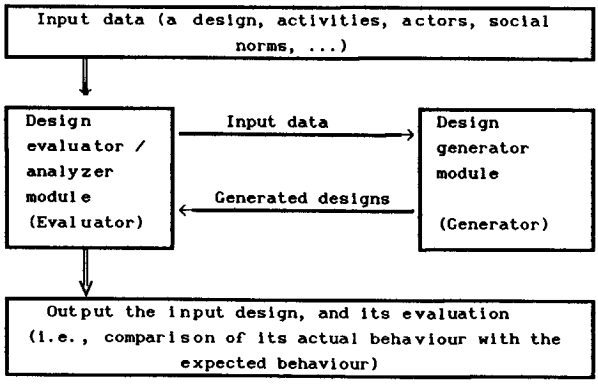


Figure 5.2: The Evaluator-Generator relationship

TOPGENE accepts design generation problems in the following forms:

Input:

- The input set A or B:
  - A: . A set of locations or activities.
    . Actors responsible for each activity.
    . Weight associated with each actor (optional).
    . Expected or actual flow index between different locations-pairs (optional).
  - B: . A set of locations or activities.
    . The expected or actual flow between different locations.
- A set of social norms reflecting the expected behavior of the design to be generated.
- (Ground-plan) type of design.
- Branching degree of each location (optional).

Output:

- A design (connectivity pattern of a building), labelled with the activities, that has a relatively high performance potential with respect to the set of given norms.
- Analysis and diagnosis of the generated design with respect to the given norms.

TOPGENE accepts design evaluation problems in the following forms:

Input:

- A design (connectivity pattern of a building) labeled with activities.
- Actors responsible for each activity, OR
  Expected or actual flow index between different location-pairs.
- Weights associated with each actor (optional).
- The anticipated (or actual) flow between different location-pairs (optional).
- A set of norms reflecting the expected behavior of the design.
- (Ground-plan) type of yardstick designs to be generated.
- Branching degree of locations in the yardstick designs (optional).

Output:

- Evaluation of the given design with respect to the set of social norms.

Design sub-processes (solid lines) that are carried out by TOPGENE are depicted in the following model. This model was discussed in chapter 2.

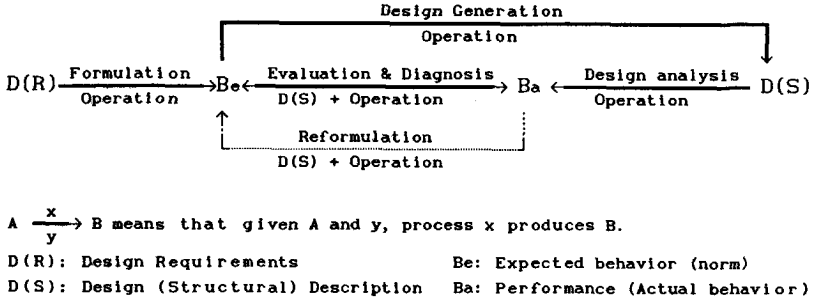


Figure 5.3: Design sub-processes (solid lines) carried-out by TOPGENE

The processes carried out by TOPGENE are based on the intuitive works of architects in arranging locations of a building or evaluating an existing design with respect to a set of social norms . For example, decisions about the (connectivity) structure, and operation (arrangement of the locations and activities on the structure) with respect to the social norms or evaluation of an existing design, one way or the other, demands consideration of interaction potentials between the activities. In practice, the interaction potentials between the activity pairs in a design are either extracted from the corresponding design statement, or intuitively estimated by the designer. TOPGENE acquires this information either directly form the user or extracts it from a design data by applying the Q-analysis method.

TOPGENE has been implemented on an IBM PS/2 model 80, and is written in GCLISP V 2.0, a dialect of Common LISP. GCLISP V. 3.0 is a full implementation of the Common LISP language, but the lower versions have some limitations including multi-dimensional array manipulations and graphic capabilities. TOPGENE accepts data, analyzes them, fits them into appropriate model, consults with the user if necessary, takes appropriate actions, and presents the result in appropriate form using a heuristic graph displayer [Watanabe-89]. More specifically, TOPGENE includes all procedures and functions necessary for:

- Developing a friendly user interface.
- Providing help to the users.
- Accepting data from the user.
- Checking the completeness of data.

- Treating the input data with the Q-analysis method.
- Inferring the circulation flow potentials between the location pairs.
- Hierarchically clustering of activity pairs with respect to their flow generation potentials.
- Path finding.
- Enforcing the planarity constraint on designs.
- Enforcing the branchiness constraints on designs.
- Graphical display of the generated designs.
- etc.

The approximate size (including comment lines) of various parts of the TOPGENE are as follows:

Module	Size in K-byte
Design Evaluator	35
Design configurer	175
Q-analysis	15
Graph manipulating algorithms	45
User interface and graphics	245
Other procedures and functions	55
Total	570

5.2 Assumptions

TOPGENE as a working system is implemented based on the following assumptions:

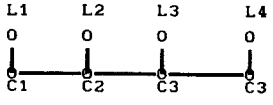
Constant (time independent) flow

The existence of association between two locations implies movement of people between them. The pattern of movement between two locations having several access paths may vary depending on their usage time, also several other factors. In other worlds, distribution of flow between two locations on paths connecting them may be time dependent and probabilistic. TOPGENE ignores the effect of time on flow degrees. This may be interpreted that behavioral aspects of buildings are considered by TOPGENE over a long period. This is completely a legitimate assumption, since in practice a table of flow between locations usually reflects observation of flow over a fixed period, and not at a moment. However, the introduction of time factor

to the process according to my judgment neither increases the complexity of the design problem nor requires a completely different approach in attacking the problems. The design generation, and evaluation problems with respect to the social norms discussed, based on a heuristic approach even at the presence of time dependent flows one way or the other demands data analysis and data abstraction similar to the one used in this work.

Proximate flow

As a flow directly disturbing a location, have effect of the total behavior of a building, so does an indirect disturbance because of a nearby (proximate) flow. A nearby flow, although may not be as effective as a direct flow, but have some contribution to social behavior of a building. An example clarifies the claim. If we assume that, for example, a location L2 is recommended as an intervening location for two locations L1 and L4, then the following single-loaded design is debatable whether to be considered as a valid solution or not. This solution according to the definition of the intervening opportunity norm is not a good solution with a good performance with respect to this norm. The reason is that this norm, as was defined in chapter 2, depends only on the direct meeting of particular flows an intervening point. This also is the way that TOPGENE calculates intervening opportunity utility of a design. TOPGENE considers a location as an intervening location for two other locations, if it is on a shortest path between the two locations. However, in most of the real designs Intervening locations are not necessarily located on a shortest path and consequently they do not have to be directly exposed to flows , but it suffices, or even it is required for an intervening location to be in the proximity of such flows. This is so in most of the practical designs. Similarly, if there is a high degree of flow between two locations L1 and L4, then without a doubt this flow passing locations C2 and C3 effect locations L2 and L3, and have contribution to the privacy and community behavior of these locations and the total design.



The current version of TOPGENE does not consider the effect of proximate flows on building patterns, but may be incorporated into future versions of

the system.

Shortest paths, and closed-world assumptions

The basis of a design analysis, a design diagnosis, and a design evaluation as it is carried out by TOPGENE is the detection of paths between locations. recognition of paths between locations and the flow potentials between them both play important rôle in these processes. TOPGENE assumes distribution of flow on the shortest paths between two locations. People, in general, have tendency in taking the shortest paths between two locations. Several factors may overrule this assumption. For example, outsiders visiting a building may not know the shortest paths between locations. There also may be several incentives for people in taking paths longer than the shortest ones, or several other psychological factors affecting decisions of people in choosing a path. The conclusion is that the usage of different paths may be different in different buildings. TOPGENE neglects the effect of these diverse phenomenon in path usage, and assumes the distribution of flow on the shortest paths between locations in a design. This is quite different from considering the psychological aspects associated with the actors using a building, or considering the probabilistic behavior for actors in a building.

The analysis of building systems by TOPGENE also is based on closed-world assumption. That is to say that TOPGENE investigates the behavior of buildings under the influence of only the inside forces. Of course, TOPGENE allows presentation of outside of a building as a location having interaction with inside locations. Such a location is considered by TOPGENE the same as other locations having logical relations with inside locations.

Topological distances

In a graph representing the connectivity property of a building, the nodes represent the locations and the links depict accesses between locations, which might be a door or imaginary barrier. Here, the metric distances, if to be considered, are in fact dimensions of locations collapsed into a node in the connectivity pattern or metric distances between center of two adjacent locations. The metric distances between locations have little effect in the study of social aspect of buildings, except for the circulation cost. This norm is under direct influence of metric distances

between locations. However, the current version of TOPGENE does not consider these distances while calculating behavior of a building or generating a design. TOPGENE considers only the topological distances in a design. This is the same as saying that a distance between two locations is taken as the number of links between them. The incorporation of metric distances in TOPGENE is an easy and trivial task to be considered in future version of the system. Again, the current approach for the implementation of TOPGENE is independent of the use of metric distances in a design, and the process of involving metric distances in a design can take place prior to the analysis of data carried out by the Q-analysis procedure, but taken into consideration by this procedure at the later stage by this procedure.

With this assumptions, TOPGENE proceeds a design analysis and design evaluation as described bellow.

5.3 Design types generated by TOPGENE

TOPGENE can generate various design types, depending on an user requirements, in response to a design problem . TOPGENE allows an user to decide the maximum branching degree of each location, the use of knowledge base of recommended and prohibited accesses in buildings, and the set of social norms as well ranking of the norms in any order. These flexibilities of the system together with its ability in analyzing and diagnosing a generated design, pose it as a perfect bench-work system for designers.

TOPGENE uses different approaches in generating designs depending on the number of the norms and (ground-floor) types of designs. We may categorize TOPGENE's algorithms for generating different types of designs according to whether a single norm or multiple norms are defined for design processes. The later class of algorithms also may be used to solve the first class of the problems. This was not obvious at the beginning of this research. Besides, test results show that specialized algorithms for generating sub-optimal designs with respect to single norms are more powerful than other class of algorithms. The reason is that these designs are closer to optimum designs compared with designs generated under the same conditions but by multi-norm algorithms. The following figure shows design types that are generated by TOPGENE with respect to a single norm or a combination of norms.

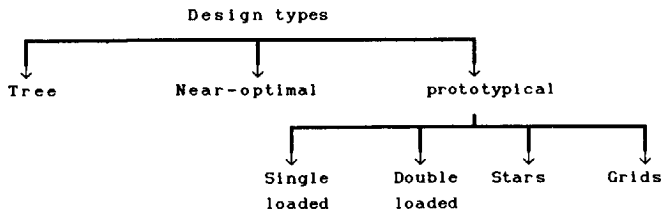


Figure 5.4: Design types generated by TOPGENE

I define a near-optimal design as a design with a performance in the neighborhood of an optimal design. A near-optimal design may range from a simple linear-tree type to a complicated design with circular paths. For example, a near-optimal design with respect to the norm community is in the form of a linear-tree, since a building with a linear-tree topology has potential for creating maximum community utility. On the other hand a design with respect to the norm privacy / circulation-cost always tends towards a compact (highly branched) design with cycles. TOPGENE distinguishes near-optimal designs for different norms, and generates them if requested by the user.

A tree type design is a near-optimal design but without any cycles. Depending on the norm, tree designs range from a simple linear-tree to a complicated one with a high degree of branchiness on some locations. The argument behind a tree type design is similar to the argument carried out for above near-optimal solution.

The near-optimal and tree type design with respect to multiple-norms are carried out under the following constraints. These constraints contribute to generation of more realistic design solutions.

- Branchiness for each location,
- Planarity (only for near-optimal type design),
- Knowledge base of recommended links, and
- Knowledge base of prohibited links.

The branchiness of each location is an user controllable constraint. The system allows setting of the same branchiness for all locations, or differently for each location.

The knowledge base consists of two separate files of recommended and

prohibited links in buildings. The use of these knowledge bases also is optional, and an user can turn them on or off separately. TOPGENE currently can learns about access properties of buildings by rote learning which is direct feeding of the access information into the system. The system may be improved so that this knowledge is accumulated by analyzing new designs that are presented to the system for an evaluation, or from designs stored in a precedent-base.

A *prototypical* type design is a repetitively used building type such as: a linear-tree (Row), a single-loaded, a double-loaded, a star, or a grid type design. TOPGENE, allows two choices for each prototypical design: with and without auxiliary location(s). By an auxiliary location, here, a location such as a hall or corridor is intended. These locations are not usually assigned to an activity. The choice of auxiliary locations for all prototypical solutions and with respect to all combination of norms is the same. So, further explanation will be skipped in further discussions, unless it is felt necessary. Examples of these design types will be given in the next chapter.

5.4 Knowledge Representation

A systematic method had to be found to encode the architectural information in terms of a data structure. There are several possibilities for TOPGENE. Topological patterns may be represented as an adjacency matrix, a linked list or a property list. The representation system must also take into account different levels of information or knowledge which play a rôle during the process of generating or evaluating designs.

TOPGENE is implemented based on AI techniques and mathematical modeling. The hybrid approach has the advantages of both in attacking the problems, but it often requires multiple knowledge representation techniques. AI emphasizes heuristics while a mathematical model can represent and manipulate mathematical characteristics underlaying a problem. Knowledge may be represented and used both in numerical or symbolic form, as appropriate to a task. However, AI is mostly known for its emphasis on symbolic knowledge representation.

The heuristic approach towards modeling of a process is mainly for reducing the search complexity. TOPGENE uses heuristic rules to circumvent

the complexity associated with the architectural design problems at topological level. TOPGENE makes inferences from knowledge of structure and behavior of buildings, by exploiting heuristics that relates the structural description of buildings (i.e., topological patterns) to their behavior in terms of social performances.

Symbolic information is used to reason about the qualitative relations in a building system, while the quantitative information is used to reason about the quantitative aspects of a design problem. The partially hierarchical clusters of location pairs are examples of symbolic information with implicit qualitative information about a design. Another example of qualitative information is the knowledge of recommended and prohibited links. Qualitative analysis and qualitative information can resolve qualitative ambiguities that may not be resolved by quantitative data. For example, the flow potentials between location pairs are key criteria in arrangement of location pairs relative to each other and with respect to social norms. Yet, experiments with early version of TOPGENE showed that quantitative information alone does not provide realistic designs. The use of an experimental knowledge base of prohibited and recommended links reduced the deficiency of the system in this respect. The integration of TOPGENE with a broader knowledge base of design could further improve the system.

Representation of design knowledge:

TOPGENE, as a heuristic program, uses rules of thumbs in relating and coding the structural-behavioral relationship in a building, and in simulating the architectural reasoning processes related to the social norms. IF-THEN inference rules are the medium for capturing and representing heuristics in programming languages. TOPGENE's heuristic rules, to be discussed, are embedded in procedures.

TOPGENE has to apply various processes on topological information for different purposes. Each process often demands a representation scheme (i.e., data structure) appropriate to itself. So various processes demand various data structure (representation). For example, most of the efficient path-finding algorithms on graphs work only on adjacency or property list representations of graphs (connectivity pattern of buildings), while efficient algorithms dealing with distances in graphs only work on distance matrix. This has forced me to choose several representation techniques in

representing topological information. While multiple representation is computationally more efficient, the impact of such an approach in terms of space efficiency is negligible.

TOPGENE specifically uses the following data structure for representing connectivity patterns of buildings.

Access list:

Many graph algorithms such as efficient planarity testing algorithms [Booth76] or [Nishizeki88] demand representation of graphs in the access list form. The access list representation also was found suitable for passing topology of buildings to the graph displayer routine for visual representation of the generated designs, and representation of recommended and prohibited accesses for storing them in the TOPGENE's knowledge bases.

Property list:

Most of the path finding algorithms work more efficiently on linked list representation of graphs than other types of representation. Property lists in the language LISP are akin, but more powerful than the linked list data structure in procedural languages. TOPGENE has to resort to temporary conversion of access list representation of a topological pattern to property lists representation when a process such as path-finding requires so.

Distance-matrix:

TOPGENE needs to keep track of distances in dynamically growing designs (topological patterns) during a design generation process. Distances of partial designs are used for diagnosing the graph structure in terms of its topological distances and finding the eccentric locations with respect to a location. Distance matrices also are found very useful in the path finding algorithm for early pruning of undesirable search paths while searching for the shortest paths between two locations.

5.5 Data abstractions

TOPGENE uses Q-analysis as a mathematical means to infer and make explicit the circulation flow potentials between the location (activity) pairs in a building from partial incomplete data. The method, as discussed in chapter

4, also allows clustering of the location-pairs with respect to their flow potential degrees in a partially hierarchical manner. Identification of the circulation flow potentials and generation of partially hierarchical clusters of activity pairs are important factor in the arrangement of activities on the pattern of a building with respect to the social norms. This information also is important in quantitative evaluation of existing designs. In the absence of the Q-analysis, TOPGENE had to resort to design generation and evaluation based on qualitative knowledge of the building design.

The Q-analysis procedure in TOPGENE takes the set of activities, the set of actors responsible for the activities, and weights associated with the actors (if available) as inputs and produces several levels of information important to TOPGENE. The result of the first phase of Q-analysis is identification of potential interactions between the activity pairs. This information derived from the degree of association between the activities, contributes to the social behavior of a building, and serves as numerical information in evaluating a design with respect to the social norms.

The information provided by the first phase is used in organizing the activity pairs into clusters of location pairs with identical degree of association. A location (activity) might be present in several clusters, depending on interactions that it has with the other locations. The possibility of presence of an activity in several levels, because of its different degrees of association with other activities, attributes this clusters of activity pairs as a partially hierarchical cluster. TOPGENE's heuristic rules use the partially hierarchical clusters to generate a design with respect to the social norms. These rules are only powerful if a higher level of data abstraction over the activity-pairs exists.

The following figure shows different levels of data abstraction carried out by Q-analysis.

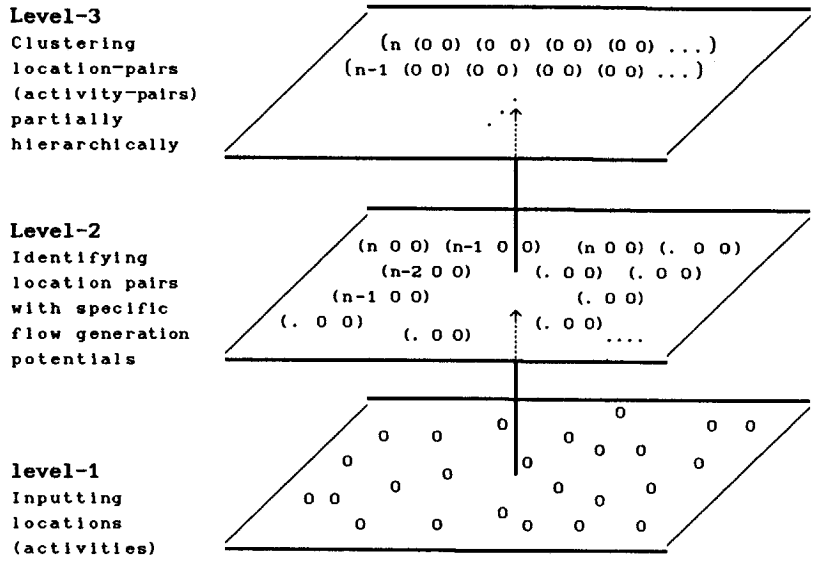


Figure 5.5: levels of data abstraction by Q-analysis

5.6 TOPGENE's approach to problem solving

TOPGENE uses a hill climbing strategy in generating designs. This approach fits in the state-space search paradigm, also the negotiation based problem solving method, discussed in chapter 3. The analysis of the components of the system and the way they are interrelated with each other, and describing its underlaying methodologies, helps better understanding of the system.

TOPGENE design generator module has four task-managers for accomplishing its job. A task-manager for generating designs with respect to a single norm, and three other task-managers for generating different designs with respect to a combination of norms.

The single norm task-manager is responsible for calling different single-norm task executers. A single-norm task executer is responsible for generating a design or preparing information for doing so. For example, the single-norm task executer for generating a near-optimal type or a tree type design with respect to the norm privacy produces directly such a design based on a hierarchical cluster of location pairs. Other single-norm task executers, including the single-norm task-executer for generating prototy-

pical type designs with respect to the privacy norm, generate a stream or streams of locations before generating a design. This stream is list of locations, linearly arranged in such a way that it has a high performance potential with respect to that norm. The generated stream(s) is then fed to a pattern generators to be used for labelling of a pattern.

The information sent to and received from a single norm task-manager are as follows:

Information sent:

- A partially hierarchical clusters of locations-pairs (activity-pairs), or
- A list of intervening opportunity triplets of locations (or activities) ordered in terms of the flow generation potential between their bracketing item the triplets.

Information received:

- A partial intermediate stream(s), or a final stream(s) of locations.

Again, a stream or streams of locations (or activities) are produced only if prototypical design (i.e., linear-tree, single-loaded, double-loaded, stars, and grids) with respect to single norms or a combination of norms is demanded. The arrangement of locations (or activities) on prototypical patterns are carried out by several pattern generators. TOPGENE has only three task-executers for generating stream(s) of locations (or activities), and a series of prototypical pattern generators accessible to all task-managers. To summarize the subject:

- A request for any prototypical solution is routed to its corresponding task-manager.
- A task-manager first ask a stream generator for a stream of locations in the design.
- The generated stream is passed to a pattern generator, responsible for a particular prototypical pattern, for labelling of a particular pattern with the locations (activities) in the stream.

The three multi-norm task-managers are responsible for generating patterns with respect to a combination of norms. These task-managers execute their

task by calling several task executers, each responsible for improving a partial design with respect to a specific norm. The norms privacy and circulation-cost are not conflicting, hence, they have the same task-executers. TOPGENE calls the task executers iteratively based on the order defined by the agenda of norms. The agenda of norms is part of the task-manager and contains a list of norms in an order defined by the user. The user is allowed to repeat any norm in the list. In this way, the priority of norms, and their degrees of contributions in affecting a design may be varied. So, different designs could be generated for the same set of data. The multi-norm task-managers and the task-executers communicate with each other under the following protocol:

Information send from a task managers to a task executer:

- A partially hierarchical cluster of location-pairs (or activity-pairs), if community, privacy, or circulation-cost norms, or a list of IO triplets of locations (or activities) ordered in terms of the flow generation potential between the IO's bracketing activities, if the IO task-executer:
- A partial design.
- A list of processed locations

Information send from a multi-norm task executers to a task manager:

- The improved partial design.
- Updated list of the processed locations.
- A processed location-pair.

Each task-executer, upon receiving a process call, is allowed to improve the partial design received by incrementing it with at most a location pairs, or a triplet of locations, if the intervening opportunity norm. Here by the word "improvement" both the structural and behavioral improvement of the partial design is intended. For this reason, such process is best characterized as a hill climbing process, that in a step-wise manner the overall performance of a design is improved. All the task execution processes described above, subject to the user request, may take place under the following constraints:

- The knowledge of recommended accesses in buildings.

- The knowledge of prohibited accesses in buildings.
- The maximum branching degree defined for each location.

In an attempt to improve their goal, non of the task executers is allowed to undo the result of the other task executers, and in cases when an attempt is blocked because of the constraints imposed on the design, then, new attempts are made by searching for next best possible action until a successful attempt is made. If an attempt by a task executer is completely blocked, then the current norm is put on a list of a *deferred-agenda*. The norms on the deferred agenda have always a higher priority over the norms on the agenda, and the task-manager in each iteration tries first to invoke a task executer according to the norm on the top of the deferred agenda.

If the *agenda* is empty, it means that attempts by all task-executers have been unsuccessful, then the control is passed to a fourth routine which is called *commitment task-executer*. The *commitment task-executer* has the function of breaking the logical deadlock for task-executers by making a forceful arbitrary assignment of an unprocessed location to the partial design. This condition arises only if there is not any logical relation between a partial design and the unprocessed locations.

The relation between pattern the generator module and these task-managers are depicted in the following figure.

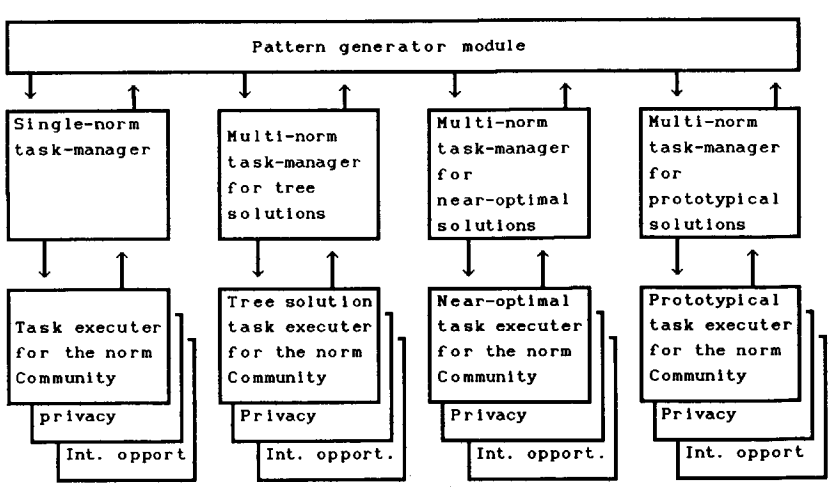


Figure 5.6: TOPGENE's task managers and task executers

5.7 Heuristics

> Heuristic: " A piece of knowledge capable of suggesting plausible action to follow or implausible ones to avoid." [Lenat82, p. 192]

I characterized architectural design at connectivity level as a NP-hard problem with a large state-space, and no efficient algorithm for its solution. In Chapter 2 I discussed the röle of heuristic programming in reducing the search efforts. In this chapter also I characterized TOPGENE as a heuristic program that uses strategic heuristic rules rather than a real-valued heuristic function [Georgeff83] in its procedures. Heuristic search based on the second technique has to use an analytical heuristic function that maps a search-state into a value reflecting the measure of promise of that state in reaching a goal state. The expression of problem specific strategies in terms of a set of processing rules was called strategic heuristics. Strategic heuristics, in comparison with real-valued heuristic, are [Georgeff83]:

- More akin to the human problem solving than the real valued heuristics.
- More transparent than real-valued heuristics.
- More adaptable and easier subject to change than analytical ones.

TOPGENE is a heuristic system that incorporates heuristic rules in the search-space of designs to reduce complexity. TOPGENE uses strategic heuristics rather than real valued heuristics, but it has a potential for the incorporation of real valued heuristics under certain conditions. The evaluator module can serve as a procedure for mapping partial designs (search states) into real values indicating their actual behavior. These values may be taken as a measure of the search-states in a means-ends analysis strategy. I will discuss this possibility in more detail in the last chapter. There are several reasons for choosing the strategic heuristic over a real valued heuristic:

- A heuristic approach based on a real-valued function would be computationally less inefficient than the current approach.
- There is always possible that a next best state found under the guidance of a analytical function is a locally optimum state that does not lead to an optimum or near optimum solution. The strong

inter-dependency of locations in a partial design with the unprocessed locations (future states in the search-state) brings about the possibility that an analytically chosen state might be a misleading path. Picking such a path will require backtracking in a later stage of the search. Backtracking is considered inefficient in most of the cases. Besides, the combinatorial inter-dependency of the location pairs in a design, support the claim that a backtracking strategy in our case runs into generating and testing of states of the search. This means that analytical heuristics with backtracking for our design problem is as inefficient as an enumeration approach.

- The goal of optimization in design is usually to improve performance towards some optimum point, and not necessarily to reach the optimum point itself. Design problems often demand satisfactory and realistic solutions rather than a mathematically optimized solution. A rigorous attempt in optimization of design in the computational sense, not only seems redundant, but also often runs into unrealistic designs.

TOPGENE uses heuristic rules provided by generalization of empirical causal relations between the structure and behavior of a building with respect to the social norms. Empirical causal relations are facts due to empirical observations of the buildings behavior. These causal relations are qualitative heuristic rules applicable in architectural domains.

Constraining relations about structure and behavior of a building may consist of isolated relations, and relations propagated in several numbers. A propagated sequence of causal relations defines a cascade of relations. A particular simple behavioral aspect of buildings may be represented by a single causal relation but a more complex behavior is often represented by a chain of related rules cascaded in a rule-set or within a procedure. A behavioral cascade in the system is the sequence of causal relations that defines a way for an initial condition (cause) to propagate through a topological network of building to cause (effect) operational events such as disturbance to a series of locations. Examples of these rules extracted from analysis of expert interviews are presented in appendix-A. Analysis of the expert interview shows that in general two types of causal rules relating the structure and behavior of buildings with respect to social norms exist:

- Rules relating the structure and behavior of buildings in the absence

of the operation of the buildings.

- Rules relating the structure and behavior of the building in the presence of operation of the building.

Most of the rules elicited from the expert are of the first type. For examples:

IF: - The arrangement of locations in serial (locations are intervening).

THEN: - Potential-exposure,
- High privacy cost,
- High Circulation-cost,
- High community utility.

IF: - Average Distance between locations is low.

THEN: - Divided Flow among paths,
- Potential for regulating Flow,
- Low circulation-cost,
- Low privacy cost.

The second type of causal relations emphasize the effect of operation on the structural-behavioral relationships in buildings. Here, connectivity patterns of buildings are considered as networks of flows, and the structural-behavioral relation in a building is determined by a mathematical analysis of its operation. Mathematically based causal relations may show themselves in quantitative or qualitative forms. For example:

IF: - The number of shortest paths between two locations L1 and L2 is p,
- Flow potential between L1 and L2 is n.

THEN: - Assume flow on each path = $\frac{n}{p}$.

The distinction between different types of causal relations enables one to understand the nature of the inferred relations, to organize design knowledge, and to model buildings' behavior. In addition, the causal relations based on mathematical principles may have broad applicability in other domains and different contexts within a domain.

The causal relations relating the structure, operation, and behavior of buildings are analyzed and cascaded in TOPGENE's procedures. The basic strategic rules used by TOPGENE in optimizing partial designs with respect to

the social norms are as follows. Protocols of several interviews with experts have been main inspiration for deriving these heuristic rules. These heuristic rules guarantee the improvement of partial designs with respect to the social norms on an iterative basis without any failure to do so. The expanded version of these strategic rules can be find in the design generation algorithms described later in this chapter.

Heuristic strategy for improving a partial design with respect to the *community* norm:

- Let *PD* be the partial design with locations (L_1, ..., L_n).
- Let *PHCLP* be the partially hierarchical clusters of location pairs.
- FOR (L_i L_j) := (POP *PHCLP*), until (*PHCLP* = NULL):
  IF - $L_i \in PD$ or $L_j \in PD$, say $L_i \in PD$.
  THEN - Find *ECC*, the list of locations in *PD* in ascending order of their eccentricity relative to L_i.
  - FOR L_k := (POP *ECC*), until (*ECC* = NULL) OR *success*:
    IF - (L_j L_k) $\notin$ *prohibited_accesses*,
    - The maximum branching degree of L_k is not violated.
    THEN - Improve *PD* by the access (L_j L_k),
    - Declare success,
    - Remove (L_i L_j) and (L_j L_k) from the *PHCLP*.
  - END-FOR.
- END-FOR.

Heuristic strategy for improving a partial design with respect to the norms *privacy / circulation cost*:

- Let *PD* be the partial design.
- Let *PHCLP* be the partially hierarchical clusters of location pairs.
- FOR (L_i L_j) := (POP *PHCLP*), until (*PHCLP* = NULL):
  IF - ($L_i \in PD$ and $L_j \in PD$),
  - {*PD*, (L_i L_j)} is planar,
  - (L_i L_j) $\notin$ *prohibited_accesses*,
  - The maximum branching degree of L_i and L_j is not violated.
  THEN - Improve *PD* by the access (L_i L_j),
  - Remove (L_i L_j) from the *PHCLP*.
  ELSE IF - $L_i \in PD$ or $L_j \in PD$, say $L_i \in PD$.
  THEN - LET *ECC* be the list of locations in *PD* is descending order of their eccentricity relative to L_i.
  - FOR L_k := (POP *ECC*), until *success*:
    IF - (L_j L_k) $\notin$ *prohibited_accesses*,
    - The maximum branching degree of L_k, set by the user, is not violated.
    THEN - Improve *PD* by the access (L_j L_k),
    - Declare success,
    - Remove (L_i L_j) and (L_j L_k) from the *PHCLP*.
  - END-FOR.
- END-FOR.

Heuristic strategy for improving a partial design with respect to the *intervening opportunity* (IO) norm:

- Let *PD* be the partial design.
- Let *OIOT* be the list of user defined IO triplets of locations.
- Order *OIOT* in terms of flow generation potentials between their bracketing locations.
- FOR (L_i L_j L_k) $\in$ (POP *OIOT*), until (*OIOT* = NULL):
  IF - $L_i \in PD$ & $L_j \in PD$ & $L_k \in PD$.
  THEN - Remove (L_i L_j L_k) from *OIOT*.
  ELSE IF - (L_i or L_k) $\notin PD$, say $L_i \notin PD$.
  THEN - Let *ECC* hold locations in *PD*, in ascending degree of their eccentricity relative to L_i, that are not on the path containing L_j and L_k,
  - PUSH L_j to *ECC*,
  - FOR L_m := (POP *ECC*), until success:
  IF - (L_i L_m) $\notin$ *prohibited_accesses*,
  - The maximum branching degree of L_m is not violated.
  THEN - Improve *PD* by the access (L_i L_m),
  - Declare success,
  - Remove (L_i L_j L_k) from *OIOT*.
  - END-FOR.
  ELSE IF - only $L_j \notin PD$.
  THEN IF - (L_i L_j) $\notin$ *prohibited_accesses*,
  - (L_j L_k) $\notin$ *prohibited_accesses*,
  - {*PD*, (L_i L_j), (L_j L_k)} is planar,
  - The maximum branching degrees of L_i and L_j are not violated.
  THEN - *PD* := {*PD*, (L_i L_j), (L_j L_k)}.
  ELSE IF - ((L_i & L_j) or (L_j & L_k)) $\notin PD$, say (L_i & L_j) $\notin PD$.
  THEN - Let *ECC* be list of locations in *PD*, in descending degree of their eccentricity relative to L_k,
  - PUSH L_k to *ECC*,
  - FOR L_m := (POP *ECC*), until (*ECC* = NULL) OR *success*:
  IF - (L_j L_m) $\notin$ *prohibited_accesses*,
  - The maximum branching degree of L_m is not violated.
  THEN - Improve *PD* by the accesses {(L_j L_m), (L_i L_j)},
  - Declare success,
  - Remove (L_i L_j L_k) from *OIOT*.
  - END-FOR.
  ELSE IF - ($L_i \notin PD$) & ($L_k \notin PD$).
  THEN - Let *paths* be List of non_circular paths in *PD* originating from L_j.
- FOR *path* := (POP *paths*), until success:
  - FOR L_m := (POP *path*), until (*ECC* = NULL) OR *success*:
  IF - (L_i L_m) $\notin$ *prohibited_accesses*.
  - The maximum branching degree of L_m is not violated.
  THEN - Improve *PD* by the accesses (L_i L_m),
  - Declare success.
  - END-FOR.
  - Remove *path* from *paths*.
  - END-FOR
  - FOR *path* := (POP *paths*), until success:
  - FOR L_m := (POP *path*), until (*ECC* = NULL) OR *success*:

IF - (Lk Lm) ∉ prohibited access,
- The maximum branching degree of Lm is not violated.
THEN - Improve *PD* by the accesses (Lk Lm),
- Declare success,
- Remove (Li Lj Lk) from *OIOT*.
- END-FOR.
- END-FOR.

5.8 Conflict resolution

The social norms defined in this thesis have some interesting characteristics with respect to an architectural design. Pair-wise comparisons of these norms show that:

- The norms community and privacy are two conflicting norms. Any attempt in an elevation and improvement of a design towards one of them may annihilate improvements already made towards the other one.
- The only two norms in mutual harmony are the norms privacy and circulation-cost. Improvement of a design with respect to any of these two norms has always a positive effect on the other.
- The community norm has the same relation with the privacy norm as with the circulation-cost norm.
- The norm intervening opportunity has a better relation with the other norms than the community and privacy relation. This norm might be in agreement with one of the norms community or privacy, but not with both of them, at the same time.

Theoretical work on conflict resolution in design, identifies two types of conflict situations: *competitive conflict situations* and *cooperative conflict situations* [Klein89]. The first situation arises when, each beneficiary in the conflict tries to fulfill his own ambitions, without being concerned about the others, or being interested in achieving a globally optimal goal in cooperation with the other parties. In the cooperative situations, on the other hand, the parties try to achieve a common goal by close cooperation with each other. Cooperation means *sacrifices, compromise,* and *abandonment of less important goals*, in hope of reaching a globally optimal goal which benefits every party. Computational models of conflict resolution are categorized into 5 major groups [Klein89]:

Development time conflict resolution:
This strategy, mostly used by knowledge based systems, requires resolution of all conflicts at the knowledge-base development time. development time conflict resolution demands enumeration of all conflicts and their solutions; so, it has disadvantage of putting all labor work on the shoulders of programmers. A middle road approach for this type of conflict resolution strategy is the run-time conflict resolution. Here, conflicts are resolved automatically during the run-time by incorporation of appropriate procedures into the process.

Backtracking-based failure handling conflict resolution:
This type of conflict resolution uses backtracking techniques in cases of conflicts to backtrack to earlier decision points to correct the wrong decision that caused the conflict. This strategy suffers from the inefficiency associated with the backtracking techniques.

Numerically-weighted constraint relaxation:
This method of conflict handling, mostly used in scheduling type problems, rely on weights assigned on requirements of the final solution. Such systems, in attempts to fulfill maximally all requirements, use constraint relaxation techniques to resolve the conflicts. This approach suffers from three main disadvantages:

- Assignment of numerical weights to constraints is unnatural to human experts,
- Different experts may use different weights to a requirement, and
- Weighted requirements lacks justifying explanation by human experts.

Specific conflict resolution advise:
Collections of specific domain-independent conflict-resolution decision rules may be incorporated in a system to resolve conflicts. This approach has the advantage of direct application of identified conflict resolution knowledge to resolve conflicts, but suffers from the limitation inheritance in domain-independent conflict resolution expertise. A wiser approach is the use of a combination of domain-independent and domain-dependent conflict resolution expertise to resolve the conflicts.

General conflict resolution expertise:

General conflict resolution expertise may be captured and stored implicitly in a system to be used during the processes. This approach gives the same level of importance to the conflict resolution knowledge as the domain knowledge.

TOPGENE's approach towards conflict resolution may be characterized as both cooperative and competitive conflict resolutions. The overall strategy used by TOPGENE in generating a design is based on a hill climbing (i.e., iterative improvement) approach. Conflicts have often to be detected prior to the resolution. However, in the case of our design problem the presence of conflicting norms in a design problem conveys the existence of consistent conflicting situation during a design. TOPGENE uses a competitive approach to deal with the conflicts by giving a chance to just one competitor, and relaxing the others iteratively. TOPGENE improves a partial design at each iteration only by a location or at most two locations This strategy gives a chance to each norm to improve partial designs with respect to their objective, at micro level. By this, the overall approach seems as if there is a sense of cooperation between all the conflicting norms, while at the macro level the norms seem to compete with each other. The competition is, first, democratic and each norm gets its own chance to fulfill its goal; and second, no norm is allowed to undo a result of the others. The main reasons for choosing this approach may be listed as follows:

- The development-time conflict resolution, the specific conflict resolution support, and the general conflict resolution expertise approaches all require domain knowledge of conflict handling. The incommeasurability of the norms implies that there are no logical relations between the norms, unless strong assumptions are made. For example, one can presume a linear relation $n1 = k*(n2)$ between an unit of a norm n_1, and a unit of a norm n_2. Upon such an assumption, the problem of conflict resolution may be handled by constraint satisfaction techniques. The main problem associated with this approach, as was mentioned earlier, is that association of numerical weights to norms is un-natural to experts. Consulting with the domain expert, did not reveal any type of domain-dependent conflict resolution strategy relating to these norms. In fact, a way of solving our design problem, as proposed in architectural literature

[Berwick71] [Tzonis87], is by generating sub-optimal designs with respect to each norm and singling out one that seems best according to a multi-criteria evaluation method such as ELECTRE [Tzonis75].

- Disregarding the incommeasurability of norms, a backtrack error handling method, or a numerically-weighted constraint relaxation method also may not be used here, for several reasons. Both of these approaches, in our design case, require either an analytical function for conversion of states of the search into numerical indicators that would show which path is a best path in the search space; or, require constraints on the minimum and maximum performances of the design with respect to the norms. Because of the heavy interactions between all states of the search as a result of the flow between locations of a design, a brute force approach can only find best paths during a search, or can find the performance constraints of a design. This approach is practically impossible. Besides, the first method, presumes the occurrence and detection of conflicts during the processes, and not prior to them. Here, at least in the case of the community and privacy norms, conflict exists at all situations. Finally, theoretically speaking, resolving a conflict between two incommeasurable norms is only possible by sacrifices and compromises from the conflicting sides.
- Architectural design is not only concerned with the social norms. Early experiments with TOPGENE showed that most of the designs generated by TOPGENE with respect to social points of view were unrealistic. For example, a design from point of view of community tends towards a linear-tree type design. Such a solution is in general not a practical solution. Similarly, a design with respect to the point of view of privacy / circulation-cost tends towards a compact pattern with high branching degree. Such solutions are also unrealistic in most cases. Integration of a complicated conflict resolution strategy into the search process, without the integration of knowledge of the architectural design, does not improve the realism of a design in any case.
- The goal of a design optimization in human problem solving is usually to improve the performance of a design, and not necessarily reach the global optima. Emphasis on global optima is not always natural. When

an architect talks about optimization of a building with respect to a view point, he talks about improvement of the building with respect to that point of view. Such a goal in design is pursued under the constraints imposed by several other view points, and with regard to the limitation of resources. As a result, in architecture and other human decision making processes, convergence to global optima is not an issue at all, but improvement towards and attaining a "satisficing" [Simon69] level of performance is of concern.

Considering above arguments the approach taken in implementing the TOPGENE is a plausible approach. Besides, the incorporation of some of the existing conflict resolution strategies discussed above into the TOPGENE also are possibilities that may be considered in implementing systems similar and akin to TOPGENE, or a new version of TOPGENE integrated with knowledge bases of architectural design.

5.9 The agenda of norms

TOPGENE uses an agenda mechanism to automate resolution of conflicts between the social norms. An agenda (sponsor) mechanism is a built-in feature of expert system tools such as Goldworks [87], which enables users to have control of the allocation of resources of the inference engine to different tasks. For example, in Goldworks, the rules pertaining to different subtasks may be clustered under different sponsors. Additional rules may be used to control the sponsors by enabling or disabling them. Each sponsor has an agenda associated with it that can contain rules or rule sets of different characteristics, such as forward rules, backward rules, or bidirectional rules. Sponsors may be ordered in a hierarchical manner, with quanta assigned to them. A limit is used to control the maximum number of rules that may be fired in an agenda at each iteration.

TOPGENE's agenda is a simplified simulation of the sponsor and agenda mechanism. The agenda of TOPGENE, which is a part of any multi-norm task-manager in TOPGENE, contains the norms in user defined ordered. TOPGENE uses this agenda of norms to control the invocation of task-executer procedures corresponding to the norms. Each task-executer is built based on the heuristic rules for improving a partial design with respect to a particular

norm. A task-executer receives a message from a task-manager and attempts to improve a partial design with respect to a norm. The norms on the agenda are processed on a first-in first-out (FIFO) basis. All norms on the top of the agenda are (after the invocation of corresponding task-executers) shifted to the bottom of the agenda, unless a task executer signals a failure. In such cases, norms are removed from the top of the agenda, and moved to a *deferred-agenda* of norms. Norms on a deferred agenda have always priority over the norms on the agenda, so, in any iteration, a task-manager always first checks for a norm on top of the deferred agenda for invocation of a task-executer. The presence of all norms on deferred-agenda signals failure of all task-executers. In this situation, the task-manager sends a message to a special task executer, whose job is to make a near-logical action for design commitment. This situation arises, when there is no logical relation between a partial design and remaining unprocessed locations. The following figure shows TOPGENE agenda control mechanism.

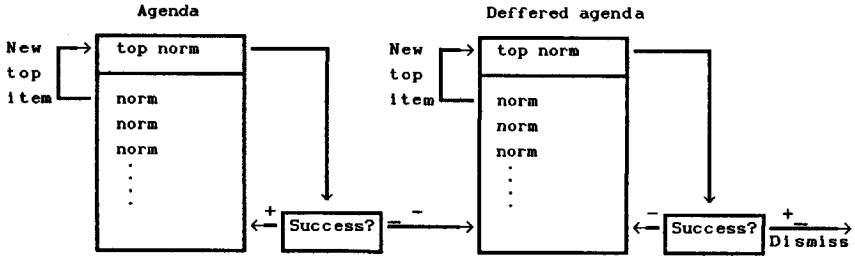


Figure 5.7: The TOPGENE agenda control mechanism

5.10 Domain knowledge as a constraint: teaching the TOPGENE

Another development in TOPGENE stemming from deficiency in generated designs was the integration of experimental knowledge bases with the system. TOPGENE presently has two knowledge bases consisting of recommended and prohibited accesses in buildings. These knowledge bases are improved via rote learning, when the system is directly taught to accept and memorize recommended and prohibited accesses in buildings. The use of the knowledge bases is optional, and the user is allowed to turn them on or off. If the knowledge base is turned on, recommended and prohibited accesses related to the design problem under process are presented to the user, and the user can decide on

the manual refinement of these accesses. The idea of integration of the knowledge base with the TOPGENE arose from the unrealistic designs generated by early versions of TOPGENE. The early version of the system, as a pure analytical system was careless about the nature of activities and their interconnections with other locations in a design.

The system, in its infancy, generated designs based on mathematical modeling of the architectural process regardless of other factors. Experiments showed that realistic design often do not emerge from pure mathematical modeling of a design process alone. The integration of knowledge of prohibited and recommended accesses in buildings improved the system considerably. A greater improvement is believed to be achievable by narrowing down the problem into a specific building type, or decomposing it into subsystems of designs for different building types. In such a case the integration of knowledge bases of different building types, or even integration of precedents of architectural designs with the system can broaden the practicality of generated solutions by TOPGENE. I will discuss this idea in the last chapter.

5.11 Planarity as a constraint

Architectural designs at topological level are only valid if they are realizable at geometric level. A necessary condition for realization of connectivity pattern of a building at geometric level is its planarity. A graph is planar if it cab be embedded in the plane without crossings. To ensure this condition TOPGENE had to resort to graph algorithms for testing the planarity of near-optimal type partial designs, that potentially may be non-planar. Other design types such as tree and prototypical designs are always planar, and do not require planarity testing.

Planarity testing and embedding of graphs have many applications. Several algorithms have been proposed for these purposes. There also are algorithms for planarization of non-planar graphs [Fisher66] [Ozawa81] [Jayakumar89]. The most efficient planarity testing algorithms are the Hopcroft and Tarjan algorithm [Hopcroft74], and the Booth and Lueker [Booth76] algorithm. Both algorithms have linear time complexity, but the later one, which is an improvement on Lempel and Even [Lempel67] is considered simpler than the first one [Nishizeki88].

TOPGENE tests planarity of a near-optimal type partial design *PD*, when *PD* is improved by adding a link L between two of its already existing nodes. In such cases TOPGENE passes {*PD* + L} to its planarity testing procedure, for a TRUE of False answer.

A graph G is planar if and only if all 2-connected components of G are planar. A graph is 2-connected if it is connected and does not have a cut-vertex (i.e., a vertex whose deletion results in disconnection of G). Furthermore a graph G with n (≥3) nodes and m edges is non-planar if $m \leq 3n-6$ [Nishizeki88]. So, first this edge-node relation is applied as a prior non-planarity test, and if it fails the testing algorithm is applied.

The Booth algorithm is used for testing and ensuring the planarity of partial designs . This algorithm employs a so called st-numbering technique and a data- structure called PQ-tree. The st-numbering and PQ-tree data structure have important rôle in Booth's planarity testing. A numbering of n nodes of a graph G by 1 to n is called a st-numbering if two vertices v_1 and v_n (i.e., vertices numbered "1" and "n") are necessarily adjacent, and each vertex v_j is adjacent to two vertices v_i and v_k, such that $i<j<k$. v_1 and v_n denoted by s and t are called source and sink of G, respectively. Every 2-connected graph has a st-numbering that can be fond in linear time [Nishizeki88]. A PQ-tree consists of "P-nodes", "Q-nodes", and "leaves". The key idea in a vertex-addition algorithm is to reduce the planarity testing of a graph to the problem of asking the permutation and reversions to make special sub-set of vertices to occupy consecutive positions. For a detailed discussion see [Booth76] or [Nishizeki88].

5.12 Branchiness as a constraint

Branchiness is another optional constraint on locations of a design in TOPGENE. By this option, an user may decide the number of links that a location might have with other locations. The idea of imposing the branchiness constraint on locations arose from experimenting with early versions of TOPGENE, whereby unlimited branchiness resulted in unrealistic designs. Decision on the number of links on locations in TOPGENE expands from near-optimal and tree types design to prototypical designs such as a star type design. In the first two types of designs the user is allowed to decide the branching degree of each locations irrespective of the other locations,

while in the second type, the branching degree of centers of stars may be determined by an user.

5.13 Designs with respect to a single norm

Decomposition of complex problems, such as designs, into several sub-problems, solving each sub-problems irrespective of the other, and synthesizing the sub-solution to arrive at a total solution, is a common approach by human problem solvers. TOPGENE, does not follow this line of approach in tackling the design problems, but generates designs with respect to isolated social norms, or a combination of them. This approach is rational, since:

- In practice some users may look for sub-optimal designs optimized with respect to single norms.
- The evaluator module uses sub-optimal solutions generated by the generatormodule as yardstick for evaluating existing designs.
- A practical approach in generating a design with respect to multiple points of view has been to generate sub-optimal designs each with respect to an isolated view point, and electing one among them that satisfies best all points of view together, by using a multi-criteria decision making method [Tzonis87]. TOPGENE's capabilities in generating diverse solutions to the same problem, including designs with respect to a single norm, pose it as a choice for integration with an automated multi-criteria decision making method.

Two types of information are important for TOPGENE in choosing a course of action in generating a design. The number of the norms (single or multiple) and type of the design. The key to generation of a design in all cases is the partially hierarchical clusters of location pairs provided by the Q-analysis routine. These clusters carry implicitly the potential of interaction between location pairs in a design. This information is the main criterion for heuristic arrangement of locations with respect to the social norms. The nature of the clusters, thus, is always invariant, but the treatment of the clusters and underlaying heuristic rules change according to the norm and type of design under the consideration. For example some combination of design type and norm requires generating a design directly based on the clusters, while others demand generating a stream or streams of

locations with specific attribute, and passing the stream(s) to a pattern generator for producing a topological pattern. The later case is for all prototypical design. In both cases TOPGENE uses the same heuristic strategic rules discussed in section 5.6.

The complete picture of how TOPGENE generates a design with respect to a single norm is depicted in figures 5.8 and 5.9. Figure 5.8 depicts the task diagram for the process, and the later one shows the data flow for the same process.

5.14 Designs with respect to the community norm

The first norm to discuss its implementation is the community norm. Under this norm a designer tries to maximize the behavior of a building in terms formation of new groups in locations of design, and increase degrees of interactions between the actors moving within the building. The community utility of a building increases with increase in meeting of flows caused by movements of people.

Intuitively an architectural design with a relatively compact topological structure and high average branching degree has a higher potential for absorbing the interactions between people than another less-compact design B with a relatively lower average branching degree. The least branched graph for a given number of nodes is a linear-tree. Linear-tree has potential for creating maximum overlaps and interactions for a building. So, a *linear-tree* type design also is the *near-optimal* design type for this norm, provided that it has a right operation in terms of arrangement of activities on its locations. TOPGENE allows an user to request generation of other types of building patterns such as a single-loaded, double loaded, stars and grid patterns. To deal with these prototypical patterns, TOPGENE first generates a stream of locations with a high potential performance with respect a norm or combination of norms, and labels the locations of a requested design with the activities on this stream. So, the task-executer for the norm community is in fact a stream generator. The input to the community norm task-executer is the hierarchical clusters of location pairs, and the output is a stream of locations for the design problem, arranged in such a way that a high performance with respect to the norm community is expected. The community stream is, thus, a multi-purpose object that can be

fed into different pattern generators produce different type of designs. Examples of these design types patterns will be presented in chapter 6. Generation of designs with respect to three other norms: intervening opportunity, privacy, and circulation-cost have two distinct approaches. One for generation pattern for the near-optimal, and tree solutions, and another for prototypical solutions.

Figure 5.10 shows levels of operations that are carried out by TOPGENE in reaching a design with respect to the community norm. The first 3 levels are the Q-analysis levels depicted earlier, the fourth level is the stream generation level, and the last level corresponds to the pattern generation processes.

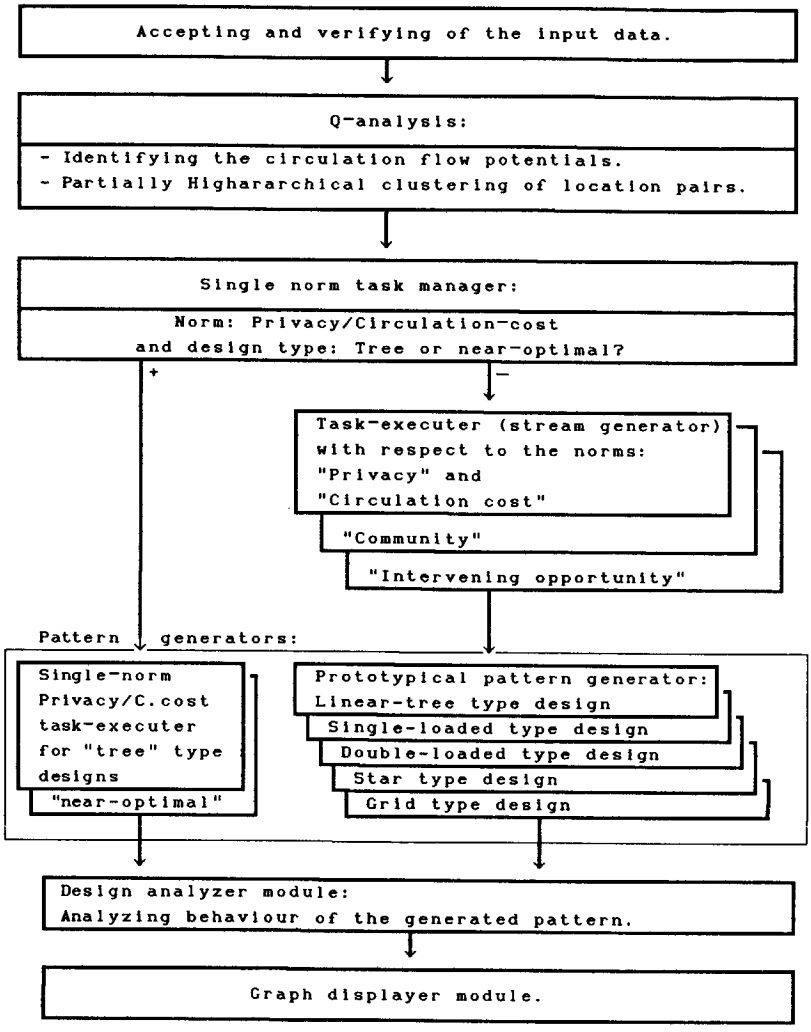


Figure 5.8: TOPGENE's task diagram for generating designs with respect to a single norm

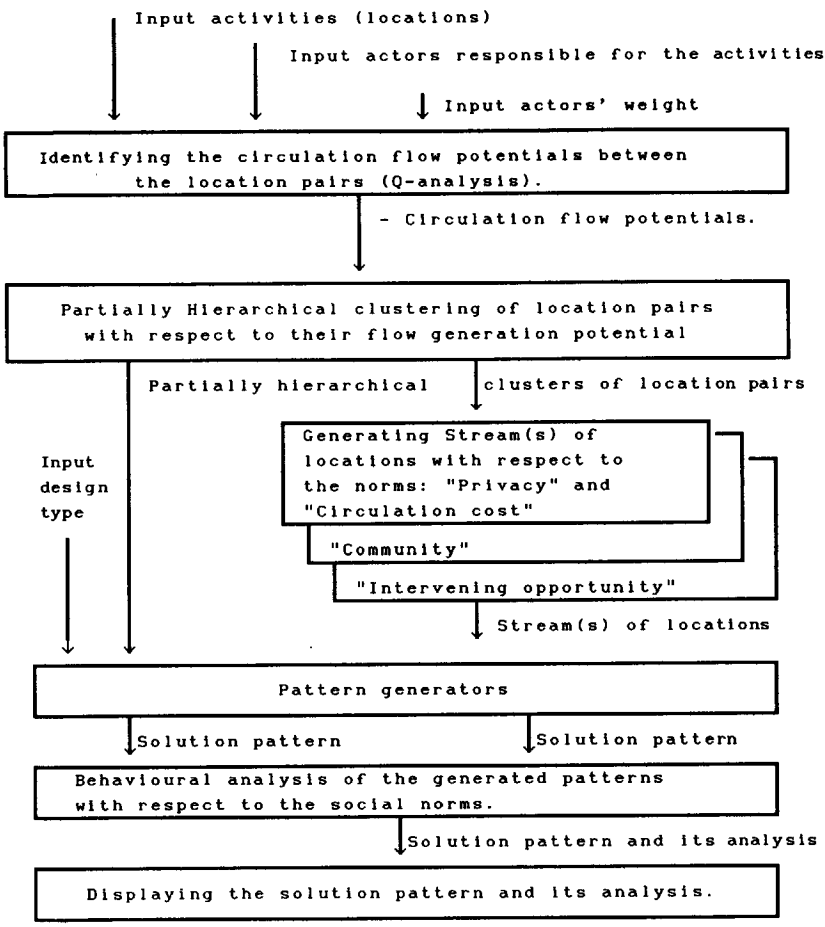


Figure 5.9: TOPGENE's data flow diagram for generating designs with respect to a single norm.

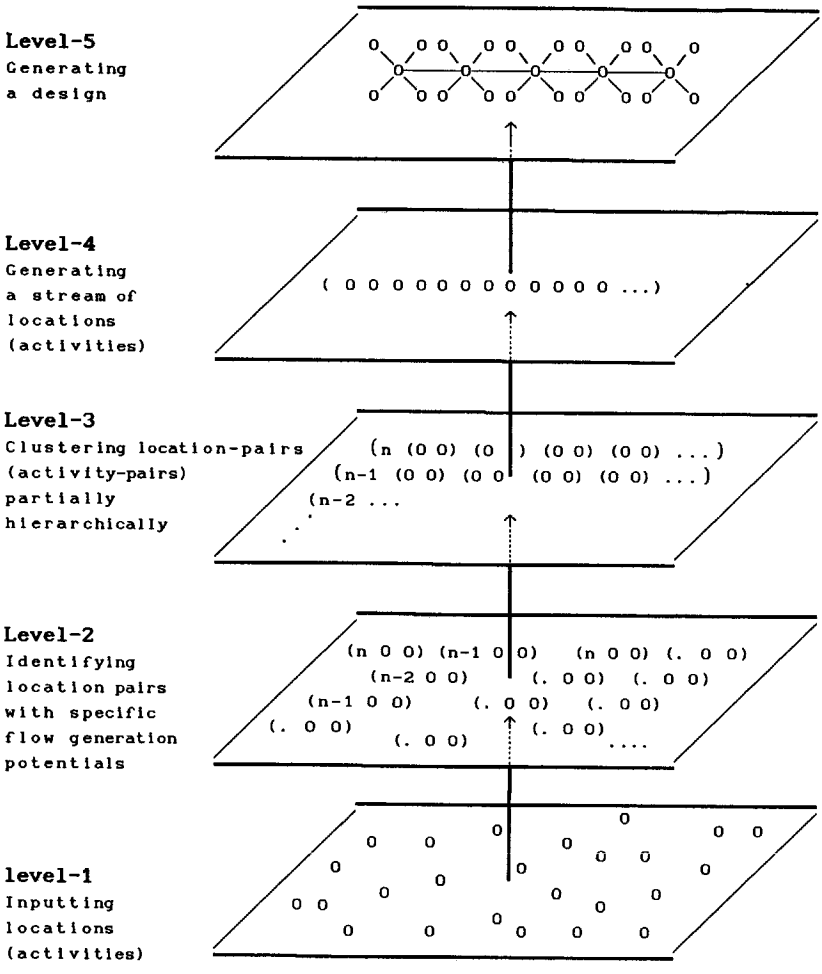


Figure 5.10: Levels of data-abstraction in generating a design pattern with respect to the community norm

5.14.1 Generating a stream of locations

Based on the hierarchical clusters of location-pairs generated by the Q-analysis procedure, and the heuristic strategy discussed in this chapter, the following algorithm generates a stream of locations (or activities) consisting of all locations in the design statement with respect to the norm community. The heuristic rule for generating such a stream puts the most

interactive activities as far as possible relative to each other. The algorithm pursues the stream generation using a hill climbing strategy, in a way that it tries to improve the community utility of the stream iteratively. This task-executer in fact generates the stream from the outer locations (i.e., most interactive locations) towards the inner locations. So technically speaking, the procedure, first, generates two sub-streams, and finally concatenates them to form the complete stream. A hand simulation of this process is presented at the end of this section.

There are situations during a design process in which no logical relation could be found between a new location to be processed and a partial design. In these cases there is a sense of uncertainty on where in the partial design to assign the new location. TOPGENE handles these situations by a look-ahead algorithm that tries to find a logical relation between the new location and the partial design by looking one step further into the state search. The look-ahead mechanism is exemplified in the following section.

The look-ahead mechanism:

The look-ahead algorithm for the community norm receives as input:

- A location pair *lp* to be processed,
- Clusters of unprocessed location-pairs *CLP*,
- The generated sub-streams (partial design) *PD*.

The look-ahead algorithm first tries to match a location in *lp* to a location in a location-pair $p \in CLP$, that its other location is already processed (i.e., it is in the partial design). In the next step, the algorithm tries to assign the locations *lp to PD* in such a way that its matched location is assigned to a part of the *PD* that has no identical location with a location in p. If, we have the following partially hierarchical clusters of locations:

(A3 A4)
(A1 A3)
(A5 A6) (A5 A1)
(A1 A7)

Then, after step 2 the stream generator algorithm faces a dead-lock situa-

tion in which it cannot decide on how to process the location-pair (A5 A6). Here, the algorithm calls the look-ahead procedure for processing (A5 A6) as is step-3 illustrated below.

Step	C-SPACE1	C-SPACE2
1	(A3)	(A1 A4)
2	(A3)	(A1 A4)
3	(A3 A5)	(A6 A1 A4)

The followings are algorithms for both string generation and look-ahead processes.

```
Procedure SINGLE-NORM-COMMUNITY-TASK-EXECUTER (sphclp)
1- Let phclp be list of the partially hierarchical clusters of location
   pairs generated by the Q-analysis algorithm.
2- Let sphclp be list of location pairs resulted from scrapping of the
   phclp.
3- LET stream1 and stream2 be two sub-streams.
4- Stream1 := NIL.
5- Stream1 := NIL.
6- Stream := Stream1 + Stream2.
7- FOR pair := (POP sphclp) until sphclp is NULL
    - LET head := (FIRST  pair).
    - LET tail := (SECOND pair)
    - COND      - (head ∈ stream) & (tail ∈ stream).
      THEN      - Do nothing.
    - COND      - (head ∈ stream1).
      THEN      - PUSH tail to stream2.
    - COND      - (head ∈ stream2).
      THEN      - stream1 := stream1 + {tail}.
    - COND      - (tail ∈ stream1).
      THEN      - PUSH head to stream2.
    - COND      - (tail ∈ stream2).
      THEN      - stream1 := stream1 + {head}.
    - OTHERWISE
                - Call LOOK-AHEAD (pair sphclp stream1 stream2) success
                - IF   - look-ahead failed.
                - THEN - stream1 := (APPEND stream1 + {head}),
                       - PUSH tail to stream2.
8- END-FOR.
9- RETURN stream := stream1 + stream2.
10- END-PROCEDURE.


Procedure LOOK-AHEAD (pair pairs stream1 stream2)
1- LET stream := stream1 + stream2.
2- LET head_pair := (FIRST  pair).
3- LET tail_pair := (SECOND pair).
4- FOR p:= (POP sphclp).
    - LET head_p := (FIRST pair).
    - LET tail_p := (SECOND pair).
```

```
    IF    - (head_p = head_pair & head_p ∉ stream & tail_p ∈ stream), OR
          - (head_p ∈ head_pair & tail_p ∉ stream & head_p ∈ stream), OR
          - (tail_p = head_pair & head_p ∉ stream & tail_p ∈ stream), OR
          - (tail_p ∈ head_pair & tail_p ∉ stream & head_p ∈ stream).
    THEN:
    COND  - (head_pair = head_p & head_p ∈ stream2).
          - stream1 := stream1 + {head_pair},
          - PUSH tail_pair to stream2,
          - success := TRUE.
    COND  - (head_pair ∈ head_p & head_p ∈ stream1).
    THEN  - stream1 := stream1 + {tail_pair},
          - PUSH head_pair to stream2,
          - success := TRUE.
    COND  - (tail_pair = tail_p & tail_p ∈ stream1).
    THEN  - stream1 := stream1 + {head_pair},
          - PUSH tail_pair to stream2,
          - success := TRUE.
    COND  - (tail_pair ∈ tail_p & tail_p ∈ stream2).
    THEN  - stream1 := stream1 + {tail_pair},
          - PUSH head_pair to stream2,
          - success := TRUE.
    COND  - (tail_pair = head_p & head_p ∈ stream1).
    THEN  - stream1 := stream1 + {head_pair},
          - PUSH tail_pair to stream2,
          - success := TRUE.
    COND  - (tail_pair ∈ head_p & head_p ∈ stream2).
    THEN  - stream1 := stream1 + {tail_pair},
          - PUSH head_pair to stream2,
          - success := TRUE.
    COND  - (tail_pair = tail_p & tail_p ∈ stream1).
    THEN  - stream1 := stream1 + {head_pair},
          - PUSH tail_pair to stream2,
          - success := TRUE.
    COND  - (tail_pair ∈ tail_p & tail_p ∈ stream2).
    THEN  - stream1 := stream1 + {tail_pair},
          - PUSH head_pair to stream2,
          - success := TRUE.
  IF    - success.
  THEN - Break the loop.
5- END-FOR.
6- IF    - success.
   THEN  - Return stream1 and stream2.
   ELSE  - Return NIL.
7- END-LOOK-AHEAD.
```

Figures 5.11 to 5.13 depict data for a design problem, partially hierarchical clusters of location pairs generated by the Q-analysis procedure based on the data, and steps showing the formation of a stream of locations with respect to the community norm. The actual flow degrees are omitted in this example, but are used by TOPGENE.

Location (Activity)	Groups responsible for an specific activity.
A1	G1,G2,G6
A2	G2,G4
A3	G3,G1,G4
A4	G4,G1
A5	G5,G7,G4,G9
A6	G5,G7,G9,G10
A7	G6,G8,G9
A8	G6,G7,G8,G9
A9	G7,G10
A10	G8,G9,G10

Figure 5.11: A set of activities and actors responsible for the activities

Potential flow	Partially hierarchical clusters
High ↑	{ (A5 A6) (A7 A8) }
	{ (A3 A4)(A5 A8)(A6 A8)(A6 A9)(A10 A6)(A10 A7)(A10 A8) }
	{ (A1 A2)(A1 A3)(A1 A4)(A1 A7)(A1 A8)(A2 A3)(A2 A4)
	(A2 A5)(A3 A5)(A4 A5)(A5 A7)(A5 A9)(A10 A5)(A6 A7)
Low ↓	(A8 A9)(A10 A9) }

Figure 5.12: Partially hierarchical clusters of activity pairs

Iteration	Stream-1	Stream-2
1	(A5)	(A6)
2	(A5 A7)	(A8 A6)
3	(A5 A7 A4)	(A3 A8 A6)
4	(A5 A7 A4 A9)	(A3 A8 A6)
5	(A5 A7 A4 A9 A10)	(A3 A8 A6)
6	(A5 A7 A4 A9 A10 A1)	(A2 A3 A8 A6)

The final stream: (A5 A7 A4 A9 A10 A1 * A2 A3 A8 A6)

Figure 5.13: Formation of a community stream from partially hierarchical clusters of locations

The "*" mark, which divides two sub-streams marks the first location pairs processed, and is useful in generating a circular pattern. A direct arrangement of a generated stream of location on a circular pattern brings the most interactive locations together (i.e., A5 and A6 in figure 5.12). Something which must be avoided for the community norm. Here, reversing one of the sub-streams (from the marked point) reduces the negative effect.

5.15 Designs with respect to privacy / circulation-cost norm(s)

Privacy and circulation costs are two non-conflicting norms that are in harmony with each other, but are in conflict with the norm *community*.

Circulation-cost is the norm under which the travelling costs of people and objects moving between location-pairs in a building are minimized. The total circulation cost in a building is proportional to the number and distances of journeys that groups make. Here, the circulation potential between the location-pairs are taken as the number of journeys. This information is provided by applying the Q-analysis method on the input data. The distances are taken as the topological distances (i.e., number of links) between locations.

The norms privacy and circulation-cost both require the same structural and operational characteristic from a design. The costs associated with these two norms, according to our definition, is reduced with decrease in interactions between the actors in a building and reduction in distances between the interactive locations. Heuristically, the following structure-selection and operation-management fulfill these conditions:

- Selecting a relatively compact and highly branched design that provides a large number of access possibilities between different locations.
- Allocating the most interactive activity pairs as close as possible (e.g. adjacent), and least interactive activity pairs in the periphery of the design.

The prerequisite for stepping towards such a goal is the identification of interaction between the location pairs, and hierarchically arrangement of them into clusters with identical interaction potentials. This, as in the case of other norm is carried out by the Q-analysis algorithm.

Generating all types of designs with respect to the community norm required a single strategy. Here, the case is different, and different types of design may require different approaches. The reason is that a near-optimal solution with respect to these norms, naturally, is leaned towards a compact planar graph or a compact tree with high degree of branchiness. This is quite different design than, for example, a prototypical design type, such as a linear-tree design. On the other hand, all prototypical designs share some similarities within themselves. There is a sense of order in the

arrangement of locations in these patterns. For example, a linear-tree may be viewed as a stretched double-loaded, or a single-loaded pattern. Something that cannot be found in irregular designs in the forms of general graphs or trees. The regularities between the prototypical patterns, gave incentive to find a single approach in generating these design types. TOPGENE first generates a stream of locations with respect to these norms. This stream is then used for labeling a design in such a way that a minimum distortion is caused to the order of the locations in the stream. The order of locations in a generated stream with respect to a set of norm(s), provides the behavior of a design with respect to the norm(s).

The following sub-sections provide detail descriptions, and algorithms for generating different designs with respect to these norms:

5.15.1 Near-optimal design (Detecting a maximum flow planar graph)

A near-optimal solution for the norms privacy / circulation-cost was characterized as a compact solution. Such a pattern necessarily must planar in order to be a feasible solution in terms of its realization at geometric level. Now, for minimizing interactions between the location pairs, intuitively one has to be able to generate a compact planar graph labeled with the locations (or activities) in such a way that the most interactive activities are adjacent. In other words, and in graph language, assuming that we have a completely connected graph G with certain flow between each location pairs, the objective would be to detect *the maximum flow planar graph* of G. There are two possibilities for achieving this goal. A top-down approach, and a bottom-up approach.

In the first case, one may start with a completely connected graph G with number of nodes equal to the number of activities in the design statement, and planarize this graph by pruning always the least interactive links. This approach may be characterized as a top-down approach that starts from a non-planar design and tries to reduce it to an acceptable planar one.

Another possibility is to start from the bottom up. In this case, one starts from a partial design consisting of a single link, and continues towards a complete design by adding a link at a time. The process necessarily would give priority to the most interactive activities, and work out towards perfection under the planarity constraint. This approach is a

heuristic approach to optimization of a design with respect to the privacy / circulation-cost norms.

The generated solutions by both algorithm would be in the neighborhood of an optimal solution, if not an optimal one. However, the solution may not necessarily be a realistic design. Such solutions obviously have the benefit of being used as yardstick for generating realistic designs by the architects. The bottom-up approach seems more suited for a system, such as TOPGENE, that tries to generate more realistic designs. To achieve this goal, TOPGENE is integrated with experimental knowledge of recommended and prohibited access between locations of a building. With such capability, the user is allowed to decide upon the use of this knowledge base during generating a solution pattern. TOPGENE, also allows the user to set a limit to the branching degree of locations in a design. This constraint also helps in generating realistic designs. The knowledge base and branching degrees have priority over the mathematical information and the heuristic strategy followed by TOPGENE. The procedure for generating near-optimal design with respect to the privacy / Circulation-cost is as follows.

```
Procedure SINGLE-NORM-PRIVACY-NEAR-OPTIMAL-TASK-EXECUTER (phclp)
1- Recall the recommended and prohibited accesses from the knowledge base.
2- Ask for branching degree.
3- LET processed_locs be list of the processed locations.
4- LET solution be the partial solution.
5- Initialize solution preferably by the recommended accesses.
6- flag := TRUE.
7- DO-UNTIL all locations are processed.
  7.1- IF    - phclp is NULL.
       THEN: - phclp := pos_links,
             - pos_links := NIL.
  7.2- IF:   - flag = NIL.
       THEN: - Call COMMITMENT_TASK_EXECUTER.
  7.3- temp_phclp := phclp.
  7.4- flag := NIL.
  7.5- FOR pair := (POP temp_phclp), until flag = TRUE.
    7.5.1- WHEN - All locations are processed.
           THEN - phclp := NIL,
                - Break the loop.
    7.5.2- COND (FIRST pair) ∈ solution & (SECOND pair) ∈ solution.
           THEN: IF - pair is not a prohibited link,
                    - Branching degree is not violated,
                    - Planarity is not violated.
                 THEN - Add the link consisting of locations in pair to
                        solution,
                    - Update branchiness of locations in pair,
                    - flag := TRUE,
```

```
                - REMOVE pair from phclp.
        COND    - (FIRST pair) ∈ solution.
        THEN    - LET neighbors be list of all locations in solution sorted
                  in ascending order of their eccentricity relative to
                  (FIRST pair).
                - FOR loc := (POP neighbors):
                    WHEN  - The link (loc (SECOND pair)) is not a prohibited
                            link.
                          - Branchiness degree is not violated.
                    THEN  - Add the link (loc (SECOND pair)) to solution,
                          - ADD (SECOND pair) to processed_locs list,
                          - Update branching degrees,
                          - REMOVE (LIST loc (SECOND pair)) from phclp,
                          - Break the loop.
              - END-FOR.
      COND - (SECOND pair) ∈ solution.
      THEN - LET neighbors be a list of locations in the partial
             solution sorted in ascending order of their
             eccentricity relative to (SECOND pair).
           - FOR loc := (POP neighbors).
               WHEN - The link (loc (FIRST pair)) is not a prohibited
                      link.
                    - Branchiness degree is not violated.
               THEN - Add the link (loc (FIRST pair)) to solution,
                    - ADD (FIRST pair) to the processed_locs,
                    - Update branching degrees,
                    - REMOVE (loc (FIRST pair)) from phclp,
                    - Break the loop.
         -END-FOR.
      OTHERWISE do nothing.
7.6- END-FOR.
7.7- IF   - Any unprocessed recommended link. THEN - Process it.
8-  END-DO-UNTIL.
9-  Return solution.
10- END-procedure.
```

5.15.2 Tree design (detecting a maximum flow spanning tree)

A tree type design with respect to the norms privacy / circulation-cost is the same as a near-optimal type design with the exception that it must not contain any cycles. If we presume all location of a design as connected and each connection is labeled with the flow potential between the location-pairs, then a tree type design with respect to the privacy / circulation-cost norm is a maximum flow spanning tree of our completely connected design. To detect such a tree, TOPGENE, however, as for the semi-optimal type designs with respect to these norms, takes a bottom-up approach by building such a tree from a pair of locations up to the complete design. The approach is basically the same as the semi-optimal type design, with the

exception that cycles are avoided in this case. So, the algorithm for the task-executer corresponding to these two norms is basically the same as algorithm described for the near-optimal designs with respect to these norms. The only difference is the step providing cycles in a design, which is unnecessary for this case.

5.15.3 Generating streams of locations for prototypical designs

The prototypical solutions for the norm privacy / circulation-cost, as for the community norm, is approached via the stream generation. A stream of locations (activities) with respect to these norms must provide minimum flow meetings in the corresponding building. Heuristically this condition is achieved if the most interactive activities are closer to each other than the less interactive ones. This strategy is reverse of the strategy used for the community norm. The criteria determining the order of location pairs to be processed are the interaction potential for the activity pairs. This order is determined by the hierarchical clustering of the activity pairs. Here also streams are generated in an iterative improvement manner, in such a way that in each iteration a location is added to the partial design.

The output from the privacy stream generator (i.e., task executer) is either a single stream of locations or multiple ones. The first case relates to a situation whereby all the locations in a problem have some kind of direct or indirect association with each other. Here, the privacy-stream generator yields only a single stream of locations related to a set of input data. The second case arises when all locations in a design do not have logical relations with each other. Here, the set of locations is logically broken into more than one disjoint sets of related locations. In such cases the stream generator would recognize such association gaps between the locations of a design and cluster them into separate streams.

Again, as for other norms, the final stream is fed to a pattern generator responsible for generating a range of patterns. Figure 5.14 shows the 4-th and 5-th levels of the prototypical design generation with respect to the privacy / circulation-cost norm(s). The first three levels are the Q-analysis levels and are identical with those depicted in figure 5.10.

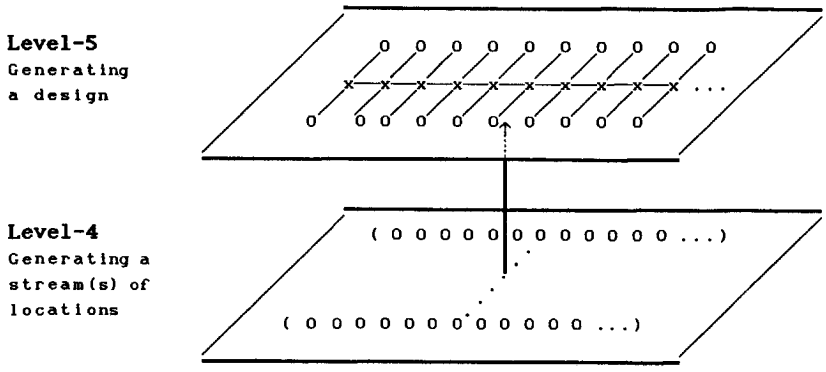


Figure 5.14: Process levels corresponding to a design generation with respect to the norm privacy or circulation-cost

Given a partially hierarchical clusters of location-pairs, the following algorithm generates a stream or streams of locations for prototypical designs with respect to the privacy / circulation-cost norm(s):

```
Procedure SINGLE-NORM-PRIVACY-PROTOTYPICAL-TASK-EXECUTER (phclp)
1- Let phclp be list of the partially hierarchical clusters of location
     pairs generated by the Q-analysis algorithm.
2- Let sphclp be list location paris resulted from scrapping of the phclp.
3- LET c_stream be the current stream.
4- LET streams be list of streams.
5- c_stream := (POP phclp).
6- LOOP:
  6.1- flag := NIL.
  6.2- FOR pair (POP sphclp):
       IF   - ((FIRST pair) ∈ c_stream & (LAST pair) ∉ pair).
       THEN - new_loc := (LAST pair).
       ELSE  IF:  - ((FIRST pair) ∉ c_stream & (LAST pair) ∈ pair).
             THEN - new_loc := (FIRST pair).
       COND  - One of the locations in pair is identical with one of the end
                 location in c_stream,
             - new_loc has the same degree of association with both of the
                 end location of c_stream.
       THEN  - Backtrack to find a processed location in the c_stream which
                 has relation with the processed location.
             - Add new_loc to end_side of the c_stream such that it is
                 closest to the processed location in c_stream,
             - Remove pair from sphclp,
             - flag := TRUE.
       COND  - A locations in pair is identical with a location of c_stream.
       THEN  - Add new_loc to the end of the c_stream closest to the location
                 identical with a location in pair.
```

```
            - Remove pair from sphclp,
            - flag := TRUE.
       OTHERWISE - Do nothing.
  6.3- END-FOR.
     WHEN - flag = NIL.
     THEN - Break the loop.
7- END_LOOP.
8- IF:     - sphclp = NULL.
   THEN:   - Return streams as the final solution,
           - Stop.
   ELSE    - Push c_stream to streams,
           - c_stream := (POP sphclp).
9- Go to step 5.
```

Applying above algorithm to the hierarchical clusters of activities presented in figure 5.12 yields the following stream for the norm privacy.

Step	C-STREAM
1	(A5 * A6)
2	(A8 A5 * A6)
3	(A7 A8 A5 * A6)
4	(A7 A8 A5 * A6 A9)
5	(A10 A7 A8 A5 * A6 A9)
6	(A1 A10 A7 A8 A5 * A6 A9)
7	(A2 A1 A10 A7 A8 A5 * A6 A9)
8	(A3 A2 A1 A10 A7 A8 A5 * A6 A9)
9	(A4 A3 A2 A1 A10 A7 A8 A5 * A6 A9)

Figure 5.15: Formation of privacy cluster from hierarchical clusters

Note that, in this example, all activities have fallen into the same stream. This implies that there is at least a chain of association between them. However, there might not be direct associations between all the location-pairs. If there was a break in this chain, then we would have several separated streams of locations each with a chain of association between their elements. In such a case the list of streams of activities would look like this: ((... * ...) ... (... * ...)).

5.16 Designs with respect to the "intervening opportunity" norm

This social norm, although different in nature from other norms, but has a common characteristic with the community norm. It tends towards a linear-tree pattern as a near-optimal solution. An optimal solution with the inter-

vening opportunity norm is a solution that should satisfy a set of in-betweenness requirements. Obviously, such requirements are best provided by a liner-tree type design.

TOPGENE, as in the case of the prototypical designs with respect to other norms, first generates a stream of locations that satisfies at least a sub-set of in-betweenness requirements in the design, then labels a design demanded by the user with the locations (or activities) in this stream. The exception is the tree type design. A tree type design is generated directly based on the in-betweenness requirements defined. This approach was taken to prove that a general tree is also a candidate solution for the intervening opportunity norm. Otherwise, in general a linear-tree might yield a better result than a general tree for the same set of data. TOPGENE uses heuristic strategy, described earlier, for generating the intervening opportunity (IO) stream. The IO task-executer generates the IO stream based on the IO-triplets input by the user to the system. The criterion for selecting an IO-triplet is its potential for contributing to the total intervening opportunity of design. This criterion is explicated from the degree of association between the bracketing locations in the triplets. The association degrees, here as in the case of other norms, are detected by the Q-analysis procedure.

The stream of locations is fed to a prototypical pattern generator for producing a final design. Obviously, some of the prototypical designs, such as double-loaded and single-loaded designs with auxiliary locations, are not fit to the intervening opportunity norm at all, since they cannot provide a large degree of in-betweenness requirements. This problem is inevitable, and cannot be overcame, unless the notion of nearness is considered in computing the behavior of buildings with respect to social norms. For some other design types, such as prototypical designs without auxiliary locations, specialized algorithms for each case may provide better results than generating designs with respect to the intervening opportunity norm based on a stream of locations. The reason is that labelling of a design with the items on a stream heavily effects the in-betweenness property of the stream, and consequently degrades the expected performance of the design with respect to this norm.

5.16.1 Generating a stream of locations

TOPGENE single-norm task-executer for the norm intervening opportunity is responsible for generating a stream of location with respect to this norm. The input to this task-executer is an arbitrary stream containing all locations for a design, and a list of intervening opportunity triplets (intervening opportunity requirements). The triplets are ranked with respect to the flow generation potentials between the bracketing locations. The flow generation potential is identified by the Q-analysis algorithm. This task-executer starts by the arbitrary stream, and slides the locations around as required by the intervening opportunity triplets and constraints their movement possibility so to preserve the in-betweenness conditions for the most important intervening locations. The approach is a heuristically guided hill-climbing approach. The intervening opportunity task-executer starts from the most valuable io-triplet (i.e., the io-triplet with most interactive first and last activity) and satisfies its required in-betweenness. In each of the following iterations this task-executer tries to satisfy the in-betweenness conditions demanded by other io-triplets in order of their priority such that previously satisfied conditions are not undone. This approach guarantees a fast solution in the neighborhood of an optimal solution without a need for a brute-force approach.

The core of the algorithm for generating a stream of locations with respect to the intervening opportunity is as follows:

- Let *OIOT* be the list of all user defined intervening opportunity triplets of locations, ranked according to the flow potential between their bracketing locations.
- Take an arbitrary stream of all locations in the problem.
- LET each location in the stream have a left-movement and a right-movement attribute.
- Assign the left-movement and the right-movement of each location to NULL.
- FOR (L_i L_j L_k) $\in$ (POP *OIOT*), until *OIOT* = NIL:
    - Move L_j in-between the first and the last location, if its left-movement and right-movement constraint allows to do so.
    - Update the movement constraints to include the current action.

-END-FOR

The expanded version of the algorithm for generating IO stream of locations is as follows:

```
Procedure SINGLE-NORM-IO-TASK-EXECUTER (IO_stream IO_triplets)
1- LET counter := 1.
2- FOR location := (POP IO_stream), until IO_stream is NULL:
  - Location's position := counter.
  - Increase counter by 1.
  - SET the "left" property of the location to NIL.
  - SET the "right" property of the location to NIL.
3- END-FOR.
4- FOR IO_triplet := (POP IO_triples), until IO_triplets is NULL:
  - LET lpos be the position of the (FIRST  IO_triplet) in the IO_stream.
  - LET mpos be the position of the (SECOND IO_triplet) in the IO_stream.
  - LET rpos be the position of the (THIRD  IO_triplet) in the IO_stream.
  - IF    - lpos > rpos.
  - THEN  - REVERSE IO_triplet.
          - SWAP rpos with lpos.
  - COND  - mpos > lpos & mpos < rpos.
    THEN  - Call (CONSTRAIN_MOVEMENT IO_triplet).
  - COND  - mpos < lpos.
    THEN  - CALL (FIRST_AFTER_MIDDLE IO_stream
                  (LIST (SECOND IO_triplet) (FIRST IO_triplet) (THIRD
                  IO_triplet) )).  OR
          - CALL (MIDDLE_BEFORE_FIRST IO_stream
                  (LIST (SECOND IO_triplet) (FIRST IO_triplet) (THIRD
                  IO_triplet) )).  OR
          - CALL (LAST_BEFORE_FIRST IO_stream (LIST (SECOND IO_triplet)
              (FIRST IO_triplet) (THIRD IO_triplet) )).
  - COND  - mpos > rpos.
    THEN  - (MIDDLE_AFTER_LAST IO_stream (LIST (FIRST IO_triplet)
                  (THIRD IO_triplet) (SECOND IO_triplet) )).
                    OR
          - (LAST_BEFORE_MIDDLE IO_stream
            (LIST (FIRST IO_triplet)
                  (THIRD IO_triplet) (SECOND IO_triplet) )).
                    OR
          - (FIRST_AFTER_LAST IO_stream
            (LIST (FIRST IO_triplet) (THIRD IO_triplet)
                  (SECOND IO_triplet) )).
5- END_FOR.
6- Generate the IO_stream according to the final position (movement const-
   raint) of each location.
7- RETURN IO_stream.
```

The function of each sub-procedure called by the IO task-executer are as described below. The detail of the algorithm is suppressed for the sake of brevity.

The procedure CONSTRAINT-MOVEMENT fixes the movement range of an inter-

vening opportunity location so that it is in-between the current bracketing locations, and all previously declared bracketing locations (if any).

The procedure FIRST-AFTER-MIDDLE checks the movement constraint of an locations in a triplet, and adjusts the range movement of them so that the first activity is positioned after the middle one.

The procedure MIDDLE-BEFORE-FIRST checks the movement constraint of the locations in a triplet, and adjusts the range movement of them so that the middle activity is positioned before the first one.

The procedure LAST-BEFORE-FIRST checks the movement constraint of the locations in a triplet, and adjusts the range movement of them so that the last activity is positioned before the first one.

The procedure MIDDLE-AFTER-LAST checks the movement constraint of the locations in a triplet, and adjusts the range movement of them so that the middle activity is positioned after the last one.

The procedure LAST-BEFORE-MIDDLE checks the movement constraint of the locations in a triplet, and adjusts the range movement of them so that the last activity is positioned after the middle one.

The procedure FIRST-AFTER-LAST checks the movement constraint of the locations in a triplet, and adjusts the range movement of them so that the first activity is positioned after the last one.

5.17 Designs with respect to multiple norms

Generation designs with respect to a combination of social norms, as was depicted in figure 5.6, are carried out by three different task-managers, each having three task-executers to its disposal. A task-executer improves a partial design with respect to a specific norm. The near-optimal and the tree type designs with respect to multiple norms, as in the cases of single norms, generate connectivity patterns of buildings with no particular regularity. The patterns of access for these types of designs are defined by the nature of data reflected in the hierarchical cluster of location pairs. The task-managers for these two types of designs proceed design generation by following the general strategy and heuristic rules, described earlier in this chapter, and calling their corresponding task-executers to generate designs using a hill-climbing strategy iteratively and in an order defined by the agenda (and differed agenda) of norms.

The tree and near-optimal patterns with respect to a combination of norms are carried out under the following constraints, discussed earlier:

- Branchiness for each location.
- Planarity (for near-optimal solution only).
- Knowledge base of recommended links.
- Knowledge base of prohibited-accesses.

These constraints contribute to generation of more realistic design solutions. The following sections describe algorithms and depict flow diagrams corresponding to different task-executers in TOPGENE.

5.18 Generating near-optimal and tree type designs with respect to multiple norms

The multi-norm-near-optimal-task-manager is responsible for generating near-optimal designs for a set of locations with respect to a combination of norms. This task-manager has three task-executers in its disposal each for optimizing partial designs with respect to a single norm.

The multi-norm task-executer for generating tree type designs is identical with the multi-norm near-optimal task-executer , except that the cyclic paths are avoided in this type of designs. For this reason, TOPGENE uses the same procedure (task-executers) for generating tree and near-optimal design types with respect to multiple norm if possible. The task-executers not being shared between them are not also too different. For this reason stating similar algorithms for these two design types are skipped. The task-diagram and the data diagram, For these design types are depicted in figures 5.16 and 5.18.

```
Procedure MULTI-NORM-COMMUNITY-TASK-EXECUTER
          (PHC_PAIRS processed_locs partial_design)
1- LET PD be the partial design.
2- FOR PAIR ∈ PHC_PAIRS, UNTIL SUCCESS:
  2.1- LET loc1 BE (FIRST  PAIR).
  2.2- LET loc2 BE (SECOND  PAIR).
    COND  - (loc1 is already processed),
          - (loc2 is already processed).
    THEN  - Do nothing.
    COND  - loc1 is already processed.
    THEN  - LET DECC_LOCS be a list of locations in PD sorted in descending
            order of their eccentricity with respect to loc1.
```

```
          - FOR loc := (POP DECC_LOCS):
            - LET link (LIST loc loc2).
            - LET branch1 := branching degree of loc.
            - LET branch2 := branching degree of loc2.
            - IF:  - (link ∉ prohibited_accesses),
                   - (branch1 > 0),
                   - (branch2 >0).
              THEN: - Add link to PD.
                   - Update branchiness of the link nodes,
                   - (PUSH loc2 processed_locs),
                   - success := TRUE,
                   - Break the loop.
        -END-FOR
    COND - loc1 is already processed.
    THEN - LET DECC_LOCS be a list of locations in PD sorted in descending
           order of their eccentricity with respect to loc2.
         - FOR loc := (POP DECC_LOCS):
            - LET link (LIST loc loc1).
            - LET branch1 := branching degree of loc.
            - LET branch2 := branching degree of loc1.
            - IF:   - (link ∉ prohibited_accesses),
                    - (branch1 > 0),
                    - (branch2  > 0).
              THEN: - Add link to PD,
                    - Update branchiness of the link nodes,
                    - (PUSH loc1 processed_locs),
                    - success := TRUE,
                    - Break the loop.
         - END-FOR.
    OTHERWISE - Do nothing.
    - IF: - success. THEN: - break the loop.
3- END-FOR.
4- END-PROCEDURE.

Procedure MULTI-NORM-NEAR-OPTIMAL-PRIVACY-TASK-EXECUTER
          (PHC_PAIRS processed_locs partial_design)
1- LET PD be the partial design.
2- FOR PAIR ∈ PHC_PAIRS, UNTIL SUCCESS:
  2.1 - LET loc1 BE (FIRST  PAIR).
  2.2 - LET loc2 BE (SECOND  PAIR).
  2.3 - LET branch1 := branching degree of loc1.
  2.4 - LET branch2 := branching degree of loc2.
    COND  - (loc1 is already processed),
          - (loc2 is already processed).
    THEN       IF:   - (link ∉ prohibited_accesses),
                     - (branch1 > 0),
                     - (branch2 > 0),
                     - (pair + PD) is planar.
               THEN: - Add link to PD,
                     - Update branchiness of the link nodes,.
                     - (PUSH loc2 processed_locs),
                     - success := TRUE,
                     - Remove pair from PHC_PAIRS.,
    COND  - loc1 is already processed.
```

```
    THEN  - LET neighbors be a list of locations in PD in ascending order of
              their eccentricity with respect to loc1.
          - (PUSH loc1 neighbors).
          - FOR loc := (POP neighbors):
            - LET - link := (LIST loc loc2).
            - LET branch := branching degree of loc.
              IF:   - (link ∉ prohibited_accesses),
                    - (branch > 0).
              THEN: - Add link to PD,
                    - Update branchiness of the link nodes,
                    - (PUSH loc2 processed_locs),
                    - (PUSH pair success),
                    - Break the loop.
            - END-FOR.
    COND  - loc2 is already processed.
    THEN  - LET neighbors be a list of locations in PD in ascending order of
              their eccentricity with respect to loc2.
          - (PUSH loc2 neighbors).
          - FOR loc := (POP neighbors):
            - LET - link := (LIST loc loc1).
            - LET branch := branching degree of loc.
              IF:   - (link ∉ prohibited_accesses) & - (branch > 0).
              THEN: - Add link to PD,
                    - Update branchiness of the link nodes,
                    - (PUSH loc1 processed_locs),
                    - (PUSH pair success),
                    - Break the loop.
            - END-FOR.
    OTHERWISE - Do nothing.
    - IF - success THEN: - break the loop.
3- END-FOR.
4- RETURN PHC_PAIRS and processed_locs.
5- END MN-OPTIMAL-PR-TASK-EXECUTER procedure.

Procedure MULTI_NORM_IO_TASK_EXECUTER
          (IO_triplets processed_locs partial_design)
1- LET PD be the partial design.
2- FOR IO_triplet ∈ IO_triplets, UNTIL SUCCESS:
  2.1 - LET loc1 be (FIRST  IO_triplet).
  2.2 - LET loc2 be (SECOND IO_triplet).
  2.3 - LET loc3 be (THIRD  IO_triplet).
    COND  - (loc1 & loc2 & loc3 are not processed).
    THEN  IF:   - (LIST loc1 loc2) ∉ prohibited_accesses,
                - (LIST loc2 loc3) ∉ prohibited_accesses.
          THEN: - FOR loc ∈ processed_locs UNTIL: processed_locs exhausted.
                    - LET branch := branching degree of loc.
                      IF:- (LIST loc loc3) ∉ prohibited_accesses,
                         - (branch >0).
                      THEN: - TEMP := LOC,
                            - Break the loop.
                - END-FOR.
                  IF:     - temp.
                  THEN:   - Add (LIST temp loc3) to PD,
                          - Update branching degree of loc3,
```

```
                        - Update branching degree of temp,
                        - (PUSH loc3 processed_locs),
                        - Add (LIST loc2 loc3) to PD,
                        - Update branching degree of loc2,
                        - Update branching degree of loc3,
                        - (PUSH loc2 processed_locs),
                        - Add (LIST loc1 loc2) to PD,
                        - Update branching degree of loc1,
                        - Update branching degree of loc2,
                        - (PUSH loc1 processed_locs),
                        - success := TRUE,
                        - Remove IO_triplets from IO_triplets,
                        - Break the loop.
COND - (loc1 is not processed),
     - (loc2 is not processed).
THEN - LET neighbors be a list of locations in PD in ascending order of
          their eccentricity with respect to location loc3.
     - (PUSH loc3 neighbors).
     - FOR loc ∈ neighbors:
       - LET branch := branching degree of loc.
       - IF:   - ((LIST loc2 loc) ∉ prohibited_accesses), AND - (branch
                 > 0).
         THEN: - Add (LIST loc2 loc) to the PD,
               - Add (LIST temp loc3) to PD,
               - Update branching degree of loc2,
               - Update branching degree of loc,
               - (PUSH loc2 processed_locs),
               - Add (LIST loc1 loc2) to PD,
               - Update branching degree of loc2,
               - Update branching degree of loc1,
               - (PUSH loc1 processed_locs),
               - (PUSH loc1 processed_locs),
               - success := TRUE,
               - Remove IO_triplets from IO_triplets,
               - Break the loop.
     - END_FOR
COND - loc1 and loc3 are not processed.
THEN - LET paths be a list of all paths from loc2 location.
     - FOR path ∈ paths:
       - FOR loc ∈ path:
           - LET branch := branching degree of loc.
             IF   - (LIST loc1 loc) ∉ prohibited_accesses, & (branch
                    > 0).
             THEN:- Add (LIST loc1 loc) to the PD,
                  - Update branching degree of loc1,
                  - Update branching degree of loc,
                  - (PUSH loc1 processed_locs),
                  - temp := (LIST loc1 loc),
                    IF:    - number of paths > 1.
                    THEN:  - Remove the path from paths,
                           - Break the loop.
       - END_FOR.
       - Remove path from paths.
       - Break the loop.
```

```
    - END-FOR.
    - IF: - success THEN: - Break the loop.
    - IF: - TEMP.
        - FOR path ∈ paths:
          - FOR loc ∈ path:
                - LET branch := branching degree of loc.
                  IF:  - (LIST loc3 loc) ∉ prohibited_accesses,
                       - (branch > 0).
                  THEN:- Add (LIST loc3 loc) to the PD,
                       - Update branching degree of loc3,
                       - Update branching degree of loc,
                       - (PUSH loc3 processed_locs),
                       - success := TRUE,
                       - Remove the path from paths,
                       - Break the loop.
          - END-FOR.
          -IF: - success THEN: - Break the loop.
    - END-FOR.
COND  - loc2 and loc3 are not processed.
THEN  - LET neighbors be a list of locations in PD in ascending order of
          their eccentricity with respect to location loc1.
        - (PUSH loc1 neighbors).
        - FOR loc ∈ neighbors:
          - LET branch := branching degree of loc.
            IF:  - (LIST loc2 loc) ∉ prohibited_accesses,
                 - (branch > 0).
            THEN - Add (LIST loc2 loc) to the PD,
                 - Update branching degree of loc2,
                 - Update branching degree of loc,
                 - (PUSH loc2 processed_locs),
                 - Add (LIST loc3 loc2) to PD,
                 - Update branching degree of loc3,
                 - Update branching degree of loc2,
                 - (PUSH loc3 processed_locs),
                 - success := TRUE,
                 - Remove IO_triplets from IO_triplets,
                 - Break the loop.
        - END-FOR.
COND  - loc1 is not processed
THEN  - LET paths := list of all paths from loc2 ∈ PD that do not cross
          loc3.
        - FOR path ∈ paths:
        - FOR loc ∈ path:
                - LET branch := branching degree of loc.
                  IF:  - (LIST loc1 loc) ∉ prohibited_accesses,
                       - (branch > 0).
                  THEN:- Add (LIST loc1 loc) to the PD,
                       - Update branching degree of loc1,
                       - Update branching degree of loc,
                       - (PUSH loc1 processed_locs),
                       - success := TRUE,
                       - remove IO_triplet from the IO_triplets,
                       - Break the loop.
        - END-FOR.
```

```
        - IF: - success THEN: - break the loop.
    COND  - loc3 is not processed.
    THEN  - LET paths := list of all paths from loc2 in PD that do not cross
                loc1.
          - FOR path ∈ paths:
          - FOR loc ∈ path:
                  - LET branch := branching degree of loc.
                    IF:  - (LIST loc1 loc) ∉ prohibited_accesses,
                         - (branch >0).
                    THEN - Add (LIST loc loc3) to the PD,
                         - Update branching degree of loc3,
                         - Update branching degree of loc,
                         - (PUSH loc3 processed_locs),
                         - success := TRUE,
                         - remove IO_triplet from the IO_triplets,
                         - Break the loop.
          - END-FOR.
          - IF: - success. THEN - Break the loop.
    COND  - loc2 is not processed.
    THEN  - WHEN: type is near_optimal
                  - LET branch1 := branching degree of loc1.
                  - LET branch2 := branching degree of loc3.
                    IF:  - (LIST loc1 loc2) ∉ prohibited_accesses,
                         - (LIST loc2 loc3) is not a prohibited_accesses,
                         - {(LIST loc1 loc2) + (loc2 loc3) + PD} is planar,
                         - (branch1 > 0) & (branch2 > 0).
                    THEN:- Add (LIST loc1 loc2) to the PD,
                         - Update branching degree of loc1,
                         - Update branching degree of loc2,
                         - (PUSH loc2 processed_locs),
                         - Add (LIST loc3 loc2) to the PD,
                         - Update branching degree of loc3,
                         - Update branching degree of loc2,
                         - success := TRUE,
                         - Remove IO_triplet from the IO_triplets.
                         - Break the loop.
     OTHERWISE - Remove IO_triplet from the IO_triplets.
  2.4 - IF: - success. THEN - Break the loop.
3- END-FOR.
4- RETURN IO_triplets and processed_locs.
5- END MULTI-NORM-IO-TASK-EXECUTER.
```

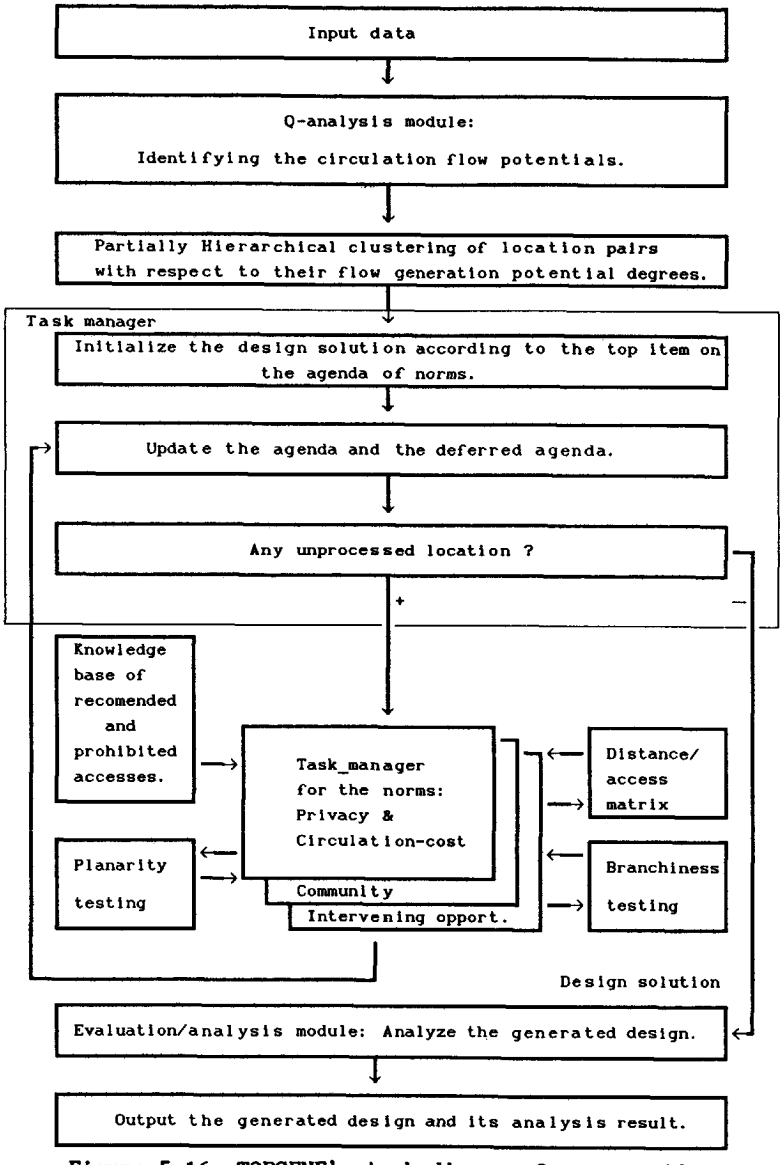


Figure 5.16: TOPGENE's task diagram for generating "near-optimal" designs with respect to a combination of norms

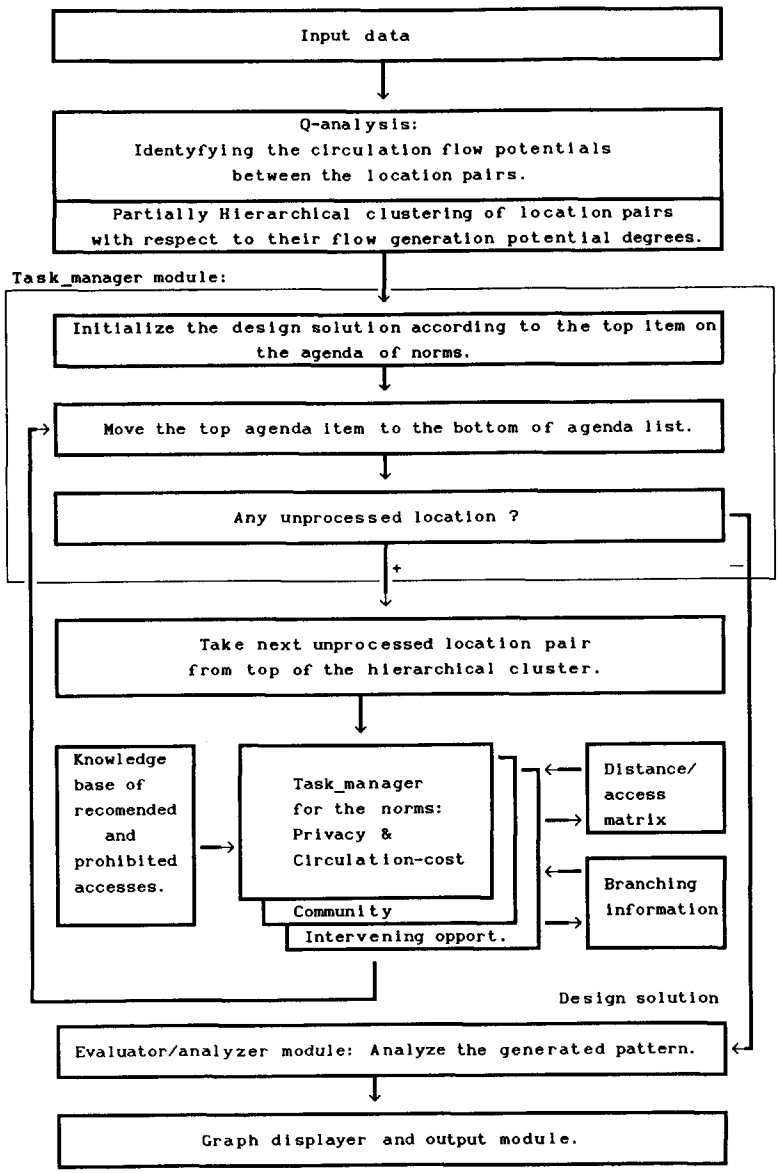


Figure 5.17: TOPGENE's task diagram for generating "Tree" designs with respect to a combination of norms

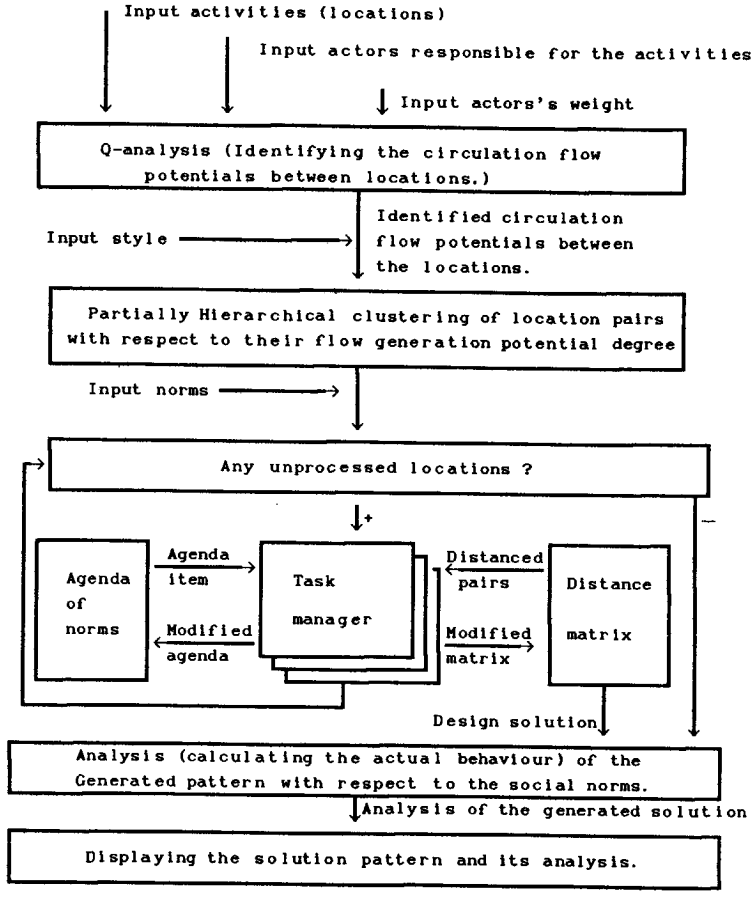


Figure 5.18: TOPGENE's data flow diagram for generating "tree" or "near-optimal" designs with respect to a combination of norms

5.19 Generating Prototypical designs with respect to multiple norms

The approach in generating a prototypical designs with respect to a combination of norms is almost the same as the one used for generating a design with respect to a single norm. TOPGENE In an attempt to generate a prototypical solution, first generate a stream of locations with respect to the multiple norms, and feeds it to the pattern generator routine for a final arrangement of the locations in the stream on a required pattern. The task-manager for the prototypical designs, as in the two other cases, has three

specialized task-executers each responsible for improving the partial design with respect to a specific norm. The task-manager calls these task-executers, based on the order defined by the deferred agenda of norms.

Figure 5.19 shows the task and data flow diagram for generating prototypical designs with respect to a combination of norms.

5.19.1 Generating a stream of locations

The multi-norm prototypical task-manager generates a stream of locations using a heuristic based hill-climbing strategy. This task-manager by iteratively calling different task-executers, each responsible for improving a partial design with respect to a single norm, passes a partial design (stream), and the list of unprocessed location pairs (resulting from scrapping of the partially hierarchical clusters of activity pairs to them), and receives an improved and sometimes unimproved) version of the partial design with possibly a reduced list of location-pairs.

The core of the task-executer algorithms is described below.

```
Procedure MN-PROTOTYPICAL-COMMUNITY-TASK-EXECUTER (stream ordered_pairs)
1- FOR pair ∈ ordered_pairs:
  1.1 LET loc1 be (FIRST  pair).
  1.2 LET loc2 be (SECOND pair).
    COND  - (loc1 is not processed),
          - (loc2 is not processed).
    THEN  - Do nothing.
    COND  - loc1 is not processed.
    THEN  - Allocate loc1 to the end location of the stream which is
            farthest from loc2.
          - success := TRUE.
    COND  - loc2 is not processed.
    THEN  - Allocate loc2 to the end location of the stream which is
            farthest from loc1.
          - success := TRUE.
    OTHERWISE - DO nothing.
  1.3 IF: - success THEN: Break the loop.
- END-FOR.
- Remove the processed pair from the ordered_pairs.
- RETURN stream & ordered_pairs.
- END of procedure.

Procedure MN-PROTOTYPICAL-PRIVACY-TASK-EXECUTER (stream ordered_pairs)
1- FOR pair ∈ ORDERED_PAIRS:
  1.1 LET loc1 be (FIRST  pair).
  1.2 LET loc2 be (SECOND pair).
```

```
          COND  - (loc1 & loc2 are not processed).
          THEN  - Do nothing.
          COND  - loc1 is not processed.
          THEN  - Allocate loc1 to the end location of the stream nearest
                  to loc2.
                - success := TRUE
          COND  - loc2 is not processed
          THEN  - Allocate loc2 to the end location of the stream which is
                  nearest to loc1.
                - success := TRUE.
          OTHERWISE - DO nothing.
  - IF: - success THEN: - Break the loop.
- END-FOR.
- Remove the processed pair from the ordered_pair.
- RETURN stream ordered_pairs.
- END of procedure.


Procedure MN_PROTOTYPICAL_IO_TASK_EXECUTER (stream IO_triplets)
1- FOR IO_triplet ∈ IO_triplet:
  1.1 LET loc1 be (FIRST  pair).
  1.2 LET loc2 be (SECOND pair).
  1.3 LET loc3 be (THIRD  pair).
          COND  - (loc1 & loc2 & loc3 are not processed).
          THEN  - Do nothing.
          COND  - (loc1 & loc2 are not processed).
          THEN  - stream := {(LIST loc1 loc2) + partial design}.
                - success := IO_triplet.
          COND  - (loc1 & loc3 are not processed).
          THEN  - (PUSH loc1 stream).
                - stream := stream + loc3.
                - success := IO_triplet.
          COND  - (loc2 & loc3 are not processed).
          THEN  - stream := {stream + (LIST loc2 loc3)}.
                - success := IO_triplet.
          COND  - loc1 is not processed
          THEN  - PUSH loc1 to the end of the stream closest to loc2.
                - success := IO_triplet.
          COND  - loc3 is not processed
          THEN  - PUSH loc3 to the end of the stream closest to loc2.
                - success := IO_triplet.
          COND  - loc2 is not processed
          THEN  - Put loc2 in_between locations loc1 and loc3 in the stream.
          OTHERWISE - Remove IO_triplet from the IO_triplets.
- IF:  - success.
  THEN  - Break the loop.
- END-FOR.
- RETURN IO_triplets and processed_locs.
- END of procedure.
```

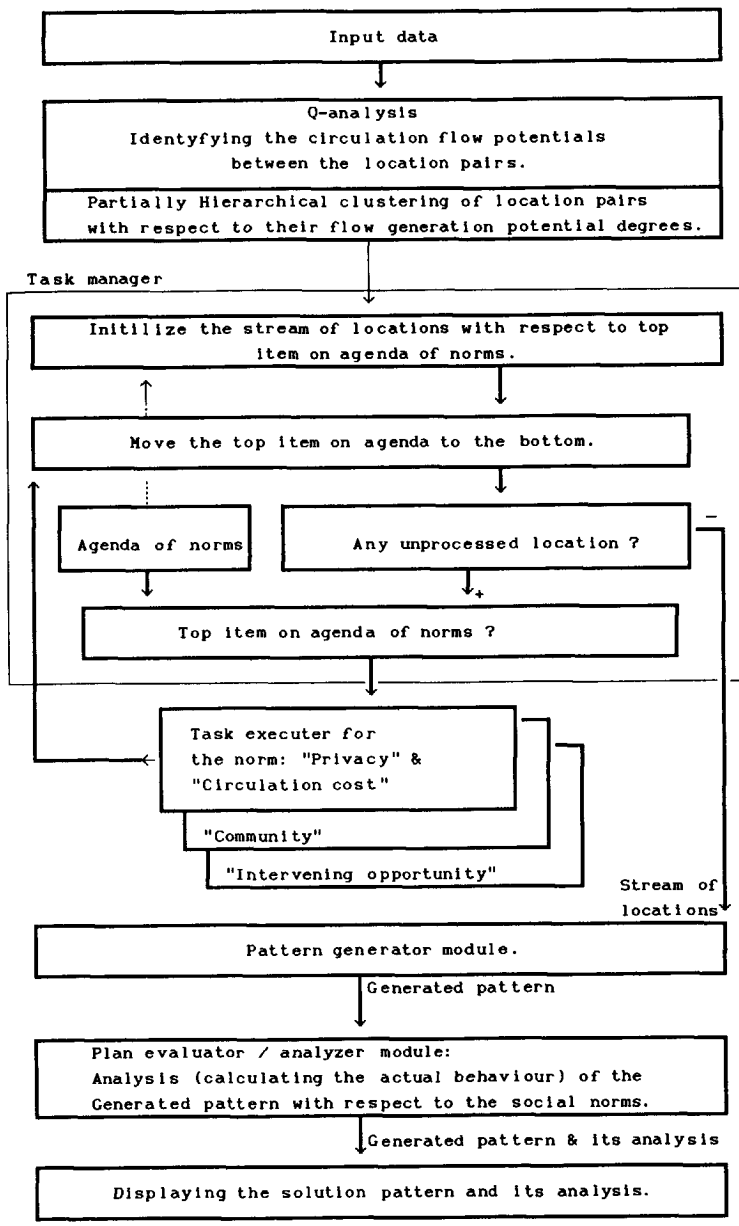


Figure 5.19: TOPGENE's task and data-flow diagram for generating "prototypical" designs with respect to a combination of norms

5.20 Analysis, diagnosis, and evaluation of designs

> "And how will you inquire, Socrates, into that which you do not know? What will you put forth as the subject of inquiry? And if you find what you want, how will you ever know that this is what you did not know?" Plato's Meno

This section discusses the analysis, diagnosis, and evaluation of a design, as it is carried out by TOPGENE. The definition of these terms was given in chapter 2.

Many factors may contribute to the analysis of a building at topological level. The following figure lists topological and operational variables contributing to the behavior of buildings with respect to the social norms discussed in this thesis.

Design-description space: Design variables	**Performance space:** Behavioral criteria
- Connectivity structure: . Paths between locations. . Intervening locations. . Locations beyound a location. . Branchiness of each location. . Penetration.	- Community-utility. - Privacy-cost. - Intervening-opportunity - Circulation-cost.
- Operation: . Activity assigned to each location. . Actors responsible for each activity. . Flow potential between locations. . A flow meeting other flows on a location. . Effect of proximate flow with respect to a location	
- Metric distances between the locations.	
- Usage time.	

Figure 5.20: Some factors relating a building description to its performance criteria

Some of the variables, shown above, may be more important than the others for analysis of a design, and some may be implicitly part of the others. For example, in the absence of quantitative data such as flow generation degree between location pairs in a design, an analysis or evaluation procedure may be carried out based on qualitative analysis of the building and heuristics rules. In such a case, available information, such as branchiness and penetration (see appendix-A for a definition) degrees, for example, have impor-

tant role in evaluation of building with respect to the social norms. A high branching degree building has potential for a high performance with respect to the privacy norm. Or, a location beyond a set of other locations cannot perform as an intervening location for them. However in our case, the existence of operational information such as flow, makes the use of qualitative analysis of a design unnecessary. As in the case of design generation, most of the information needed here are also prepared by the Q-analysis method. The same process carried out for the design generation, such as finding the flow generation potential between different location pairs, and hierarchical clustering of location-pairs is applied here as well.

There are also variables in above table that are either disregarded, or taken in an special way in the current implementation of TOPGENE. For example, metric distances are neglected by TOPGENE. TOPGENE, in general, carries out the analysis and evaluation of design under the assumptions mentioned in section 5.2.

The plan evaluator module of TOPGENE, is used for two purposes: The analysis and diagnosis of designs generated by the plan generator module, and the evaluation of existing designs presented to the system. In the first case the plan evaluator takes the generated design, analyze and produces its analysis results in numerical forms. An analysis of a design by TOPGENE results in the actual behavioral values of the design with respect to the norms under the consideration.

Diagnosis was defined as the process of finding the causes contributing to certain behavior or misbehavior of a system. Diagnosis of many physical systems, often starts with observation of a misbehavior (symptom) from the system, assuming a misbehavior as a result of malfunctioning of a specific part, and investigating if the misbehavior of that part explains the observed symptom [Struss88].

The hybrid analytical and qualitative approach by TOPGENE, as described in chapter one, is better than a pure qualitative approach for the same purpose. The use of Q-analysis in analyzing design data, and identification of potential flow degrees between location pairs in a design is the main contributor to analytical power of the system. All the norms discussed in this work, as described in chapter two, are under direct influence of the flow degrees between location-pairs in a design. Other factors contributing to the analysis, diagnosis, and evaluation of a building, are the weights

associated with the actors and flow indices between location-pairs. TOPGENE also allows optional input of these data. The total behavior of a design consists of the aggregation of its behaviors at micro-levels (locations).

According to the definition of the social norms in chapter 2, the behavior of a building with respect to these norms is under direct influence of the pattern of flow in a building. For example meetings of flow or absence interactions between the flows in a building have effect on the behavior of that building with respect to the privacy and community norms. Similarly, the intervening opportunity norm is defined by the passage of particular flows on certain locations. The circulation-cost on the other hand is under the influence of flows between location-pairs. TOPGENE provides the user causes of behavior of a building system with respect to the social norms, without going to detailed explanation. The following paragraphs display a diagnosis report on a generated design with respect to the social norms forms:

Diagnosis of design with respect to the community norm:
The community utility for location TV-ROOM is 67 units.
The community utility for location BEDROOM-3 is 20 units.
The community utility for location KITCHEN-1 is 5 units.
etc.

Diagnosis of design with respect to the norm privacy:
The privacy cost for TV-ROOM is 67 units.
The privacy cost for BEDROOM-3 is 20 units.
The privacy cost for KITCHEN-1 is 5 units.
etc.

Diagnosis of design with respect to the norm circulation cost:
The circulation cost between BEDROOM-1 and LIVING-ROOM is: 125 units.
The circulation cost between BEDROOM-2 and LIVING-ROOM is: 87 units.
The circulation cost between HALL and LIVING-ROOM is: 150 units.
etc

Diagnosis of design with respect to the intervening opportunity norm:
The intervening opportunity of the HALL as a result of being on the shortest path in-between location pairs (KITCHEN-1 LIVING-ROOM) is 10.

The intervening opportunity of the HALL as a result of being on the shortest path in-between location pairs (BEDROOM-2 LIVING-ROOM) is 6.
The intervening opportunity of the HALL as a result of being on the shortest path in-between location pairs (BEDROOM-1 LIVING-ROOM) is 6.
etc.

I defined the analysis of a design as the act of finding the behavioral values of each location in a design with respect to the social norms, and presenting the cumulated values as the analysis result of a design. For example, the result of behavioral analysis of a generated design is usually as follows. This analysis may be limited to fewer norms, if information about a norm is not available to the system.

Community utility: 1272 units utility.
Privacy cost : 1074 units cost.
Circulation cost : 2108 units cost.
Intervening opportunity utility: 28 units utility.

The analysis and derivation of actual performance behavior of buildings are carried out by TOPGENE for both generated designs and designs presented to the system, but evaluation is limited to the existing designs.

By evaluating an existing building with respect to the social norms, we are interested in the system to judge the actual behavior of the system with respect to an expected behavior. The problem, here is that the expected behavior of a design presented to the system is not explicitly given to the system. The only available information to the system is design description, its operation to a certain degree, and a set of social norms reflection of user's point of views in judging the design. To fill the data gaps, TOPGENE proceeds with the following steps:

- Applies Q-analysis on the activities and actors to:
  - . Derive the potential interactions between activity pairs in design.
  - . Cluster partially hierarchical the activity pairs with respect to their potential interactions.
- Analyzes the building based on the Q-analysis results to calculate its actual behavior.
- Generates sub-optimal yardstick solutions with respect to each social

norms.

- Analyzes the yardstick solutions based on the Q-analysis results to calculate their behavioral values with respect to the norms. The behavioral values of the yardstick solutions are taken as the expected behavior of the input design.
- Compares the actual and expected behavioral values of the input design to judge actual behavior (i.e., evaluate).

In addition TOPGENE allows the user to decide the type of yardstick designs for evaluation. This option provides several opportunity for a user to evaluate a design with respect to a set of social norms. Other advantage of this option is that it prohibit the system from unfair evaluate of an existing design in certain circumstances. For example, in many cases it is unfair to evaluate the social behavior of a design delivered under certain structural constraint against, for example, a near-optimal design. A design with a certain structural type in many cases is better evaluated if it is judged against a design for the same set of data and with the same structural type but optimized with respect to certain criteria.

Here is result of evaluation of a design by TOPGENE with respect to the community, privacy, and circulation-cost norms:

Result of the behavioral analysis of the input design and yardstick designs:

Design	Community utility	Privacy cost	Circulation cost
Input design	1474	240	2422
Community yardstick	5141	0	6089
Privacy yardstick	1814	0	2762
Circ.-cost yardstick	1814	0	2762

Evaluation result:

The community utility of the input design is about 29% of the yardstick solution, and it is very bad.

The privacy cost is 240 unit, which is higher than the generated yardstick solution.

The circulation-cost of the input design is about 12% lower than the generated yardstick solution, and it is excellent.

The task-diagram and data-flow diagram for the whole process of evaluating a design is depicted below. Examples of designs activities carried out by TOPGENE will be given in the following chapter.

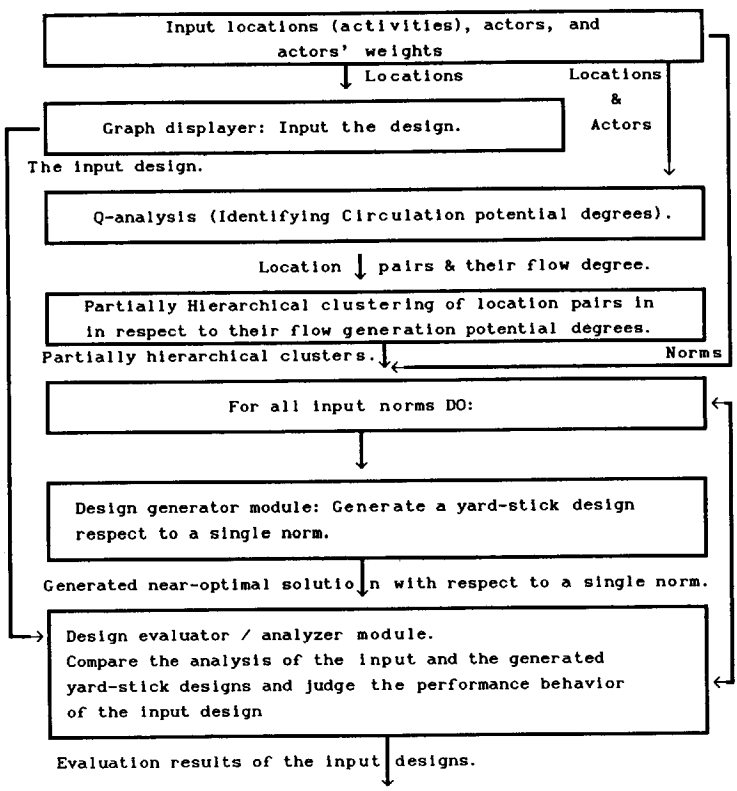


Figure 5.21: TOPGENE's Data flow diagram for evaluation a design with respect to a set of social norms

Next chapter examines examples of designs generated by TOPGENE under different design requirements.

Chapter 6
Experimenting with TOPGENE
Examples and results

When you can measure what you are speaking about, and express it in numbers, you know something about it.
(Lord Kelvin)

This chapter presents examples of design generation, and design evaluation sub-processes carried out by TOPGENE. The chapter also gives an analysis of TOPGENE's behavior in currying these design sub-processes.
TOPGENE follows a design generation by a behavioral analysis of design with respect to the social norms. The system also can give a diagnosis of the design for its behavior upon the user's request. The behavioral analysis and diagnosis results are in numerical form. These results provide us a means for analyzing the performance of TOPGENE itself.

In this chapter, first I present three different sets of design data, upon which experiments with TOPGENE are based. Each proceeding section, give examples of design types generated by the system. These examples are followed by a table consisting of the result of the behavioral analysis of different types of designs generated under the same requirements. These tables are complete in terms of covering design types; but, they are not exhaustive in terms of covering all designs that may be produced for a set of data but with respect to different set of requirements.

The concluding subsection evaluates TOPGENE. An analysis of behavior of designs generated with respect to single norms helps us to evaluate TOPGENE's behavior in generating designs in general. The reason is that TOPGENE generates designs with respect to a combination of norms based on the same heuristic rules that it uses for generating sub-optimal designs with respect to single norms. The evaluation of system with respect to single norms is approached from two perspectives:

- The structural analysis of generated designs.
- The behavioral analysis of generated designs.

TOPGENE also may be evaluated by examination of designs that it generates with respect to a set of norms against corresponding sub-optimal designs with respects to single norms. The difficulty, here, is obtaining of sub-optimal

designs. This difficulty is related to the intractability of our design problem. The generation of sub-optimal designs are only possible through a brute-force approach. However, partial implementation of the design problem in a neural network has provided us with a means for partial testing of TOPGENE. The neural network approach, to be discussed in the next chapter, is a brute force approach comparing to TOPGENE's heuristic strategy. The network generates only linear-tree type designs with respect to the community and privacy / circulation-cost norm(s). These designs not necessarily optimal, but they are in the neighborhood of optimal designs. Besides, the linear type designs are computationally among the most complex types of designs. For this reason the network solutions provide us an essential means for testing the TOPGENE's heuristic power with respect to some social norms.

6.1 Examples of design problems

To test TOPGENE, three sets of design data are gathered. These data-sets, formulated by an architectural student or extracted from architectural books, have been sufficient for formulating a wide range of design generation and design evaluation problems.

TOPGENE can generate various types of designs with different social behaviors for a set of data. This, in turn, provides many possibilities for evaluating an existing design. Evaluation is carried out by comparing the values of behavioral criteria of an existing design against values the same criteria for a set of generated yardstick designs.

I have to mention beforehand that some formulation of design problems presented in the following sections may seem unrealistic. This is also true for some designs. For example, demanding a linear-tree design with community performance for a house may occur rarely in practice. Yet, this design is valuable in situations, such as arrangement spaces for public usage that requires community performance. This odd and abnormal formulations of design problems are, therefore, used purposely to show the capability of the system in producing a diverse range of designs with different behaviors. Here are the data-sets:

6.1.1 Data-set 1: A house

The first data-set belongs to a small-scale house with 12 locations. The locations of the house and their users are shown in the following table:

Locations	Actors
1- BEDROOM-1	DAUGHTER-1 DAUGHTER-2

2- BEDROOM-2	SON-1 SON-2
3- BEDROOM-3	FATHER MOTHER
4- BATH-ROOM	SONS FATHER MOTHER GUESTS DAUGHTERS STUDY-ROOM FATHER
6- TV-ROOM	SONS FATHER MOTHER GUESTS DAUGHTERS
7- TOILET	SONS FATHER MOTHER GUESTS DAUGHTERS
8- KITCHEN-2	MOTHER
9- KITCHEN-1	MOTHER
10- BEDROOM-4	GUESTS
11- LIVING-ROOM	SONS FATHER MOTHER GUESTS DAUGHTERS
12- HALL	SONS FATHER GUESTS MOTHER DAUGHTERS

Furthermore, a complementary data consisting of a table of flow degree between location-pairs of the house is assumed to be available. This table may be the actual flow between the location-pairs in the design gathered over a period by observation, or may reflect the expected flow between different location-pairs is the design.

Locations

	1	2	3	4	5	6	7	8	9	10	11	12
1		0	0	5	0	2	2	0	1	0	3	5
2			0	5	0	2	2	0	1	0	3	5
3				7	2	2	2	0	2	0	2	5
4					1	0	10	0	5	1	2	4
5						4	2	0	1	0	6	3
6							5	0	4	2	8	3
7								0	6	2	14	4
8									10	0	0	5
9										1	10	6
10											2	1
11												6

This table enriches the data-set. A complete data-set naturally helps TOPGENE in producing a more reliable design in response to a problem. Experiments show that the absence of detailed information, such as the expected flow between locations of a building, or lack of design knowledge (e.g., knowledge of recommended and prohibited accesses in buildings) may result in undesirable designs. For example, above table shows that all actors in a house, such as father, mother, sons, and daughters use most of the locations. Such a table clearly confuses any designer, including TOPGENE, while trying to analyze and hierarchically cluster the location-pairs with respect to their interaction degree. Addition of more information to the data-set, or integration of knowledge base of access with the system eliminates possible ambiguities. Both of these solutions was implemented in TOPGENE and considerable improvement in the quality of design was gained.

6.1.2 Data-set 2: A police-station

The second data-set belongs to a police-station with two floors. The locations (activities) and actors assigned to them for different floors are reflected

in the following tables. No flow indices are assumed between the locations of this building.

Locations in the first floor	Actors
1- CORRIDOR	
2- KITCHEN	COFFEE-MAIDS
3- PERSONNEL-TOILET	INSTRUCTORS PERSONNEL PROPERTY-KEEPER PORTER COFFEE-MAIDS WARDER
4- OFFICES	PERSONNEL
5- INSTRUCTION-ROOM3	OFFICERS INSTRUCTORS
6- INSTRUCTION-ROOM2	OFFICERS INSTRUCTORS
7- INSTRUCTION-ROOM1	OFFICERS INSTRUCTORS
8- CONFERENCE-ROOM2	PERSONNEL SECRETARIES DEPUTY CHIEF
9- CONFERENCE-ROOM1	PERSONNEL SECRETARIES DEPUTY CHIEF
10- RESTAURANT	OFFICES INSTRUCTORS PROPERTY-KEEPER PERSONNEL PORTER COFFEE-MAIDS WARDER SECRETARIES DEPUTY CHIEF
11- STAFF-TOILET	SECRETARIES DEPUTY CHIEF
12- ARCHIVE	SECRETARIES
13- SECRETARY-ROOM	SECRETARIES
14- CHIEF-ROOM	SECRETARIES DEPUTY CHIEF

Locations in the second-floor	Actors
1- CORRIDOR	
2- VISITORS-ENTRANCE	VISITORS PORTER
3- PERSONNEL-ENTRANCE	OFFICERS INSTRUCTORS PERSONNEL PROPERTY-KEEPER PORTER COFFEE-MAIDS SECRETARIES WARDER DEPUTY CHIEF
4- RECEPTION2	VISITORS PERSONNEL
5- RECEPTION1	VISITORS PERSONNEL
6- BICYCLE-SHED	OFFICERS PERSONNEL PORTER
7- GARAGE	OFFICERS DEPUTY
8- PROPERTY-INTAKE	PROPERTY-KEEPER WARDER
9- VISITORS-TOILET	VISITORS
10- PROPERTY-STORAGE	PROPERTY-KEEPER WARDER
11- BRIEFING-ROOM	OFFICERS INSTRUCTORS DEPUTY
12- DRESSING-ROOM3	INSTRUCTORS PERSONNEL
13- DRESSING-ROOM2	OFFICERS
14- DRESSING-ROOM1	OFFICERS
15- REPORT-ROOM2	PERSONNEL
16- REPORT-ROOM1	PERSONNEL DEPUTY
17- PORTERS-LODGE	PORTER
18- CELLS	PRISONERS WARDER
19- PRISONERS-ENTRANCE	OFFICERS PRISONERS WARDER

6.1.3 Data-set 3: A hospital

Data-set 3 belongs to a four-stories hospital. This data-set is used to show the capability of the TOPGENE in evaluating existing designs. The locations and actors responsible for the actors in the hospital are as follows. I have not separated locations of hospital in different floors, since they will be evident in the last section, where the connectivity pattern of existing design for each floor is displayed.

Locations	Actors
1- SURGERY	SURGERY-PATIENTSSURGERYNURSESSURGERY-PATIENTS-HELPSSURGEONSINTERNISTS PHYSIOTHERAPISTS CLERGYMAN VISITORS MENTAL-ATTENDANTS
2- UROLOGY	UROLOGY-PATIENTS UROLOGY-NURSES UROLOGY-PATIENTS-HELPS SURGEONS UROLOGISTS INTERNISTS PHYSIOTHERAPISTS CLERGYMAN MENTAL-ATTENDANTS VISITORS
3- ORTHOPEDY	ORTHOPEDY-PATIENTS ORTHOPEDY-NURSES ORTHOPEDY-PATIENTS-HELPS SURGEONS ORTHOPAEDISTS INTERNISTS PHYSIOTHERAPISTS CLERGYMAN MENTAL-ATTENDANTS VISITORS
4- JAW-SURGERY	JAW-SURGERY-PATIENTS JAW-SURGERY-NURSES JAW-SURGERY-PATIENTS-HELPS SURGEONS JAW-SURGEONS INTERNISTS PHYSIOTHERAPISTS CLERGYMAN MENTAL-ATTENDANTS VISITORS
5- TNE-SURGERY	TNE-SURGERY-PATIENTS TNE-SURGERY-NURSES TNE-SURGERY-PATIENTS-HELPS SURGEONS TNE-SURGEONS INTERNISTS PHYSIOTHERAPISTS CLERGYMAN MENTAL-ATTENDANTS VISITORS
6- EYE-SURGERY	EYE-SURGERY-PATIENTS EYE-SURGERY-NURSES EYE-SURGERY-PATIENTS-HELPS SURGEONS EYE-SURGEONS INTERNISTS PHYSIOTHERAPISTS CLERGYMAN MENTAL-ATTENDANTS VISITORS
7- CARDIOLOGY	CARDIOLOGY-PATIENTSCARDIOLOGY-NURSESCARDIOLOGY-PATIENTS-HELPSSURGEONS CARDIOLOGISTS INTERNISTS PHYSIOTHERAPISTS CLERGYMAN MENTAL-ATTENDANTS VISITORS
8- PULMONOLOGY	EYE-SURGERY-PATIENTS EYE-SURGERY-NURSES EYE-SURGERY-PATIENTS-HELPS SURGEONS EYE-SURGEONS INTERNISTS PHYSIOTHERAPISTS CLERGYMAN MENTAL-ATTENDANTS VISITORS
9- INTERNAL-SURGERY	INTERNAL-SURGERY-PATIENTS INTERNAL-SURGERY-NURSES INTERNISTS INTERNAL-SURGERY-PATIENTS-HELPS SURGEONS PHYSIOTHERAPISTS CLERGYMAN MENTAL-ATTENDANTS VISITORS
10- DERMATOLOGY	DERMATOLOGY-PATIENTS DERMATOLOGY-NURSES DERMATOLOGY-PATIENTS-HELPS SURGEONS INTERNISTS PHYSIOTHERAPISTS DERMATOLOGISTS CLERGYMAN MENTAL-ATTENDANTS VISITORS
11- GYNAECOLOGY	GYNAECOLOGY-PATIENTS GYNAECOLOGY-NURSES GYNAECOLOGY-PATIENTS-HELPS SURGEONS VISITORS INTERNISTS GYNAECOLOGISTS OBSTETRISTS PHYSIOTHERAPISTS CLERGYMAN MENTAL-ATTENDANTS
12- OBSTETRY	OBSTETRY-PATIENTS OBSTETRY-NURSES OBSTETRY-PATIENTS-HELPS SURGEONS INTERNISTS OBSTETRISTS PHYSIOTHERAPISTS CLERGYMAN MENTAL-ATTENDANTS VISITORS
13- PEADIATRY	PEADIATRY-PATIENTS PEADIATRY-NURSES PEADIATRY-PATIENTS-HELPS SURGEONS INTERNISTS PEADIATRISTS PRODUCERS PHYSIOTHERAPISTS CLERGYMAN MENTAL-ATTENDANTS VISITORS
14- NEUROLOGY	NEUROLOGY-PATIENTS NEUROLOGY-NURSES NEUROLOGY-PATIENTS-HELPSSURGEONSINTERNISTSNEUROLOGISTSPHYSIOTHERAPISTS CLERGYMAN MENTAL-ATTENDANTS VISITORS
15- PSYCHIATRY	PSYCHIATRY-PATIENTS PSYCHIATRY-NURSES PSYCHIATRY-PATIENTS-HELPS SURGEONS INTERNISTS PSYCHIATRISTS PHYSIOTHERAPISTS CLERGYMAN MENTAL-ATTENDANTS VISITORS
16- RADIOLOGY	RADIOLOGY-PATIENTS RADIOLOGY-NURSES RADIOLOGY-PATIENTS-HELPS SURGEONS INTERNISTS RADIOLOGISTS PHYSIOTHERAPISTS VISITORS
17- SPECIAL-CARE	SPECIAL-CARE-PATIENTSSPECIAL-CARE-NURSESSURGEONSCARDIOLOGISTSINTERNISTS CLERGYMAN MENTAL-ATTENDANTS
18- ADMISSION	SURGERY-PATIENTS UROLOGY-PATIENTS ORTHOPEDY-PATIENTS JAW-SURGERY-PATIENTS TNE-SURGERY-PATIENTS EYE-SURGERY-PATIENTS CARDIOLOGY-PATIENTS PULMONOLOGY-PATIENTS INTERNAL-SURGERY-PATIENTS DERMATOLOGY-PATIENTS GYNAECOLOGY-PATIENTS OBSTETRY-PATIENTS PEADIATRY-PATIENTS NEUROLOGY-PATIENTS PSYCHIATRY-PATIENTS RADIOLOGY-PATIENTS SPECIAL-CARE-PATIENTS ADMISSION-ADMINISTRATION-PERSONNEL
19- SPIRITUAL-MENTAL-CARE	SOCIAL-WORKERS CLERGYMAN MENTAL-ATTENDANTS
20- VISITORS-ACCOMMODATION	KITCHEN-PERSONNEL VISITORS
21- PATHO-PHYSIOLOGY	PATHO-PHYSIOLOGY-NURSES CARDIOLOGISTS PULMONOLOGISTS INTERNISTS NEUROLOGISTS
22- RADIO-ISOTOPE-LAB	RADIO-LABORANTS
23- RONTGENDIAGNOSIS	SURGEONS JAW-SURGEONS TNE-SURGEONS

EYE-SURGEONS CARDIOLOGISTS PULMONOLOGISTS INTERNISTS OBSTETRISTS
PEADIATRISTS RADIOLOGISTS PATHO-PHYSIOLOGISTS ROENTGEN-LABORANTS

24- OPERATING-ROOM OPERATION-NURSES ANESTHESISTS SURGEONS POLY-OR-PERSONNEL BEDS-PERSONNEL

25- RECOVERY RECOVERY-NURSES SURGEONS ANESTHESISTS POLY-OR-PERSONNEL BEDS-PERSONNEL

26- DELIVERY-ROOM GYNAECOLOGISTS ANESTHESISTS

27- PHYSICAL-THERAPY PSYCHIATRISTS REHABILITATIONISTS PHYSIOTHERAPISTS OCCUPATIONAL-THERAPISTS-1
OCCUPATIONAL-THERAPISTS-2 PATHO-PHYSIO-ADMINISTRATORS
ADMISSION-ADMINISTRATION-PERSONNEL PHYSIO-ADMINISTRATORS SOCIAL-WORKERS
PHYSIO-PERSONNEL

28- STAFF-ACCOMMODATION
STAFF-SECRETARIES

29- NIGHT-SERVICE SURGEONS RADIO-LABORANTS SPECIAL-LABORANTS TECH-SERVICES-PERSONNEL

30- POLYCLINIC-OPERATION
SURGEONS UROLOGISTS JAW-SURGEONS
GYNAECOLOGISTS ANESTHESISTS POLY-OR-PERSONNEL

31- POLYCLINIC SURGEONS UROLOGISTS ORTHOPAEDISTS JAW-SURGEONS TNE-SURGEONS EYE-SURGEONS
CARDIOLOGISTS PULMONOLOGISTS INTERNISTS DERMATOLOGISTS GYNAECOLOGISTS
PEADIATRISTS NEUROLOGISTS PSYCHIATRISTS ANESTHESISTS
STAFF-SECRETARIES POLY-PERSONNEL

32- CLINICAL-LAB PATHOLOGISTS CLINICAL-CHEMISTS
LABORANTS LAB-PERSONNEL AUTOPSY-ASSISTANTS

33- SPECIAL-LAB BACTERIOLOGISTS PATHOLOGISTS AUTOPSY-ASSISTANTS

34- PHARMACY PHARMACISTS PHARMACY-PERSONNEL

35- STERILIZATION PHARMACISTS PHARMACY-PERSONNEL

36- MORTUARIUM PHARMACISTS AUTOPSY-ASSISTANTS

37- MANAGEMENT MANAGEMENT-PERSONNEL SERVICE-CHIEFS DIRECTORS-SECRETARIES

38- ADMINISTRATION STAFF-SECRETARIES GENERAL-ADMINISTRATION-PERSONNEL
RECEPTIONISTS ARCHIVES-PERSONNEL

39- ARCHIVES ARCHIVES-PERSONNEL

40- PERSONNEL-ACCOMMODATION
SURGERY-NURSES UROLOGY-NURSES ORTHOPEDY-NURSES
JAW-SURGERY-NURSES TNE-SURGERY-NURSES
EYE-SURGERY-NURSES CARDIOLOGY-NURSES PULMONOLOGY-NURSES
INTERNAL-SURGERY-NURSES DERMATOLOGY-NURSES GYNAECOLOGY-NURSES
OBSTETRY-NURSES PEADIATRY-NURSES NEUROLOGY-NURSES PSYCHIATRY-NURSES
RADIOLOGY-NURSES SPECIAL-CARE-NURSES OPERATION-NURSES RECOVERY-NURSES
PATHO-PHYSIOLOGY-NURSES SURGERY-PATIENTS-HELPS UROLOGY-PATIENTS-HELPS
ORTHOPEDY-PATIENTS-HELPS JAW-SURGERY-PATIENTS-HELPS
TNE-SURGERY-PATIENTS-HELPS EYE-SURGERY-PATIENTS-HELPS
CARDIOLOGY-PATIENTS-HELPS PULMONOLOGY-PATIENTS-HELPS
INTERNAL-SURGERY-PATIENTS-HELPS DERMATOLOGY-PATIENTS-HELPS
GYNAECOLOGY-PATIENTS-HELPS OBSTETRY-PATIENTS-HELPS
PEADIATRY-PATIENTS-HELPS NEUROLOGY-PATIENTS-HELPS
PSYCHIATRY-PATIENTS-HELPS RADIOLOGY-PATIENTS-HELPS
SURGEONS UROLOGISTS ORTHOPAEDISTS JAW-SURGEONS TNE-SURGEONS EYE-SURGEONS
CARDIOLOGISTS PULMONOLOGISTS INTERNISTS DERMATOLOGISTS GYNAECOLOGISTS
OBSTETRISTS PEADIATRISTS NEUROLOGISTS PSYCHIATRISTS RADIOLOGISTS
PATHO-PHYSIOLOGISTS ANESTHESISTS BACTERIOLOGISTS
REHABILITATIONISTS PATHOLOGISTS PHARMACISTS CLINICAL-CHEMISTS
PHYSIOTHERAPISTS PRODUCERS OCCUPATIONAL-THERAPISTS-1
OCCUPATIONAL-THERAPISTS-2 PATHO-PHYSIO-ADMINISTRATORS
ADMISSION-ADMINISTRATION-PERSONNEL PHYSIO-ADMINISTRATORS RADIO-LABORANTS
SPECIAL-LABORANTS SOCIAL-WORKERS PHYSIO-PERSONNEL ROENTGEN-LABORANTS
PHARMACY-PERSONNEL LABORANTS LAB-PERSONNEL AUTOPSY-ASSISTANTS
STAFF-SECRETARIES MANAGEMENT-PERSONNEL
SERVICE-CHIEFS DIRECTORS-SECRETARIES
GENERAL-ADMINISTRATION-PERSONNEL RECEPTIONISTS KITCHEN-PERSONNEL
LINEN-PERSONNEL BEDS-PERSONNEL POLY-OR-PERSONNEL POLY-PERSONNEL
TECH-SERVICES-PERSONNEL TRANSPORT-PERSONNEL ENERGY-PERSONNEL
STORE-KEEPERS ARCHIVES-PERSONNEL CLERGYMAN
MENTAL-ATTENDANTS

41- CENTRAL-KITCHEN KITCHEN-PERSONNEL

42- LINEN-SERVICE LINEN-PERSONNEL

43- BEDS-SERVICE BEDS-PERSONNEL

44- DOMESTIC-SERVICE TRANSPORT-PERSONNEL STORE-KEEPERS

45- STORES	TRANSPORT-PERSONNEL STORE-KEEPERS
46- TECH-SERVICE	TRANSPORT-PERSONNEL TECH-SERVICES-PERSONNEL
47- TECH-INSTALLATION	TECH-SERVICES-PERSONNEL
48- ENERGY	TECH-SERVICES-PERSONNEL ENERGY-PERSONNEL

Furthermore, the following weights are assumed to be associated with the actors.

Actor	Weights
ARCHIVES-PERSONNEL	2
AUTOPSY-ASSISTANTS	1
ANESTHESISTS	4
BEDS-PERSONNEL	3
BACTERIOLOGISTS	1
CARDIOLOGISTS	2
CARDIOLOGY-PATIENTS-HELPS	2
CARDIOLOGY-NURSES	12
CARDIOLOGY-PATIENTS	36
CLINICAL-CHEMISTS	2
CLERGYMAN	1
DIRECTORS-SECRETARIES	3
DERMATOLOGISTS	1
DERMATOLOGY-PATIENTS-HELPS	1
DERMATOLOGY-NURSES	4
DERMATOLOGY-PATIENTS	12
EYE-SURGEONS	2
EYE-SURGERY-NURSES	3
EYE-SURGERY-PATIENTS	8
EYE-SURGERY-PATIENTS-HELP	1
ENERGY-PERSONNEL	2
LINEN-PERSONNEL	10
LAB-PERSONNEL	10
LABORANTS	40
PHARMACISTS	1
PATHO-PHYSIOLOGY-NURSES	12
PULMONOLOGY-NURSES	8
PHARMACY-PERSONNEL	8
PULMONOLOGY-PATIENTS-HELPS	1
INTERNAL-SURGERY-PATIENTS-HELPS	5
JAW-SURGERY-PATIENTS-HELPS	1
JAW-SURGEONS	1
JAW-SURGERY-NURSES	4
JAW-SURGERY-PATIENTS	12
RECEPTIONISTS	3
GENERAL-ADMINISTRATION-PERSONNEL	40
MANAGEMENT-PERSONNEL	3
SPECIAL-LABORANTS	38
STAFF-SECRETARIES	6
PHYSIO-PERSONNEL	15
SOCIAL-WORKERS	1
ADMISSION-ADMINISTRATION-PERSONNEL	5
REHABILITATIONISTS	1
GYNAECOLOGISTS	3
GYNAECOLOGY-PATIENTS-HELPS	2
GYNAECOLOGY-NURSES	12
GYNAECOLOGY-PATIENTS	36
RECOVERY-NURSES	5
OPERATION-NURSES	20
OCCUPATIONAL-THERAPISTS-2	3
OCCUPATIONAL-THERAPISTS-1	3
ROENTGEN-LABORANTS	20
RADIO-LABORANTS	7

RADIOLOGISTS	3
RADIOLOGY-PATIENTS-HELPS	2
RADIOLOGY-NURSES	8
RADIOLOGY-PATIENTS	25
KITCHEN-PERSONNEL	15
NEUROLOGISTS	2
NEUROLOGY-PATIENTS-HELPS	2
NEUROLOGY-NURSES	15
NEUROLOGY-PATIENTS	31
PSYCHIATRISTS	2
PSYCHIATRY-PATIENTS-HELPS	2
PSYCHIATRY-NURSES	10
PSYCHIATRY-PATIENTS	28
PULMONOLOGISTS	2
PULMONOLOGY-PATIENTS	24
PATHO-PHYSIOLOGISTS	5
PATHO-PHYSIO-ADMINISTRATORS	4
PHYSIOTHERAPISTS	2
PHYSIO-ADMINISTRATORS	2
PATHOLOGISTS	2
POLY-PERSONNEL	50
POLY-OR-PERSONNEL	20
PRODUCERS	1
PEADIATRISTS	2
PEADIATRY-PATIENTS-HELPS	4
PEADIATRY-NURSES	20
PEADIATRY-PATIENTS	65
OBSTETRY-PATIENTS-HELPS	2
OBSTETRY-NURSES	10
OBSTETRY-PATIENTS	24
OBSTETRISTS	1
INTERNAL-SURGERY-NURSES	38
INTERNAL-SURGERY-PATIENTS	114
ORTHOPAEDISTS	2
ORTHOPEDY-PATIENTS-HELPS	2
ORTHOPEDY-NURSES	12
ORTHOPEDY-PATIENTS	36
UROLOGISTS	2
UROLOGY-PATIENTS-HELPS	2
UROLOGY-NURSES	10
UROLOGY-PATIENTS	30
VISITORS	40
MENTAL-ATTENDANTS	2
INTERNISTS	5
SURGEONS	6
SURGERY-PATIENTS-HELPS	7
SURGERY-NURSES	40
SURGERY-PATIENTS	120
STORE-KEEPERS	2
SERVICE-CHIEFS	5
SPECIAL-CARE-NURSES	9
SPECIAL-CARE-PATIENTS	20
TRANSPORT-PERSONNEL	5
TECH-SERVICES-PERSONNEL	15
TNE-SURGEONS	2
TNE-SURGERY-NURSES	8
TNE-SURGERY-PATIENTS	24
TNE-SURGERY-PATIENTS-HELPS	1

6.2 Partially hierarchical clusters of locations and streams of locations

Automatic analysis of above data-sets by TOPGENE yields the following partially hierarchical clusters of location-pairs, and streams of locations with respect to the community and the privacy / circulation-cost norm(s). The streams of locations serve for generating prototypical designs. The previous order of data-sets are kept intact in this section.

The partially hierarchical clusters of location-pairs for the house:

Association degree	Cluster of location-pairs
97	(LIVING-ROOM TOILET)
70	(BATH-ROOM TOILET)
56	(LIVING-ROOM TV-ROOM)
35	(TOILET TV-ROOM)
14	(BATH-ROOM LIVING-ROOM) (BATH-ROOM BEDROOM-3)
10	(KITCHEN-1 LIVING-ROOM) (KITCHEN-1 KITCHEN-2) (BATH-ROOM BEDROOM-2) (BATH-ROOM BEDROOM-1)
7	(BATH-ROOM TV-ROOM)
6	(ROOM STUDY-ROOM) (LIVING-ROOM BEDROOM-2) (LIVING-ROOM BEDROOM-1) (KITCHEN-1 TOILET)
5	(BATH-ROOM KITCHEN-1)
4	(BEDROOM-3 LIVING-ROOM) (KITCHEN-1 TV-ROOM) (BEDROOM-3 TOILET) (BEDROOM-2 TOILET) (BEDROOM-1 TOILET) (STUDY-ROOM TV-ROOM) (BEDROOM-3 TV-ROOM) (BEDROOM-2 TV-ROOM) (BEDROOM-1 TV-ROOM)
2	(BEDROOM-4 LIVING-ROOM) (BEDROOM-4 TOILET) (BEDROOM-4 TV-ROOM) (BEDROOM-3 KITCHEN-1) (STUDY-ROOM TOILET) (BEDROOM-3 STUDY-ROOM)
1	(KITCHEN-2 LIVING-ROOM) (BATH-ROOM BEDROOM-4) (KITCHEN-2 TOILET) (KITCHEN-2 TV-ROOM) (BATH-ROOM KITCHEN-2) (BEDROOM-3 KITCHEN-2) (BATH-ROOM STUDY-ROOM)
0	(HALL ...)

The streams of locations generated by TOPGENE for the community and privacy / circulation-cost norm(s) based on above partially hierarchical clusters of location are shown below.

The community stream of locations for the house data-set:

(LIVING-ROOM BATH-ROOM KITCHEN-2 HALL * BEDROOM-4 STUDY-ROOM BEDROOM-1 BEDROOM-2 KITCHEN-1 BEDROOM-3 TV-ROOM TOILET)

The privacy stream of locations for the house:

(BEDROOM-4 KITCHEN-2 KITCHEN-1 TV-ROOM LIVING-ROOM * TOILET BATH-ROOM BEDROOM-3 STUDY-ROOM BEDROOM-2 BEDROOM-1 HALL)

The partially hierarchical clusters of locations for first-floor of the police-station:

Association degree	Clusters of location-pairs
6	(PERSONNEL-TOILET RESTAURANT)
4	(CONFERENCE-ROOM1 CONFERENCE-ROOM2)

	(CONFERENCE-ROOM2 RESTAURANT)
3	(CONFERENCE-ROOM2 STAFF-TOILET) (CONFERENCE-ROOM2 CHIEF-ROOM)
	(CONFERENCE-ROOM1 STAFF-TOILET) (CONFERENCE-ROOM1 CHIEF-ROOM)
	(RESTAURANT STAFF-TOILET) (CHIEF-ROOM RESTAURANT)
	(CHIEF-ROOM STAFF-TOILET)
2	(INSTRUCTION-ROOM1 INSTRUCTION-ROOM3)
	(INSTRUCTION-ROOM1 INSTRUCTION-ROOM2)
	(INSTRUCTION-ROOM3 RESTAURANT)
	(INSTRUCTION-ROOM2 RESTAURANT)
	(INSTRUCTION-ROOM1 RESTAURANT)
1	(KITCHEN PERSONNEL-TOILET) (KITCHEN RESTAURANT)
	(OFFICE PERSONNEL-TOILET) (INSTRUCTION-ROOM3 PERSONNEL-TOILET)
	(INSTRUCTION-ROOM2 PERSONNEL-TOILET)
	(INSTRUCTION-ROOM1 PERSONNEL-TOILET)
	(CONFERENCE-ROOM2 PERSONNEL-TOILET)
	(CONFERENCE-ROOM1 PERSONNEL-TOILET)
	(CONFERENCE-ROOM2 OFFICES) (CONFERENCE-ROOM1 OFFICES)
	(OFFICES RESTAURANT) (ARCHIVE CONFERENCE-ROOM2)
	(CONFERENCE-ROOM2 SECRETARY-ROOM)
	(ARCHIVE CONFERENCE-ROOM1) (CONFERENCE-ROOM1 SECRETARY-ROOM)
	(ARCHIVE RESTAURANT) (RESTAURANT SECRETARY-ROOM)
	(ARCHIVE STAFF-TOILET)
	(SECRETARY-ROOM STAFF-TOILET) (ARCHIVE SECRETARY-ROOM)
	(ARCHIVE CHIEF-ROOM) (CHIEF-ROOM SECRETARY-ROOM)
0	(CORRIDOR ...)

The community stream of locations for first-floor of the police-station:

(PERSONNEL-TOILET CONFERENCE-ROOM2 INSTRUCTION-ROOM3 CORRIDOR * SECRETARY-ROOM ARCHIVE OFFICES KITCHEN INSTRUCTION-ROOM1 INSTRUCTION-ROOM2 CHIEF-ROOM STAFF-TOILET CONFERENCE-ROOM1 RESTAURANT)

The privacy stream of locations for first-floor of the police-station:

(OFFICES INSTRUCTION-ROOM1 INSTRUCTION-ROOM2 INSTRUCTION-ROOM3 KITCHEN-PERSONNEL-TOILET * RESTAURANT CONFERENCE-ROOM2 CONFERENCE-ROOM1 STAFF-TOILET CHIEF-ROOM ARCHIVE SECRETARY-ROOM CORRIDOR)

The partially hierarchical clusters of locations for second-floor of the police-station:

Association degree	Clusters of location-pairs
3	(BICYCLE-SHED PERSONNEL-ENTRANCE)
	(BRIEFING-ROOM PERSONNEL-ENTRANCE)
2	(PERSONNEL-ENTRANCEPRISONERS-ENTRANCE)(CELLSPRISONERS-ENTRANCE)
	(GARAGE PERSONNEL-ENTRANCE)(PERSONNEL-ENTRANCEPROPERTY-INTAKE)
	(PERSONNEL-ENTRANCE PROPERTY-STORAGE)
	(DRESSING-ROOM3 PERSONNEL-ENTRANCE)
	(PERSONNEL-ENTRANCE REPORT-ROOM1) (RECEPTION1 RECEPTION2)
	(BRIEFING-ROOM GARAGE) (PROPERTY-INTAKE PROPERTY-STORAGE)
1	(BICYCLE-SHED PRISONERS-ENTRANCE) (GARAGE PRISONERS-ENTRANCE)
	(PRISONERS-ENTRANCE PROPERTY-INTAKE)
	(PRISONERS-ENTRANCE PROPERTY-STORAGE)
	(BRIEFING-ROOM PRISONERS-ENTRANCE)
	(DRESSING-ROOM2 PRISONERS-ENTRANCE)
	(DRESSING-ROOM1 PRISONERS-ENTRANCE)
	(PERSONNEL-ENTRANCE VISITORS-ENTRANCE)
	(RECEPTION2 VISITORS-ENTRANCE) (RECEPTION1 VISITORS-ENTRANCE)
	(BICYCLE-SHED VISITORS-ENTRANCE) (VISITORS-ENTRANCE VISITORS-TOILET)

	(PORTERS-LODGE VISITORS-ENTRANCE) (PERSONNEL-ENTRANCE RECEPTION2) (PERSONNEL-ENTRANCE RECEPTION1) (DRESSING-ROOM2 PERSONNEL-ENTRANCE) (DRESSING-ROOM1 PERSONNEL-ENTRANCE) (PERSONNEL-ENTRANCE REPORT-ROOM2) (PERSONNEL-ENTRANCE PORTERS-LODGE) (CELLS PERSONNEL-ENTRANCE) (BICYCLE-SHED RECEPTION2) (RECEPTION2 VISITORS-TOILET) (DRESSING-ROOM3 RECEPTION2) (RECEPTION2 REPORT-ROOM2) (RECEPTION2 REPORT-ROOM1) (BICYCLE-SHED RECEPTION1) (RECEPTION1 VISITORS-TOILET) (DRESSING-ROOM3 RECEPTION1) (RECEPTION1 REPORT-ROOM2) (RECEPTION1 REPORT-ROOM1) (BICYCLE-SHED GARAGE) (BICYCLE-SHED BRIEFING-ROOM) (BICYCLE-SHED DRESSING-ROOM3) (BICYCLE-SHED DRESSING-ROOM2) (BICYCLE-SHED DRESSING-ROOM1) (BICYCLE-SHED REPORT-ROOM) (BICYCLE-SHED REPORT-ROOM1) (BICYCLE-SHED PORTERS-LODGE) (DRESSING-ROOM2 GARAGE) (DRESSING-ROOM1 GARAGE) (GARAGE REPORT-ROOM1) (CELLS PROPERTY-INTAKE) (CELLS PROPERTY-STORAGE) (BRIEFING-ROOM DRESSING-ROOM3) (BRIEFING-ROOM DRESSING-ROOM2) (BRIEFING-ROOM DRESSING-ROOM1) (BRIEFING-ROOM REPORT-ROOM1) (DRESSING-ROOM3 REPORT-ROOM2) (DRESSING-ROOM3 REPORT-ROOM1) (DRESSING-ROOM2 DRESSING-ROOM1) (REPORT-ROOM1 REPORT-ROOM2)
0	(CELLS CORRIDOR)

The community stream of locations for second-floor of the police-station:

(BICYCLE-SHED BRIEFING-ROOM PRISONERS-ENTRANCE GARAGE PROPERTY-INTAKE PROPERTY-STORAGE DRESSING-ROOM-3 REPORT-ROOM1 RECEPTION-2 VISITORS-ENTRANCE REPORT-ROOM2 CORRIDOR * PORTERS-LODGE VISITORS-TOILET DRESSING-ROOM1 DRESSING-ROOM2 RECEPTION1 CELLS PERSONNEL-ENTRANCE)

The privacy stream of locations for second-floor of the police-station:

(CORRIDOR REPORT-ROOM2 REPORT-ROOM1 PROPERTY-STORAGE PROPERTY-INTAKE CELLS PRISONERS-ENTRANCE BICYCLE-SHED * PERSONNEL-ENTRANCE BRIEFING-ROOM GARAGE DRESSING-ROOM2 DRESSING-ROOM1 DRESSING-ROOM3 RECEPTION2 RECEPTION1 VISITORS-ENTRANCE VISITORS-TOILET PORTERS-LODGE)

The partially hierarchical clusters of locations for the basement of the hospital is shown below. Few relations exist between locations in the basement of the hospital. For this reason, most of the location-pairs in this floor fall into the cluster with 0 degree of association.

Association degree	**Clusters of location-pairs**
5	(DOMESTIC-SERVICE TECH-SERVICE)
0	All other location-pairs.

The community stream of locations for basement of the hospital:

(DOMESTIC-SERVICE CORRIDORS BEDS-SERVICES ARCHIVES PATHO-PHYSIOLOGY * MORTUARIUM LINEN-SERVICE CORRIDORS TECH-SERVICE)

The privacy / circulation-cost stream of locations for the basement of the hospital is as follows. This stream is divided in two sub-streams. A sub-stream consisting of two activities: domestic-service and tech-service, and another sub-stream embodying other activities in this floor. This shows that

two activities domestic-service and tech-service are completely isolated from other activities, and there is no association between these two activities and other activities in this floor.

(DOMESTIC-SERVICE * TECH-SERVICE) (PATHO-PHYSIOLOGY MORTUARIUM ARCHIVES LINEN-SERVICE BEDS-SERVICES CORRIDORS * CORRIDOR-10)

The partially hierarchical clusters of locations for the ground-floor of the hospital:

Association degree	Cluster of location-pairs
65	(ADMISSION PEADIATRY)
56	(NEUROLOGYPSYCHIATRY)(PEADIATRYPSYCHIATRY)(OBSTETRYPSYCHIATRY) (NEUROLOGY PEADIATRY) (NEUROLOGY OBSTETRY) (OBSTETRY PEADIATRY)
31	(ADMISSION NEUROLOGY)
28	(ADMISSION PSYCHIATRY)
24	(ADMISSION OBSTETRY)
22	(POLYCLINIC RONTGENDIAGNOSTICS)
16	(POLYCLINIC POLYCLINIC-OPERATION)
13	(POLYCLINIC PSYCHIATRY) (NEUROLOGY POLYCLINIC) (PEADIATRY POLYCLINIC) (PEADIATRY RONTGENDIAGNOSTICS)
12	(OBSTETRY RONTGENDIAGNOSTICS)
11	(OBSTETRY POLYCLINIC) (PSYCHIATRY RONTGENDIAGNOSTICS) (NEUROLOGY RONTGENDIAGNOSTICS)
7	(DELIVERY-ROOM POLYCLINIC) (DELIVERY-ROOM POLYCLINIC-OPERATION) (POLYCLINIC-OPERATION RONTGENDIAGNOSTICS)
6	(POLYCLINIC-OPERATIONPSYCHIATRY) (NEUROLOGYPOLYCLINIC-OPERATION) (PEADIATRY POLYCLINIC-OPERATION) (OBSTETRY POLYCLINIC-OPERATION)
5	(ADMISSION PHYSICAL-THERAPY)
4	(PHYSICAL-THERAPY PSYCHIATRY)
2	(PHYSICAL-THERAPY POLYCLINIC) (NEUROLOGY PHYSICAL-THERAPY) (PEADIATRY PHYSICAL-THERAPY) (OBSTETRY PHYSICAL-THERAPY)
0	(ENERGY STORES) (CENTRAL-KITCHEN STORES) (CENTRAL-KITCHEN CORRIDOR21) All other location-pairs.

The community stream of locations for the ground-floor of hospital:

(ADMISSION PSYCHIATRY RONTGENDIAGNOSTICS POLYCLINIC-OPERATION DELIVERY-ROOM ENERGY CENTRAL-KITCHEN CORRIDORS * CORRIDORS STORES PHYSICAL-THERAPY POLYCLINIC OBSTETRY NEUROLOGY PEADIATRY)

The privacy stream of locations (activities) for the ground-floor of hospital also is divided into two sub-streams. This is again due to lack of association between any two locations in the sub-streams.

(PHYSICAL-THERAPY ADMISSION * PEADIATRY PSYCHIATRY NEUROLOGY OBSTETRY RONTGENDIAGNOSTICS POLYCLINIC POLYCLINIC-OPERATION DELIVERY-ROOM) (CORRIDORS CENTRAL-KITCHEN ENERGY * STORES)

The partially hierarchical clusters of location-pairs for first-floor of the hospital:

Association degree	Cluster of location-pairs
66	(NIGHT-SERVICE PERSONNEL-ACCOMMODATION)
55	(CLINICAL-LAB PERSONNEL-ACCOMMODATION)
53	(OPERATING-ROOM PERSONNEL-ACCOMMODATION)

51	(ADMINISTRATION PERSONNEL-ACCOMMODATION)
38	(PERSONNEL-ACCOMMODATION RECOVERY)
33	(OPERATING-ROOM RECOVERY)
25	(PERSONNEL-ACCOMMODATION SPECIAL-CARE)
15	(PERSONNEL-ACCOMMODATION TECH-INSTALLATIONS)
	(NIGHT-SERVICE TECH-INSTALLATIONS)
	(PERSONNEL-ACCOMMODATION VISITORS-ACCOMMODATION)
11	(MANAGEMENT PERSONNEL-ACCOMMODATION)
9	(PERSONNEL-ACCOMMODATION STERILIZATION)
	(PERSONNEL-ACCOMMODATION PHARMACY)
	(PHARMACY STERILIZATION)
7	(PERSONNEL-ACCOMMODATION RADIO-ISOTOPE-LAB)
	(NIGHT-SERVICE RADIO-ISOTOPE-LAB)
6	(PERSONNEL-ACCOMMODATION STAFF-ACCOMMODATION)
	(ADMINISTRATION STAFF-ACCOMMODATION)
	(NIGHT-SERVICE RECOVERY) (NIGHT-SERVICE OPERATING-ROOM)
	(NIGHT-SERVICE SPECIAL-CARE) (RECOVERY SPECIAL-CARE)
	(OPERATING-ROOM SPECIAL-CARE)
4	(PERSONNEL-ACCOMMODATION SPECIAL-LAB)
	(PERSONNEL-ACCOMMODATION SPIRITUAL-MENTAL-CARE)
3	(CLINICAL-LAB SPECIAL-LAB) (SPECIAL-CARE SPIRITUAL-MENTAL-CARE)
0	Corridor pairs.

The community stream of locations for first-floor of the hospital:

(NIGHT-SERVICE CLINICAL-LAB OPERATING-ROOM ADMINISTRATION RECOVERY SPECIAL-CARE TECH-INSTALLATIONS VISITORS-ACCOMMODATION MANAGEMENT STERILIZATION PHARMACY RADIO-ISOTOPE-LAB SPECIAL-LAB SPIRITUAL-MENTAL-CARE CORRIDORS * CORRIDORS PERSONNEL-ACCOMMODATION)

The privacy stream of locations for first-floor of the hospital:

(RADIO-ISOTOPE-LAB MANAGEMENT VISITORS-ACCOMMODATION STAFF-ACCOMMODATION ADMINISTRATION TECH-INSTALLATION NIGHT-SERVICES * PERSONNEL-ACCOMMODATION CLINICAL-LAB SPECIAL-LAB OPERATING-ROOM RECOVERY SPECIAL-CARE SPIRITUAL-MENTAL-CARE STERILIZATION PHARMACY) (CORRIDORS * CORRIDORS)

The partially hierarchical clusters of location-pairs for second-floor of the hospital:

Association degree	Clusters of location-pairs
70	(EYE-SURGERY PULMONOLOGY)
56	(DERMATOLOGY GYNAECOLOGY) (GYNAECOLOGY INTERNAL-SURGERY)
	(GYNAECOLOGY PULMONOLOGY) (CARDIOLOGY GYNAECOLOGY)
	(EYE-SURGERY GYNAECOLOGY) (GYNAECOLOGY TNE-SURGERY)
	(GYNAECOLOGY JAW-SURGERY) (GYNAECOLOGY ORTHOPEDY)
	(GYNAECOLOGY UROLOGY)(GYNAECOLOGY SURGERY)
	(DERMATOLOGY INTERNAL-SURGERY)
	(DERMATOLOGY PULMONOLOGY) (CARDIOLOGY DERMATOLOGY)
	(DERMATOLOGY EYE-SURGERY)(DERMATOLOGY TNE-SURGERY)
	(DERMATOLOGY JAW-SURGERY) (DERMATOLOGY ORTHOPEDY)
	(DERMATOLOGY UROLOGY) (DERMATOLOGY SURGERY)
	(INTERNAL-SURGERY PULMONOLOGY)
	(CARDIOLOGY INTERNAL-SURGERY) (EYE-SURGERY INTERNAL-SURGERY)
	(INTERNAL-SURGERY TNE-SURGERY) (INTERNAL-SURGERY JAW-SURGERY)
	(INTERNAL-SURGERY ORTHOPEDY) (INTERNAL-SURGERY UROLOGY)
	(INTERNAL-SURGERY SURGERY)(CARDIOLOGY PULMONOLOGY)
	(PULMONOLOGY TNE-SURGERY) (JAW-SURGERY PULMONOLOGY)
	(ORTHOPEDY PULMONOLOGY) (PULMONOLOGY UROLOGY)
	(PULMONOLOGY TNE-SURGERY) (CARDIOLOGY EYE-SURGERY)
	(CARDIOLOGY TNE-SURGERY) (CARDIOLOGY JAW-SURGERY)

(CARDIOLOGY ORTHOPEDY) (CARDIOLOGY UROLOGY)
(CARDIOLOGY SURGERY) (EYE-SURGERY TNE-SURGERY)
(EYE-SURGERY JAW-SURGERY)
(EYE-SURGERY ORTHOPEDY) (EYE-SURGERY UROLOGY) (EYE-SURGERY SURGERY)
(JAW-SURGERY TNE-SURGERY) (ORTHOPEDY TNE-SURGERY)
(TNE-SURGERY UROLOGY) (SURGERY TNE-SURGERY)
(JAW-SURGERY ORTHOPEDY) (JAW-SURGERY UROLOGY)
(JAW-SURGERY SURGERY) (ORTHOPEDY UROLOGY) (ORTHOPEDY SURGERY)
(SURGERY UROLOGY)

53 (GYNAECOLOGY RADIOLOGY) (DERMATOLOGY RADIOLOGY)
(INTERNAL-SURGERY RADIOLOGY) (PULMONOLOGY RADIOLOGY)
(CARDIOLOGY RADIOLOGY) (EYE-SURGERY RADIOLOGY)
(RADIOLOGY TNE-SURGERY) (JAW-SURGERY RADIOLOGY)
(ORTHOPEDY RADIOLOGY)
(RADIOLOGY UROLOGY) (RADIOLOGY SURGERY)

0 Corridor pairs.

The community stream of locations for second-floor of the hospital:

(EYE-SURGERY GYNAECOLOGY CORRIDORS * CORRIDORS RADIOLOGY SURGERY UROLOGY ORTHOPEDY JAW-SURGERY TNE-SURGERY CARDIOLOGY INTERNAL-SURGERY DERMATOLOGY PULMONOLOGY)

The privacy / circulation-cost stream of locations for second-floor of the hospital:

(RADIOLOGY SURGERY UROLOGY ORTHOPEDY JAW-SURGERY TNE-SURGERY CARDIOLOGY INTERNAL-SURGERY DERMATOLOGY GYNAECOLOGY EYE-SURGERY * PULMONOLOGY) (CORRIDORS * CORRIDORS)

Examples of the designs generated by TOPGENE are given in the following four sections. The first three sections present samples of sub-optimal designs with respect to a norm. The fourth section is devoted to samples of designs produced with respect to combination of norms. A performance analysis of complete sets of design types produced for data-sets are presented at the end of each section. The last section of this chapter presents examples of design evaluation by TOPGENE.

6.3 Designs with respect to the community norm

The first type of designs generated with respect to the community norm is the linear-tree type designs. This type of design also is called the near-optimal and tree type design with respect to the community norm. Two types of linear-tree designs, namely linear-tree without auxiliary locations and linear-tree with auxiliary locations, are distinguished by TOPGENE. Addition of auxiliary locations to a linear-tree type design stretches it, and consequently stimulates the formation of new groups in the auxiliary locations, which in turn increase its performance with respect to the community norm. Examples of this design type with their corresponding behavioral analysis for the house problem are displayed below.

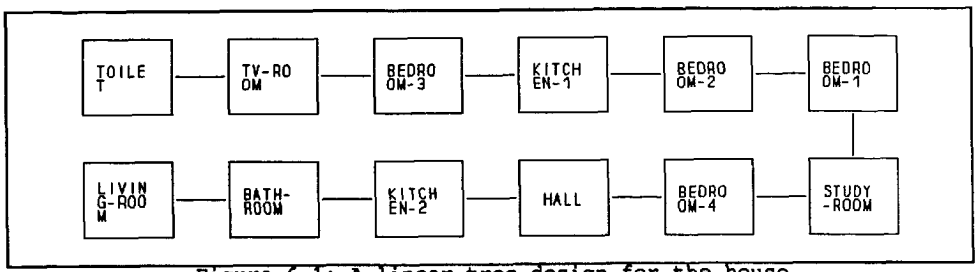


Figure 6.1: A linear-tree design for the house

Result of the **behavioral analysis** of the generated design:
Community utility = 2785 units.
Privacy-cost = 2466 units.
Circulation-cost = 3621 units.

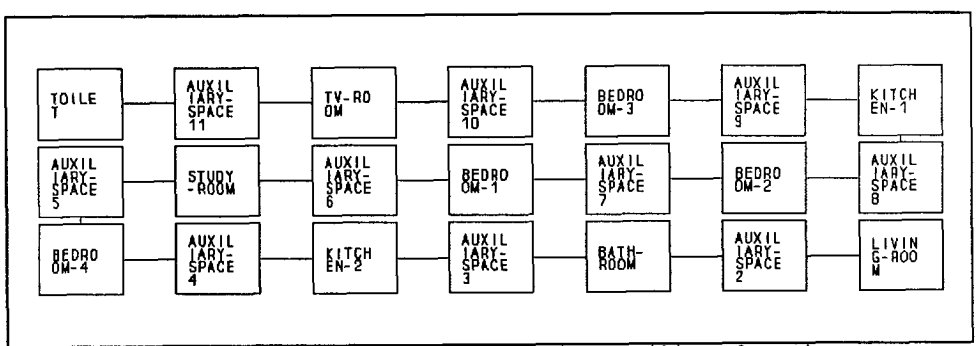


Figure 6.2: A linear-tree design with auxiliary-locations for the house

Result of the **behavioral analysis** of the generated design:
Community utility = 5350 units.
Privacy-cost = 2466 units.
Circulation-cost = 6186 units.

Examples follow of other prototypical design types with respect to the community norm for our data-sets. A few other design examples with respect to the community norm also will be exhibited in the final section of this chapter. These designs are used as yardsticks for evaluating existing designs.

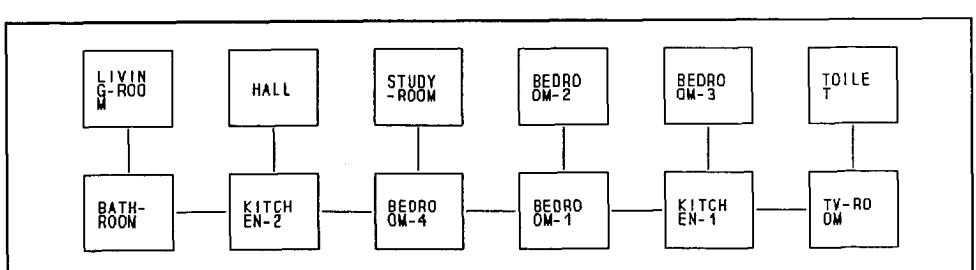


Figure 6.3: A single-loaded design for the house

Result of the **behavioral analysis** of the generated design:
Community utility = 1611 units.
Privacy-cost = 1611 units.
Circulation-cost = 2447 units.

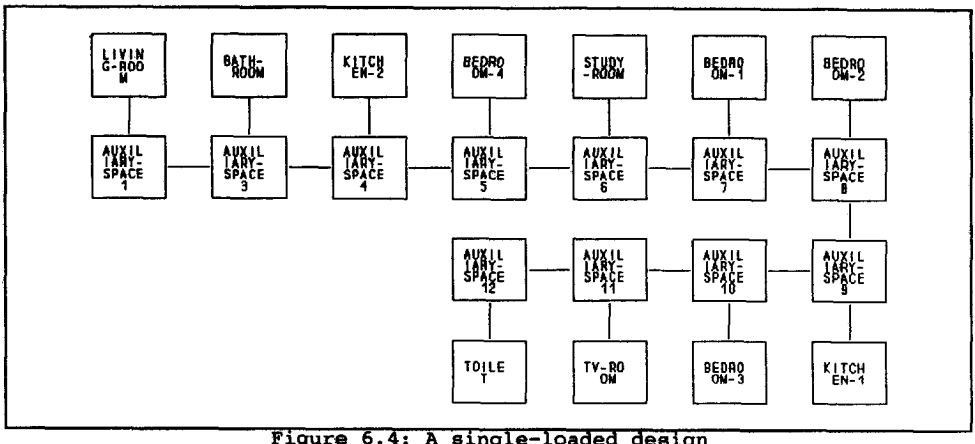


Figure 6.4: A single-loaded design
with auxiliary locations for the house

Result of the **behavioral analysis** of the generated design:
Community utility = 3302 units.
Privacy-cost = 0 units.
Circulation-cost = 4138 units.

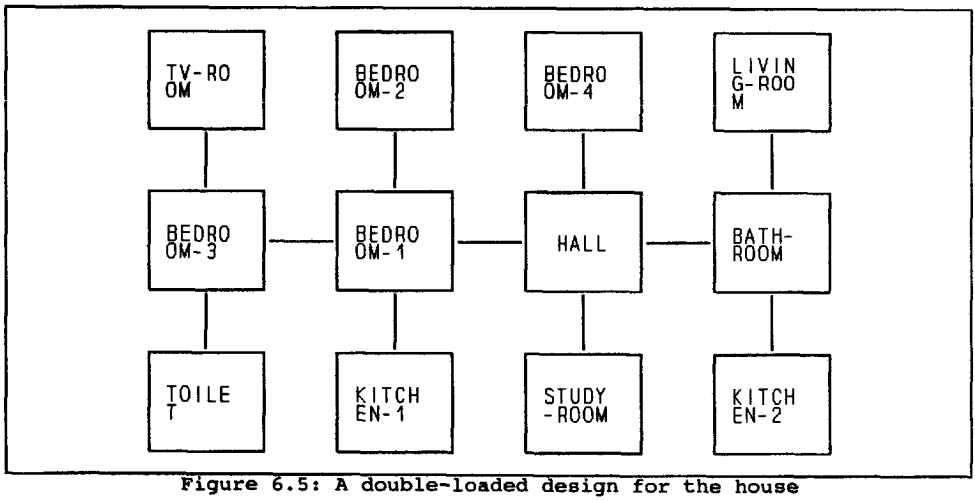


Figure 6.5: A double-loaded design for the house

Result of the **behavioral analysis** of the design:
Community Utility = 1162 Units.
Privacy-cost = 831 units.
Circulation-cost = 1998 units.

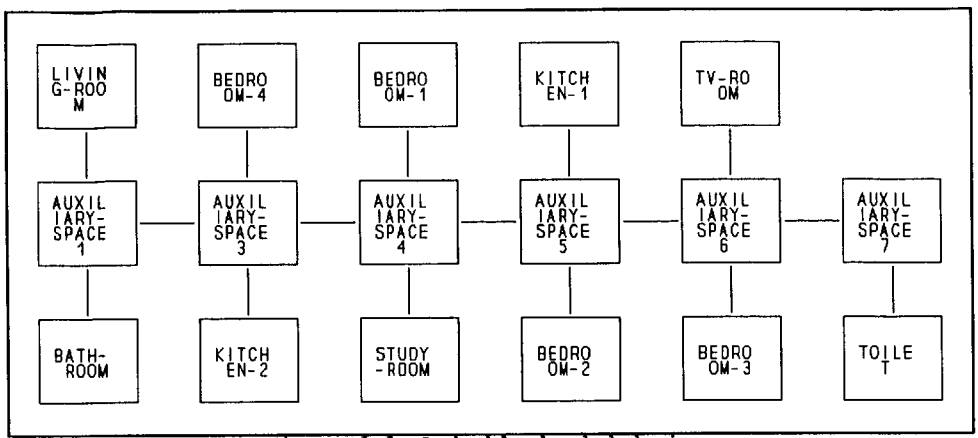


Figure 6.6: A double-loaded design
with auxiliary locations for the house

Result of the behavioral analysis of the generated design:
Community utility = 1877 units.
Privacy-cost = 0 units.
Circulation-cost = 2713 units.

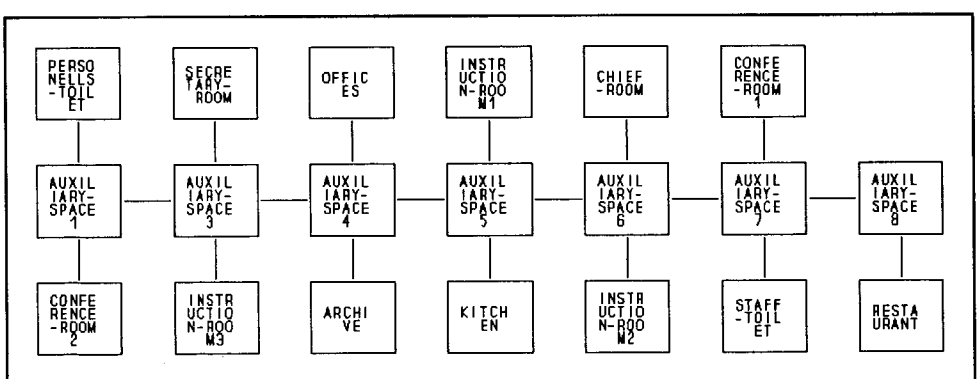


Figure 6.7: A double loaded type design with auxiliary
locations for 1st-floor of the police-station

Result of the behavioral analysis of the generated design:
Community utility = 193 units.
Privacy-cost = 0 units.
Circulation-cost = 339 units.

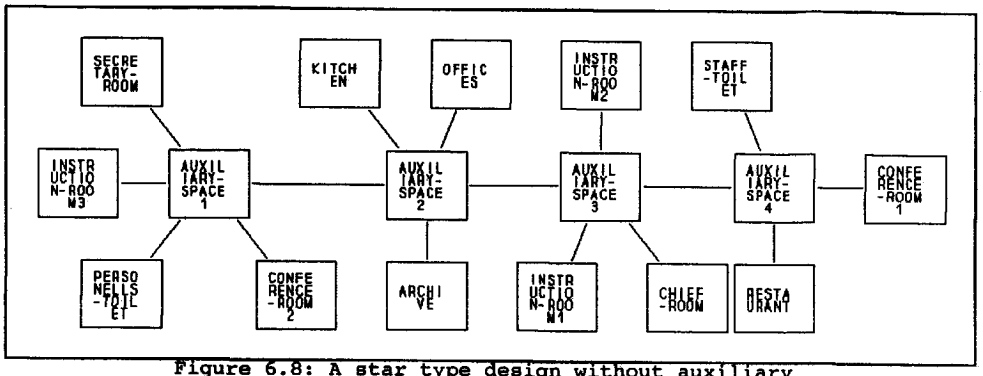


Figure 6.8: A star type design without auxiliary locations for 1st-floor of the police-station

Result of the **behavioral analysis** of the generated design:
Community utility = 120 units.
Privacy-cost = 120 units.
Circulation-cost = 266 units.

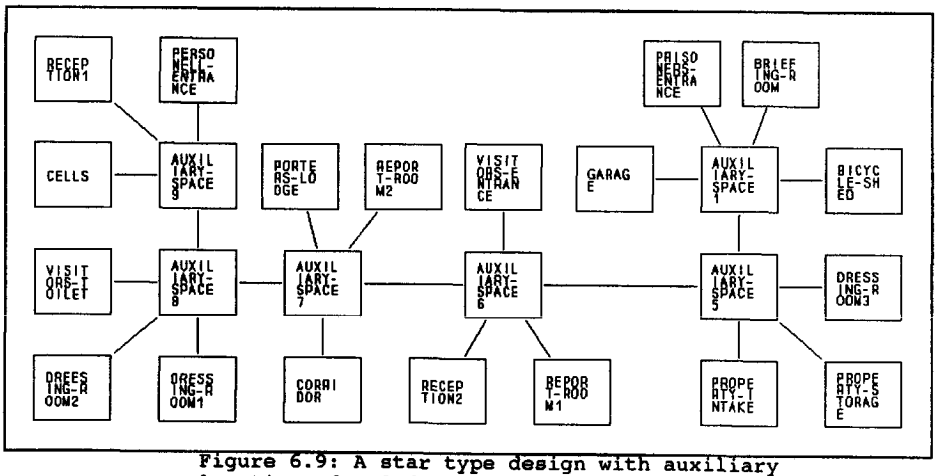


Figure 6.9: A star type design with auxiliary locations for 2nd-floor of the police-station

Result of the **behavioral analysis** of the generated design:
Community utility = 274 units.
Privacy-cost = 0 units.
Circulation-cost = 428 units.

Several grid type designs may be generated for the same set of data by varying dimension of the grids. TOPGENE permits users to decide the length and width of a grid type design. The labelling of a grid type design is carried out based on the order of locations (or activities) of the design in the generated community stream of locations. The locations in a community stream are arranged in a way that the highest interactive activities are as far as

possible. This arrangement of locations provides the operation of the design. Obviously, there are many choices for labelling of the grid based on the stream of locations. In addition the stream of locations must be broken into several sub-streams depending on the dimension of the grid. Any choice of allocation has some drawback in terms of disturbing the operational semantic that a stream carries. Heuristic rules reveal that labelling of locations from the same directions, as depicted in the following figure is more favourable than other strategies. A pair-wise comparison of labelled activities provide an insight into the matter. Such an comparison reveals that this strategy potentially keeps the most interactive location-pairs more apart than, for example, a bi-directional labelling. A grid type design, depending on the number of locations in a design, may be incomplete in the last row.

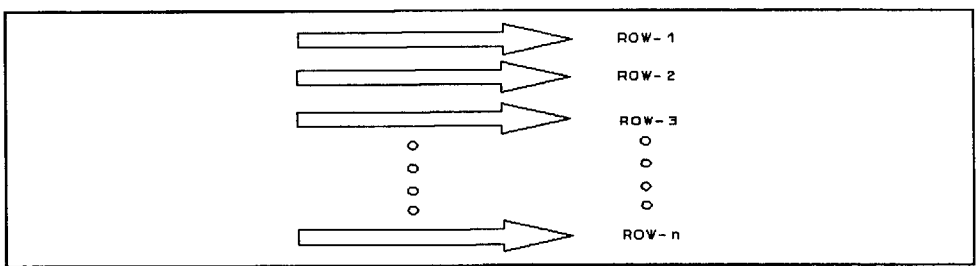


Figure 6.10: Labelling strategy for a grid type design with a community stream of locations (activities)

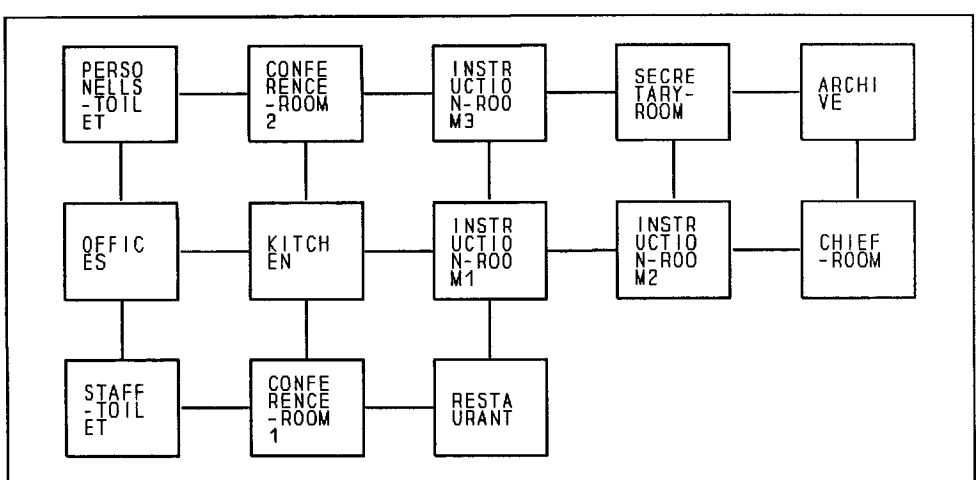


Figure 6.11: A 3 by 4 grid type design for the 1st-floor of police-station

Result of the **behavioral analysis** of the generated design:
Community utility = 1877 units.
Privacy-cost = 0 units.
Circulation-cost = 2713 units.

I skip exhibition of all possible designs with respect to the community norm. Yet, the result of behavioral analysis of a complete set of design types for the house and the police-station problems are presented in the following tables.

Results of the behavioral analysis of different designs for the house, generated by TOPGENE with respect to the community norm:

design type	Community utility	Privacy cost	Circulation cost
Linear-tree	2785	2466	3621
Linear-tree*	5350	2466	6186
Single loaded	1611	1611	24^Xx
Single-loaded*	3302	0	4138
Double-loaded	1162	831	1998
Double-loaded*	1877	0	2713
Star	781	781	1617
Star*	1269	0	2105
Grid (3x4)	974	948	1811

* Designs with auxiliary spaces.

Results of the behavioral analysis of designs for first-floor of the police-station, generated by TOPGENE with respect to the community norm:

design type	Community utility	Privacy cost	Circulation cost
Linear-tree	386	352	532
Linear-tree*	772	352	923
Single loaded	220	220	366
Single-loaded*	583	0	678
Double-loaded	181	138	327
Double-loaded*	294	0	440
Star	120	120	266
Star*	193	0	339
Grid (3*4)	120	120	266

* Designs with auxiliary spaces.

Results of the behavioral analysis of different designs for second-floor of the police-station, generated by the TOPGENE with respect to the community norm:

design type	Community utility	Privacy cost	Circulation cost
Linear-tree	572	572	726
Linear-tree*	1221	528	1375
Single loaded	309	309	463
Single-loaded*	726	0	880
Double-loaded	229	229	383
Double-loaded*	404	0	558
Star	194	194	348
Star*	274	0	428
Grid (3*6)	174	171	328

* Designs with auxiliary spaces.

6.3.1 An analysis of TOPGENE performance

Above tables present performance values of different designs for our data-sets. These designs were generated with respect to the community norm. However, as we can see in the tables, TOPGENE has provided us with the behavioral values of generated designs with respect to the privacy and circulation-cost norms as well. The reason is that TOPGENE tries to analyze a design with respect to all social norms, if sufficient information is available. For example, because of lack of information for intervening opportunity locations no behavioral analysis with respect to this norm is provided.

The results of behavioral analysis of different designs gathered in above tables helps in analyzing the behavior of TOPGENE. For example we are always interested in knowing the optimality of a design generated by TOPGENE with respect to a norm. This question is only answered if we had knowledge of the performance of the optimal design for each case. But, here, we do not have any knowledge of an optimal design with respect to a norm, unless such a design is generated and its performance behavior with respect to a norm is calculated. Finding an optimal design, as was discussed in chapter 2, is a hard task because of the intractability of our design problem. What we have in our hand is the neural network approach for generating the liner-tree type design, with respect to these norms. A comparison of TOPGENE and the neural network designs are given in chapter 8. These results, as we will see, show that TOPGENE heuristic approach in generating design with respect to the community norm is effective and yields designs that are in the neighborhood of an optimal design.

The main purpose of this section is to check the structure and behavior of the generated designs against each other, and analyze them to see whether they have a correct behavior relative to each other and in concordance with our expectation. The basic information available to us for this purpose is the results of the behavioral analysis of these designs in hard numerical form. The absence of this information, or if TOPGENE only could provide a qualitative analysis of these designs, then, without any doubt, judging the performance of TOPGENE was a difficult task.

We discard the analysis result of the designs with respect to the privacy, and the circulation-cost norms, since these norms have had no role in these designs. Based on the heuristic rules, discussed in previous chapter, we expect a high community performance for stretched design types, and low community performance as generated designs for the same set of data shrink towards more compact ones. This is of course only true if the operation of the design is kept almost intact. The operations of the buildings corresponding to our data are in fact defined by the arrangement of the locations (activities) in the community streams. The operation of all designs except a

liner-tree type design is changed in some degrees once it is labelled with the locations in a stream. Yet, these changes are very small with respect to the original arrangement of locations in the streams. This is evident from different performance behavior that we have for different designs. In fact, the connectivity structure of designs put certain constraint on the arrangements and ultimately effects the operation of the generated designs. The effect of this constraint is reflected in the above tables. These tables show that the linear-tree type designs for all cases are best designs with respect to the community norm. The next best design, in all cases, is the single loaded design with auxiliary locations. Looking at the designs presented in the last section, we can see that indeed this type of designs are the most stretched ones after the linear-tree type designs for a specific problem. If we continue analytical comparisons of community utilities of the generated designs for a specific problem, on the condition that two classes of designs with auxiliary locations and without auxiliary locations are analyzed separately, then we see that for all cases, and with no exception, the next best designs are double-loaded, star, and grid type respectively. The conclusion is that designs generated by TOPGENE with respect to the community norm are indeed in conformity with our expectations and intuitions. This supports the idea that the system has a well performance in generating designs with respect to the community norm.

The following sections surveys performance of TOPGENE in generating designs with respect to the privacy / circulation-cost norm(s).

6.4 Designs with respect to the privacy / circulation-cost norm(s)

The norms "privacy" and "circulation-cost" were characterised as two non-conflicting norms that have tendency towards highly branched and compact designs as near-optimal designs. The exception are the single-loaded, the double-loaded, and star types designs with auxiliary locations. These design types are perfect designs for the privacy norm. These design provide a zero privacy cost so may be considered as perfect designs with respect to this norm. If we disregard these three design types, then the labelling of a design with respect to these norms requires that most interactive location-pairs to have direct access with each other. Such strategy insures minimum disturbances on the locations of design as a result of its operation.

This section examines examples of designs generated by TOPGENE with respect to these norms. I will analyze these designs by inspecting their structure and operation, and comparing their performance analysis with respect to these norms. All examples of designs displayed in this section, or their behavioral values gathered in the tables exhibited at the end of this section are only generated with respect to the point(s) of view of the privacy /

circulation-cost. So, first the essential information for these designs are only the behavioral values corresponding to the privacy and circulation-cost norms. The performance values corresponding to the community norm are produced as supplementary information. Again, as in previous examples, no intervening opportunity requirements have been defined for these designs, and no intervening opportunity utility is calculated for these designs.

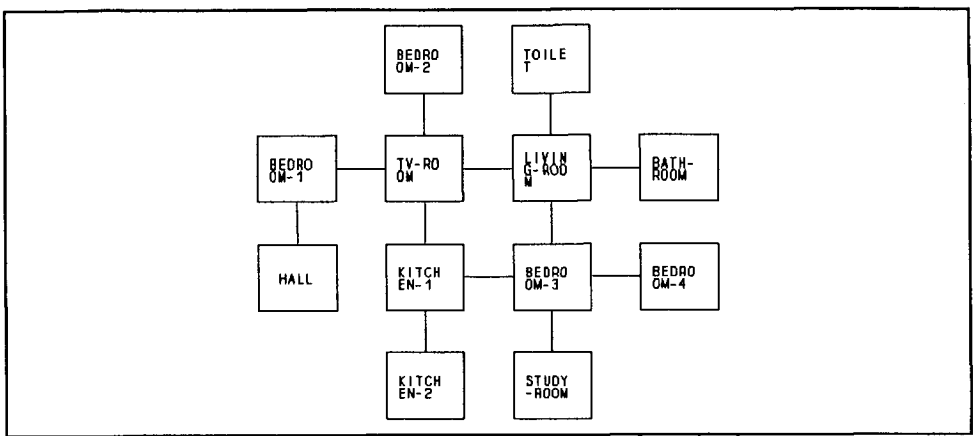


Figure 6.12: A near-optimal design for the house

Above design is generated under the following branchiness constraints:
Maximum branching degree of locations was set to 4, except the branching degrees of Bath-room = 1, Toilet = 1, Kitchen-2 = 1, and the hall = 6.

Diagnosis of the design in terms of disturbances on its locations:
TV-room = 52 units of disturbances, bedroom-3 = 32 units, kitchen-1 = 10 units, and living-room = 182 units.

Result of the **behavioral analysis** of the generated design:
Community utility = 276 units.
Privacy-cost = 276 units.
Circulation-cost = 1112 units.

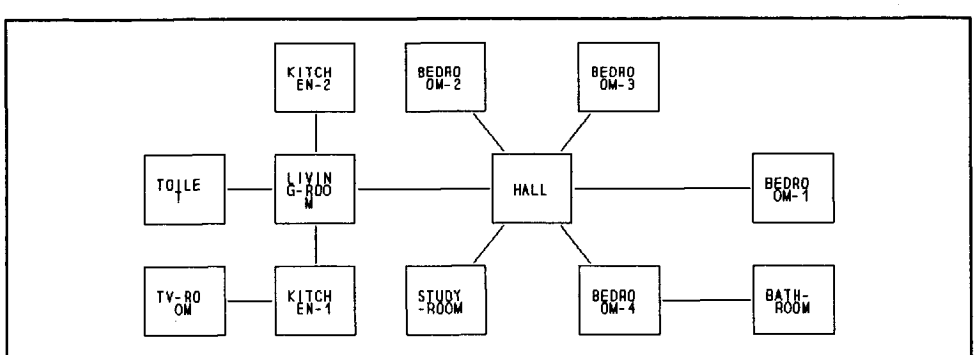


Figure 6.13: A near-optimal design for the house

Above design is generated under the following constraints:

- Knowledge base was used.
- Maximum branching degree of locations were set to 4, except: kitchen-2=1, toilet=1, bath-room=1, and hall=6.

Diagnosis of the design with respect to disturbances on locations:
bedroom-4 = 132 units, kitchen-1 = 117 units, living-room = 173 units, and hall = 195 units.

Result of the **behavioral analysis** of the generated design:
Community utility = 617 units.
Privacy-cost = 422 units.
Circulation-cost = 1453 units.

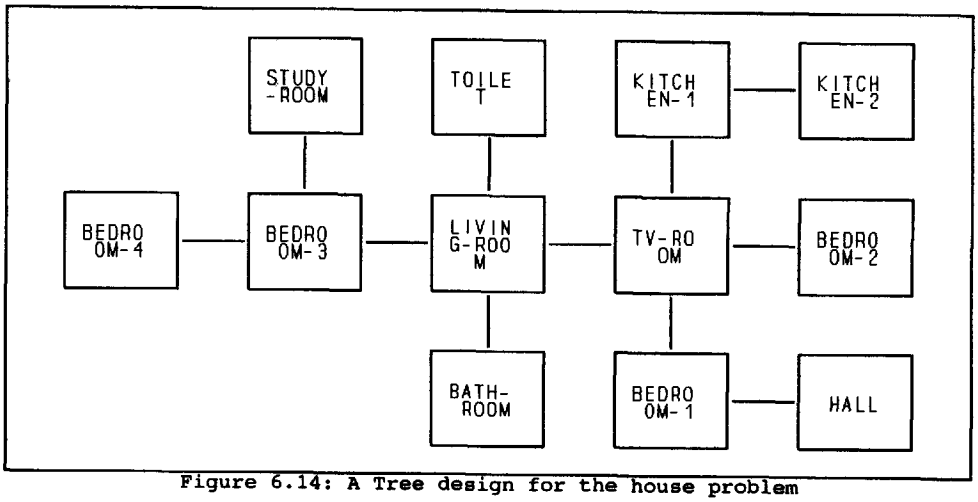


Figure 6.14: A Tree design for the house problem

Above design is generated under the following constraints:

- The knowledge bases of recommended and prohibited links were not used.
- Maximum branching degree of locations was set equal to 4, except for the bath-room = 1, toilet = 1, kitchen-2 = 1, and the hall = 6.

Diagnosis of the design in terms of disturbances on its locations:
tv-room = 67 units, bedroom-3 = 20 units, kitchen-1 = 5 units, and living-room = 190 units.

Result of the **behavioral analysis** of the generated design:
Community utility = 282 units.
Privacy-cost = 282 units.
Circulation-cost = 1118 units.

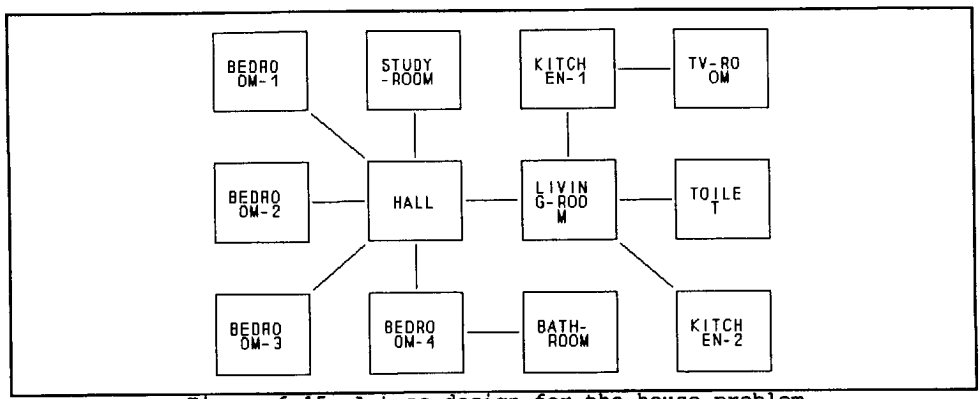


Figure 6.15: A tree design for the house problem

This design is generated under the following constraints:

Knowledge bases of recommended and prohibited accesses were used.

Maximum branching degree of locations were set to 4, except

bath-room = 1, toilet = 1, kitchen-2 = 1, and hall = 6.

Diagnosis of the design in terms of disturbances on its locations:
hall = 195 units, bedroom-4 = 132 units, kitchen-1 = 117 units, and living-room = 173 units.

Result of the **behavioral analysis** of the generated design:
Community utility = 617 units.
Privacy-cost = 422 units.
Circulation-cost = 1453 units.

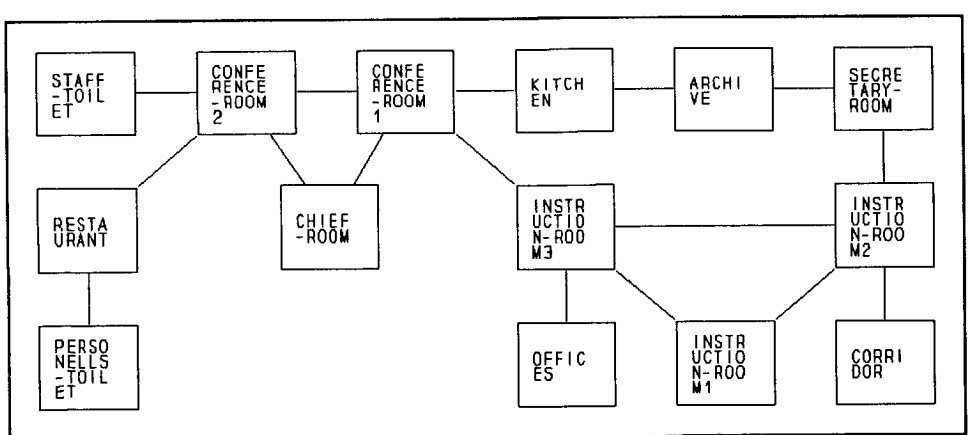


Figure 6.16: A near-optimal design for
1st-floor of the police-station

Result of the **behavioral analysis** of the generated design:
Community utility = 88 units.
Privacy-cost = 88 units.
Circulation-cost = 234 units.

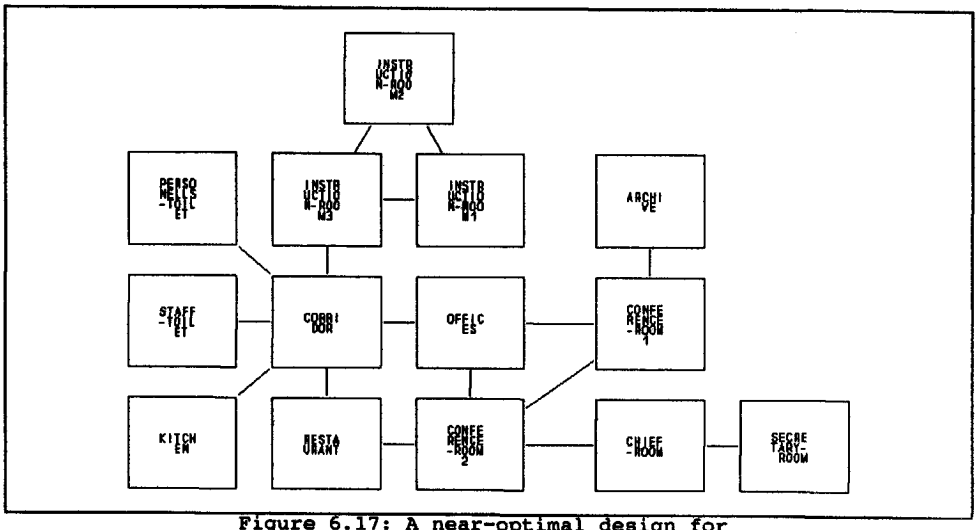


Figure 6.17: A near-optimal design for
1st-floor of the police-station

Above design is generated in the presence of the knowledge-bases.
Result of the **behavioral analysis** of the generated design:
Community utility = 83 units.
Privacy-cost = 48 units.
Circulation-cost = 229 units.

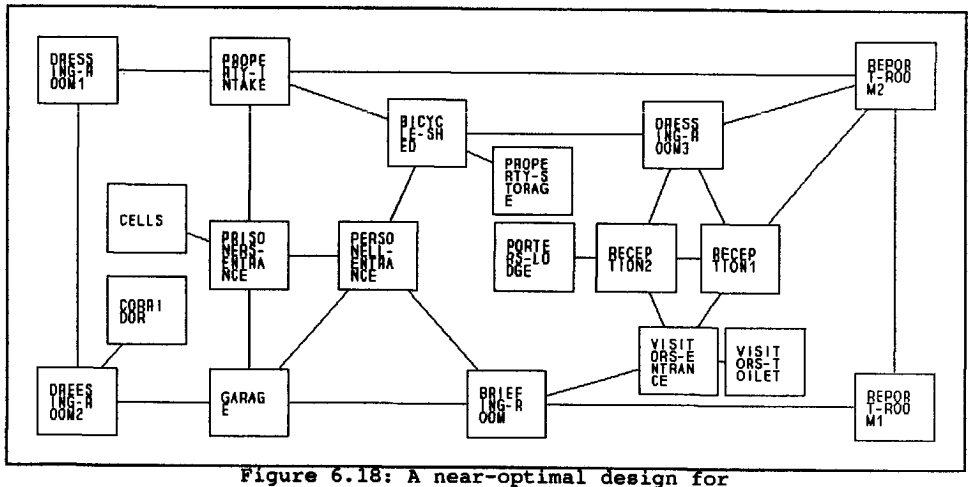


Figure 6.18: A near-optimal design for
2nd-floor of the police-station

Result of the **behavioral analysis** of the generated design:
Community utility = 63 units.
Privacy-cost = 63 units.
Circulation-cost = 217 units.

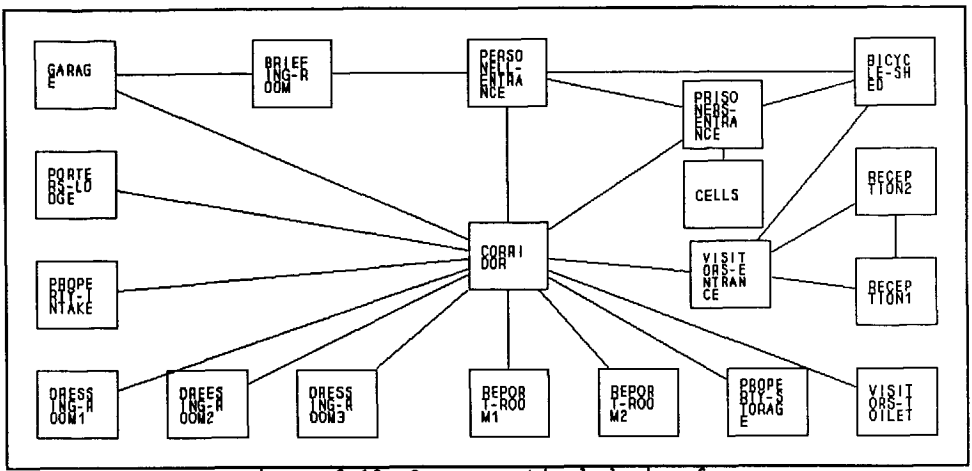


Figure 6.19: A near-optimal design for 2nd-floor of the police-station

This design is generated in the presence of the knowledge-bases.

Result of the **behavioral analysis** of the generated design:
Community utility = 82 units.
Privacy-cost = 31 units.
Circulation-cost = 236 units.

The prototypical designs

The number of possible prototypical designs for a single problem, here also as in the case of community norm, ranges to a variety of designs, depending on the constraints used. One should note that while two prototypical type designs generated with respect to two different norms for the same data-set may structurally be the same, but the arrangements of activities on them (i.e., their operation) are different.

I skip exhaustive exhibition of examples of prototypical designs generated with respect to the privacy norm here. But, in order to have an analytical view on the performance of TOPGENE in generating designs with respect to the privacy and circulation-cost norms, the results of behavioral analysis of these designs are displayed at the end of this section.

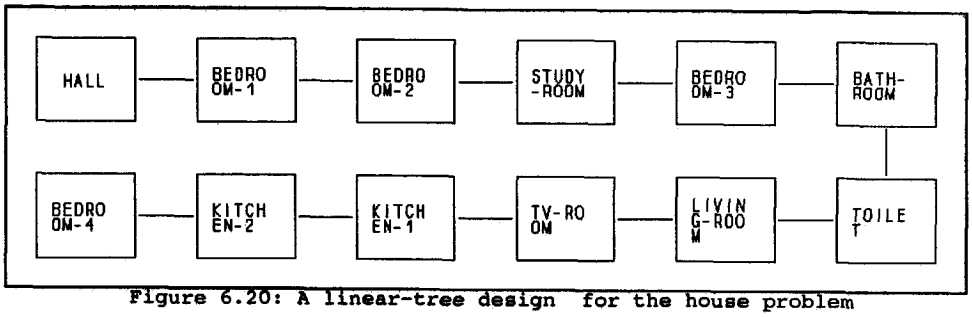


Figure 6.20: A linear-tree design for the house problem

Result of the **behavioral analysis** of the generated design:
Community utility = 392 units.
Privacy-cost = 385 units.
Circulation-cost = 1228 units.

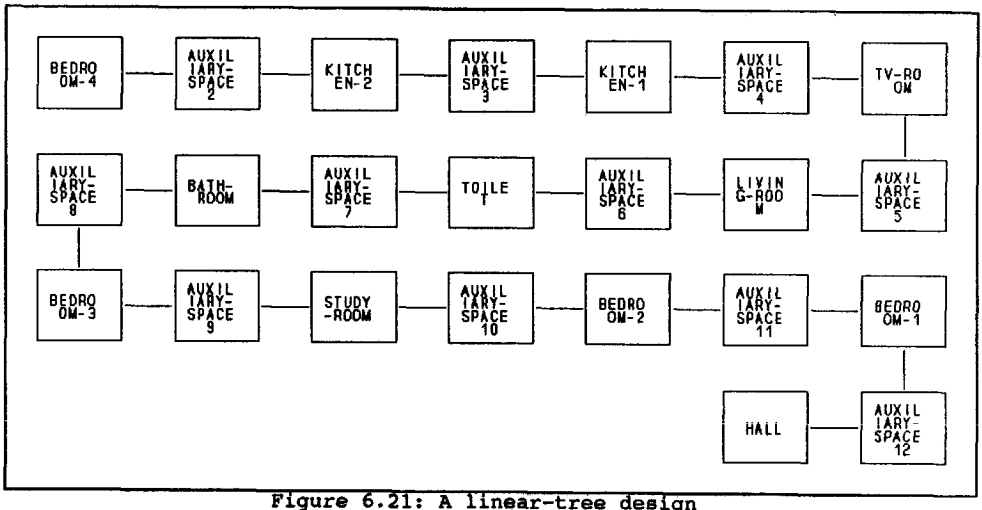


Figure 6.21: A linear-tree design
with auxiliary locations for the house

The labelling procedure for generation of grid type design with respect to the privacy / circulation-cost norm(s) is different than the generation of same design type with respect to the community norm. Here, on the contrary to the previous case, the labelling of a grid with the locations (or activities) in the privacy stream(s) proceeds differently for every adjacent rows. The reason is that, in order to have a high performance with respect to the privacy / circulation-cost norm(s), the most interactive locations in a design must be kept as close as possible relative to each other. The proposed strategy not only preserves this property of the privacy stream, but also increases it to a certain degree. This is because of additional accesses that a grid type design provides between locations of a design on every adjacent rows. The following figure depicts this labelling strategy.

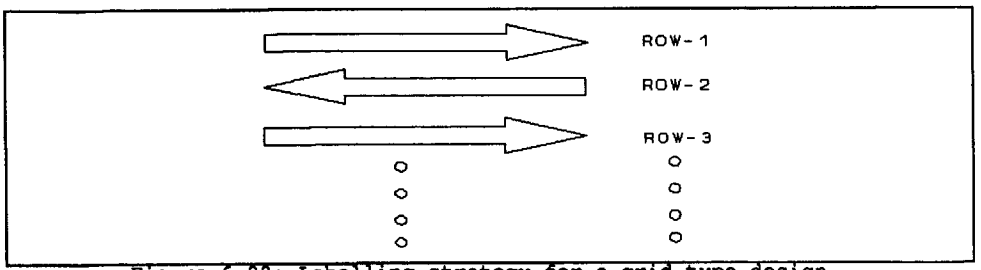


Figure 6.22: Labelling strategy for a grid type design with a privacy stream(s) of locations (activities)

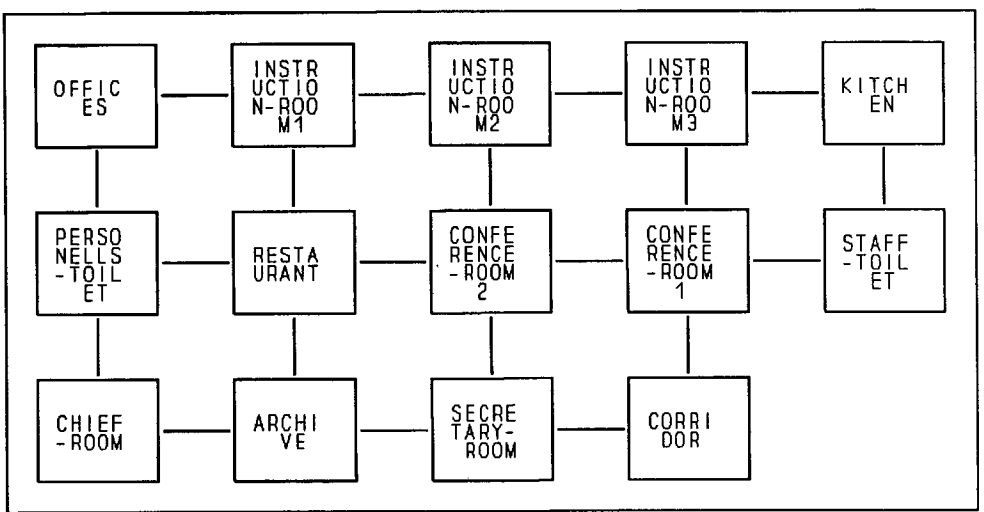


Figure 6.23: A grid design for 1st-floor of the police-station

Results of the behavioral analysis of the generated designs for the house problem with respect to the privacy / circulation-cost norm(s) are as follows:

Design type	Community utility	Privacy cost	Circulation cost
Near-optimal	276	276	1112
Near-optimal+K.B.	617	422	1453
Tree	282	282	1118
Tree+K.B.	617	422	1453
Linear-tree	392	385	1228
Linear-tree*	1202	385	2038
Single-loaded	396	389	1232
Single-loaded*	1228	0	2064
Double-loaded	436	414	1272
Double-loaded*	813	0	1649
Star	420	420	1256
Star*	678	0	1514
Grid (3*4)	390	387	1226

* Designs with auxiliary locations.

Result of the behavioral analysis of the generated designs for first-floor of the police-station with respect to the privacy / circulation-cost norm(s):

Design type	Community utility	Privacy cost	Circulation cost
Near-optimal	88	88	234
Near-optimal+K.B.	83	48	229
Tree	85	85	231
Tree+K.B.	91	54	237
Linear-tree	108	108	254
Linear-tree*	289	108	435
Single-loaded	83	83	229
Single-loaded*	254	0	400
Double-loaded	71	71	217
Double-loaded*	167	0	313
Star	81	81	227
Star*	126	0	272
Grid (3*4)	84	80	230

* Designs with auxiliary locations.

Results of the behavioral analysis of the generated designs for second-floor of the police-station with respect to the privacy / circulation-cost norm(s):

Design type	Community utility	Privacy cost	Circulation cost
Near-optimal	63	63	217
Near-optimal+K.B.	82	31	236
Tree	102	102	256
Tree+K.B.	111	56	265
Linear-tree	244	244	398
Linear-tree*	565	244	719
Single-loaded	149	149	303
Single-loaded*	398	0	552
Double-loaded	134	134	288
Double-loaded*	240	0	394
Star	129	105	283
Star*	180	0	334
Grid (3*4)	107	106	261

* Designs with auxiliary locations.
The near-optimal and tree type designs were generated under the following branchiness constraints:
Property-storage = 1, cell = 1, toilet = 1, and corridor = 20.

An analytical view on above tables, similar to case of the community norm, suggests the performance of TOPGENE's in generating different designs with respect to the point(s) of view of privacy / circulation-cost.

6.4.1 An analysis of TOPGENE performance

Here, similar to the case of other norms, the first question in our mind is about the nearness of the behavior of a generated design to the behavior of an optimal design. A question that may not be answered, unless all types of optimal designs are generated by using an enumeration technique. However, as

in case of the community norm, a comparison of linear-tree type designs for a set of data with respect to the privacy / circulation-cost norm(s) generated by both TOPGENE and a neural network will be presented in chapter 8. This comparison, as we will see, shows that designs generated by TOPGENE with respect to the privacy / circulation-cost norm(s) are also in the neighborhood of the optimal designs. This performance by TOPGENE, as a heuristic program is remarkable.

In this section we will analyze the performance of TOPGENE in generating designs with respect to the privacy and circulation-cost norms. This analysis is carried out by looking at the behavioral values of different designs, gathered in above tables, with respect to these norms.

According to our intuitions and heuristic rules used in generating these designs, we expect a high performance with respect to these norms for more compact designs, and a decrease in the performance of designs as we shift our attentions towards less compact ones. The most compact designs are the near-optimal designs. These are the designs possibly with circular paths. Above tables shows that the near-optimal designs are indeed best performing designs with respect to the circulation-cost norm. This is also true for the behavior of these designs with respect to the privacy norm, if we disregard designs with the auxiliary locations. The auxiliary locations in prototypical designs have the property of absorbing all disturbances as a result of flow between different location-pairs in a building, and therefore providing a zero privacy-cost for the building. So, these solutions have a zero privacy cost. This is of course only true if disturbances as a result of proximate flows are discarded. If we neglect these designs, we observe that for the cases of the police-station the most compact designs have a better performance with respect to the privacy and circulation-cost norms, and as we pay our attention towards more stretched type designs such as single-loaded, and linear-tree types, then their performance with respect to these norms decreases. This is a kind of performance that we expect from TOPGENE. However, in the case of designs for the house problem this is not quite true. A comparison of behavioral values of different designs for this problem seems counter-intuitive at first glance. For example, the double-loaded design seems to have a better performance than the linear-tree type design for this problem. This was quite unpredictable. A detailed analysis of the house data-set and the labelling strategy used by TOPGENE revealed the reasons behind this behavior from TOPGENE. A look at the house clusters of location-pairs shows that except for three location-pairs (living-room toilet), (bath-room toilet), and (living-room tv-room) respectively with 97, 70 and 56 degree of associations, the association degree between the rest of location-pairs drops to considerably low degrees. This irregularity in a design data, specially for small scale designs is not without a consequence for the labelling approach used by TOPGENE.

TOPGENE stream generator algorithm, in attempt to optimize the privacy performance of the stream, puts these high interactive locations as close as possible to each other. This insures a good performance for prototypical designs based on this stream. A labelling of a double loaded-design with such a stream, although intuitively, and indeed in most of the cases should increase the privacy output of the design. But, for a design data with a few number of location-pairs having a high degree of interactions, this expectation may not be fulfilled. The reason is that the labelling procedure may separate the small number of highly interactive location-pairs. This in turn may cause a poorer performance for a linear-tree type design.

I will skip a more detailed analysis of TOPGENE's behavior, and leave the matter to the reader. A detailed analysis of the system, in fact needs a continual look-back at the implementation detail of the system. As an example, one may wounder whether a star type design for a data-set should have a better performance with respect to privacy norm or a double-loaded type design? The answer to this problem lies in the branching degree of central locations in the star type design. This branching degree is a user controlled variable. A star type design with a somewhat high branching degree with respect to a double-loaded design obviously should have a better privacy and circulation costs performances than a double-loaded design for the same data-set.

6.5 Designs with respect to the intervening opportunity norm

This norm, although in conflicts with all other social norms, but is in some respect similar to the community norm. An optimal design with respect to the intervening opportunity (hereafter IO) norm is a design that satisfies a set of in-betweenness conditions demanded from the design. The most suitable design type for this norm, the same as the community norm, tends towards a linear-tree type design. This implies that a satisfactory design, specially for a problem with relatively high in-betweenness requirements between its locations, inclines towards a liner-tree design. For this reason, TOPGENE precedes the generation of any design type for this norm by first generating a stream of locations that satisfies maximum in-betweenness requirements. The exception is the tree design. TOPGENE distinguishes a tree-type design for this norm from a linear-tree type. This approach was taken to show the appropriateness of a general tree type design for the intervening opportunity norm. Otherwise, a linear-tree design generally yields a better result than a tree type design for the same set of data. A prototypical design, here, as in the case of previous norms, is labelled by locations (activities) in the stream of locations.

I have chosen the house data-set to show examples of designs with

respect to this norm. Furthermore, the following intervening opportunity requirements have been assumed for the house:

(Bedroom-1 hall living-room) (Bedroom-2 hall living-room)
(Hall bedroom-2 bedroom-3) (Kitchen-1 hall toilet)
(Hall toilet bath-room) (Kitchen-1 hall living-room)
(Kitchen-2 kitchen-1 hall) (Hall bedroom-3 study-room)
(Hall bedroom-1 toilet) (Hall bedroom-2 toilet)

Upon analysis of the data, TOPGENE ranks the input triplets according to the interaction potential between their bracketing pair. The ranking triplets are used as a guide in choosing next location to be processed.

The following figures show samples of designs generated with respect to this norm. Note that in the case of this norm, TOPGENE provides us with the behavioral analysis of designs with respect to all social norms discussed in this work. The additional behavioral values for this case, as in previous cases, are produced as a side effect of the process of the behavioral analysis.

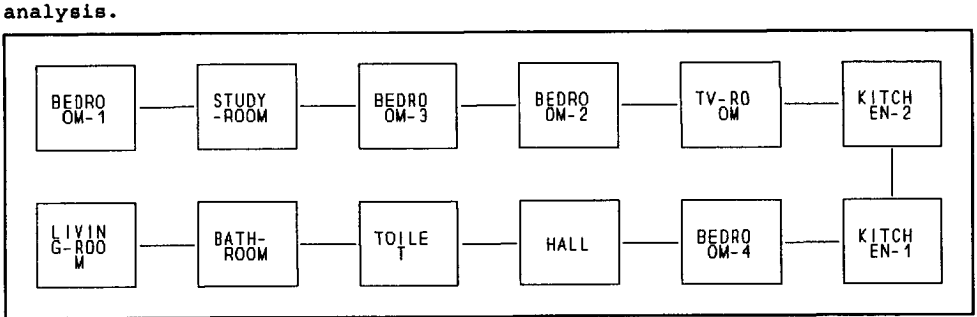


Figure 6.24: A near-optimal design for the house

This design is generated under the above defined intervening opportunity requirements.

Diagnosis of design with respect to the intervening opportunity (IO) norm:
The IO utility of the HALL as a result of being on the shortest path between the location-pair (KITCHEN-1 LIVING-ROOM) is 10.
The IO utility of the HALL as a result of being on the shortest path between the location-pair (BEDROOM-2 LIVING-ROOM) is 6.
The IO utility of the HALL as a result of being on the shortest path between the location-pair (BEDROOM-1 LIVING-ROOM) is 6.
The IO utility of the HALL as a result of being on the shortest path between location-pairs (KITCHen-1 AND TOILET) is 6.

Result of the **behavioral analysis** of the generated design:
Community utility = 1272 units.
Privacy-cost = 1074 units.
Circulation-cost = 2108 units.
Int. opportunity utility = 28 units.

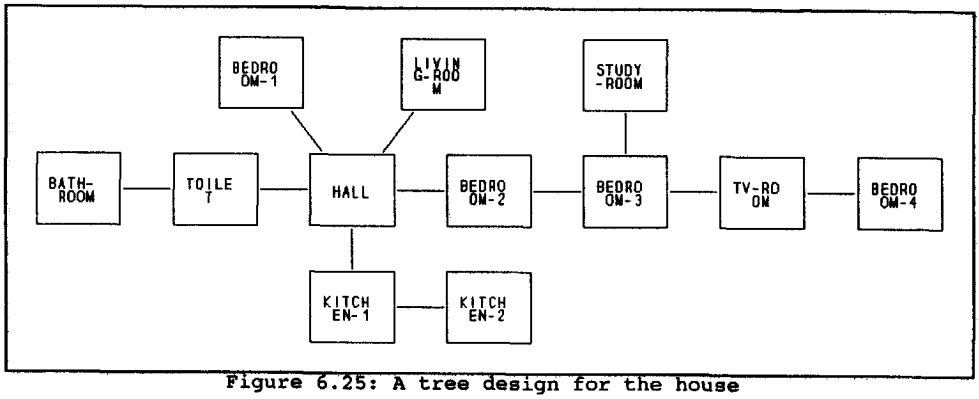


Figure 6.25: A tree design for the house

This design is generated under the intervening opportunity requirements defined above, and the following branchiness constraints: Branching degree of each location is equal to 4, except the hall that was set to 6.

Diagnosis of design with respect to the intervening opportunity (IO) norm:
The IO utility of the HALL as a result of being on the shortest path between the location-pair (KITCHen-1 LIVING-ROOM) is 10.
The IO utility of the HALL as a result of being on the shortest path between the location-pair (BEDROOM-2 LIVING-ROOM) is 6.
The IO utility of the HALL as a result of being on the shortest path between the location-pair (BEDROOM-1 AND LIVING-ROOM) is 6.
The IO utility of the HALL as a result of being on the shortest path between the location-pair (KITCHen-1 AND TOILET) is 6.

Result of the analysis of different design types with respect to the IO norm for the house data-set is gathered in the following table.

Design type	community utility	privacy cost	Circ. cost	I.O. utility
Tree	670	348	1506	28
Linear-tree(SO)	272	1074	2108	28
Linear-tree*	2962	1074	3798	28
Double-loaded	906	540	1742	28
Single-loaded*	2108	0	2944	0
Double-loaded	570	570	1406	0
Double-loaded*	1243	0	2079	0
Star	530	530	1366	0
Star*	1011	0	1847	0
Grid (3*4)	524	486	1360	1

* design with auxiliary locations.

6.5.1 An analysis of TOPGENE performance

Looking at the behavioral values of different design types, gathered in above table, show that the high performing designs with respect to this norm, as we expect, are the linear-tree and tree designs. Above table also suggests that other prototypical designs, such as single-loaded, double-loaded, and star designs have a very bad performance with respect to this norm. This performances are not unexpected. Prototypical designs with auxiliary locations can not provide intervening opportunity utility at all. All activities on such designs are allocated on branches. These locations obviously cannot function as an intervening opportunity point for any bracketing location-pair. Similarly, in other prototypical designs, most of the activities are allocated to isolated branches. So, non of these activities can act as an intervening opportunity activity.

In general, we can say that prototypical designs are not good candidates for yielding a high performance with respect to the intervening opportunity norm, unless the concept of proximate flow (i.e., flow passing nearby a location) is counted in designs. In this case, which is indeed practical in many designs, people passing-by the intervening locations, and not crossing them, in fact benefit from these locations. Present version of TOPGENE, as was discussed in the last chapter, does not consider the effect of nearby flows to the performance calculation of a building. Besides, all prototypical designs, here as in the cases of other norms, are generated based on a stream of locations. Here, the intervening opportunity stream, as opposed to other streams that were controlling the interaction potentials between the actors, provides the in-betweenness property of a design. The latter property attributed to an intervening opportunity stream is quite different from the previous one. The in-betweenness property is much more sensitive to changes in a stream than interaction in the same stream. The labelling of a branched prototypical design with an intervening opportunity norm completely disturbs the operation of design implicit in the order of locations in the stream. That is why, here, labelling of any prototypical designs, other than the linear-tree type designs with an intervening opportunity stream, has significant impact on the operation of design with respect to this norm. A solution to this shortcoming is, devising specialized algorithms for each prototypical case. The current approach in generating design with respect to the intervening opportunity norm based on the stream of locations is, therefore, only valuable for the linear-tree and tree types designs. These designs are near-optimal designs for this norm.

6.6 Designs with respect to multiple norms

This section presents examples of designs with respect to a combination of norms. The near-optimal and tree-type designs, as explained in the last chapter, are generated by special task-executers, while the prototypical designs are generated by the prototypical-task-executer. The latter one first generates a stream of locations (activities) that is for labelling of a prototypical design at a latter stage.

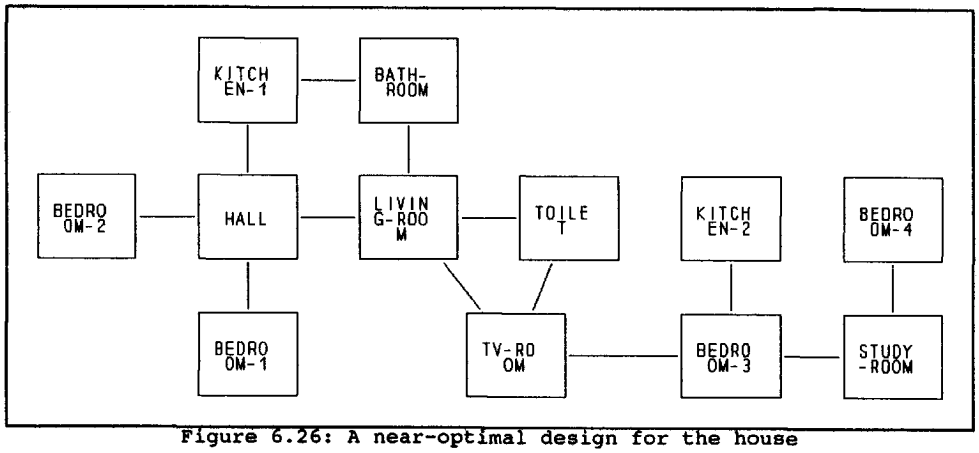


Figure 6.26: A near-optimal design for the house

Above design is generated under the following constraints:

- Norms defined: Intervening opportunity, privacy / circulation-cost, and community.
- The intervening opportunity triplets defined for the locations: the same as those used in the section 6.5.

Diagnosis of the design with respect to the intervening opportunity (IO) norm:
The IO utility of the HALL as a result of being on the shortest path between the location-pair (BEDROOM-2 LIVING-ROOM) is 6.
The IO utility of the HALL as a result of being on the shortest path between the location-pair (BEDROOM-1 LIVING-ROOM) is 6.
The IO utility of the HALL as a result of being on the shortest path between the location-pair (KITCHen-1 LIVING-ROOM) is 5.
The IO utility of the HALL as a result of being on the shortest path between the location-pair (KITCHen-1 AND TOILET) is 3.

Result of the **behavioral analysis** of the generated design:
Community utility = 324 units.
Privacy-cost = 260 units.
Circulation-cost = 1160 units.
I.O. utility = 20 units.

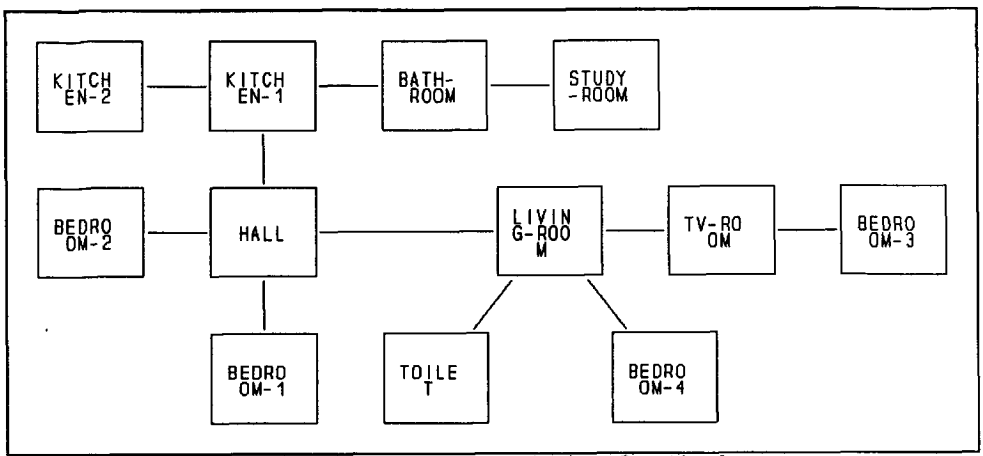


Figure 6.27: A tree type design for the house

Above design is generated under the same conditions as defined for above design.

Result of the **behavioral analysis** of the generated design for the house:
Community utility = 554 units.
Privacy-cost = 360 units.
Circulation-cost = 1390 units.
I.O. utility = 28 units.

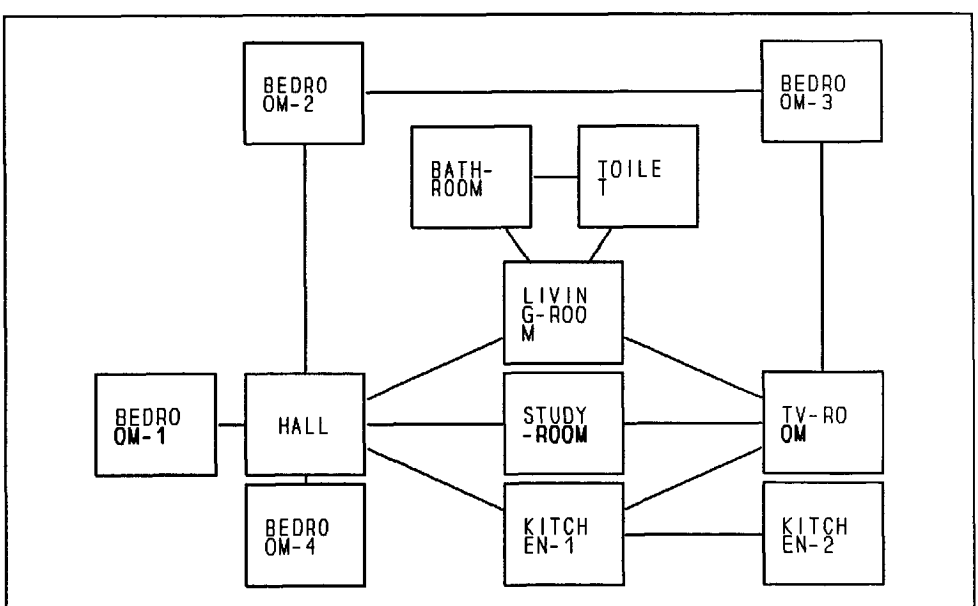


Figure 6.28: A near-optimal type design for the house

Above design is generated under the constraints of intervening opportunity and

privacy norms. The intervening opportunity triplets were the same as used for previous design. Maximum branching degree of locations were set to 4, except: toilet = 2, bath-room = 2, and hall = 6.

Diagnosis of the design with respect to the intervening opportunity (IO) norm:
The IO utility of the HALL as a result of being on the shortest path between the location-pair (BEDROOM-2 LIVING-ROOM) is 6 units.
The IO utility of the HALL as a result of being on the shortest path between the location-pair (BEDROOM-1 LIVING-ROOM) is 6 units.
The IO utility of the HALL as a result of being on the shortest path between the location-pair (KITCHen-1 LIVING-ROOM) is 5 units.
The IO utility of the HALL as a result of being on the shortest path between the location-pair (KITCHen-1 AND TOILET) is 3 units.

Diagnosis of the design in terms of disturbances on locations:
study-room = units, bedroom-3 = 4 units, kitchen-1 = 7 units, tv-room = 44 units, living-room = 109 units, and hall = 68 units.

Result of the **behavioral analysis** of the design:
Community utility = 233 units.
Privacy-cost = 165 units.
Circulation-cost = 1069 units.
I.O. utility = 20 units.

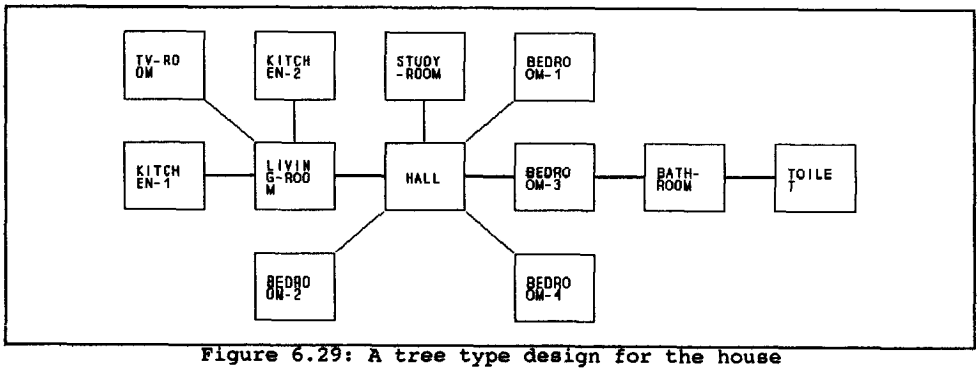


Figure 6.29: A tree type design for the house

Above design is generated under the intervening opportunity and Privacy norms. Maximum branching degree of locations was set to 4.

Diagnosis of the design with respect to the intervening opportunity (IO) norm:
The IO utility of the HALL as a result of being on the shortest path between the location-pair (BEDROOM-2 LIVING-ROOM) is 6 units.
The IO utility of the HALL as a result of being on the shortest path between the location-pair (BEDROOM-1 LIVING-ROOM) is 6 units.
The IO utility of the HALL as a result of being on the shortest path between the location-pair (KITCHen-1 LIVING-ROOM) is 10 units.
The IO utility of the HALL as a result of being on the shortest path between the location-pair (KITCHen-1 AND TOILET) is 6 units.

Diagnosis of the design in terms of disturbances on its locations:
bedroom-1 = 1 units, bedroom-3 = 13 units, for kitchen-1 = 5 units, bedroom-2 = 42 units, toilet = 63 units, hall = 119 units, and living-room = 130 units.

Result of the **behavioral analysis** of the design:
Community utility = 372 units
Privacy-cost = 253 units.
Circulation-cost = 1208 units.
I.O. utility = 28 units.

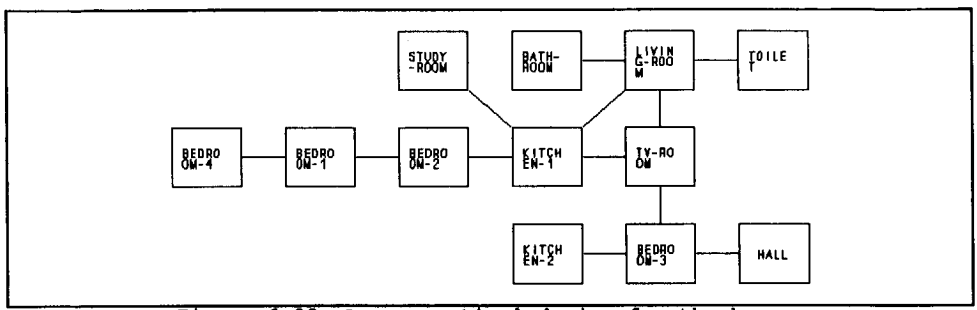


Figure 6.30: A near-optimal design for the house

Above design was generated under the community and privacy / circulation-cost norms without use of the knowledge-bases. Maximum branching degree of locations for this design was set to 4, except, kitchen-2 = 1, toilet = 1, bath-room = 1, and hall = 6 units.

Diagnosis of the design in terms of disturbances on its locations:
kitchen-1 = 70 units, tv-room = 39 units, bedroom-2 = 31 units,
bedroom-3 = 14 units, bedroom-1 = 7 units, and living-room 177 = units.

Result of the **behavioral analysis** of design:
Community utility = 338 units.
Privacy-cost = 338 units.
Circulation-cost = 1174 units.

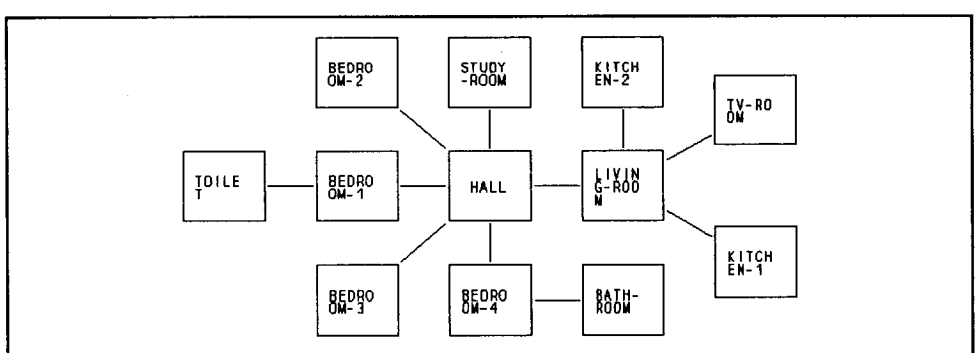


Figure 6.31: A near-optimal design for the house

This design is generated under the same conditions as used for the previous design, except that the knowledge-bases are used.

Diagnosis of the above design with respect to the privacy norm:
The disturbance degree for: bedroom-1 = 222 units, bedroom-4 = 132 units, living-room = 91 units, and hall = 131 units.

Result of the **behavioral analysis** of design:
Community utility = 776 units.
Privacy-cost = 445 units.
Circulation-cost = 1612 units.

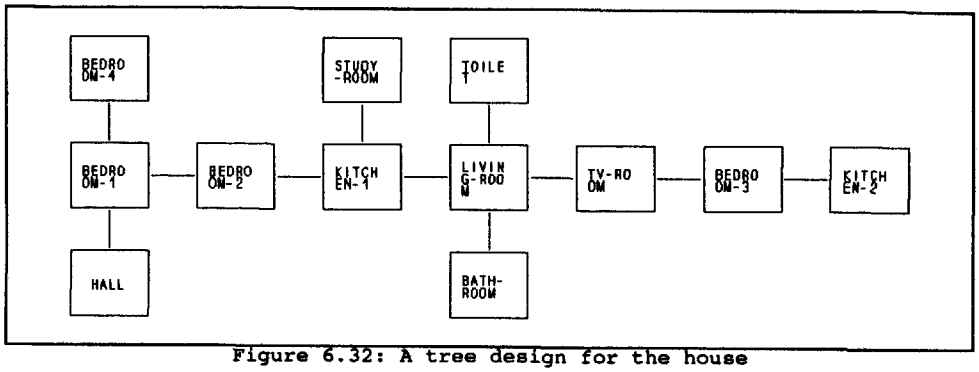


Figure 6.32: A tree design for the house

Above design is generated under the same conditions as used for previous design, except that the knowledge-base was not used.

Diagnosis of the generated design: The disturbance degree for: bedroom-1 = 7 units, or bedroom-2 = 31, bedroom-3 = 14 units, kitchen-1 = 70 units, tv-room = 39 units, and living-room = 209 units.

Result of the **behavioral analysis** of the design:
Community utility = 370 units.
Privacy-cost = 370 units.
Circulation-cost = 1206 units.

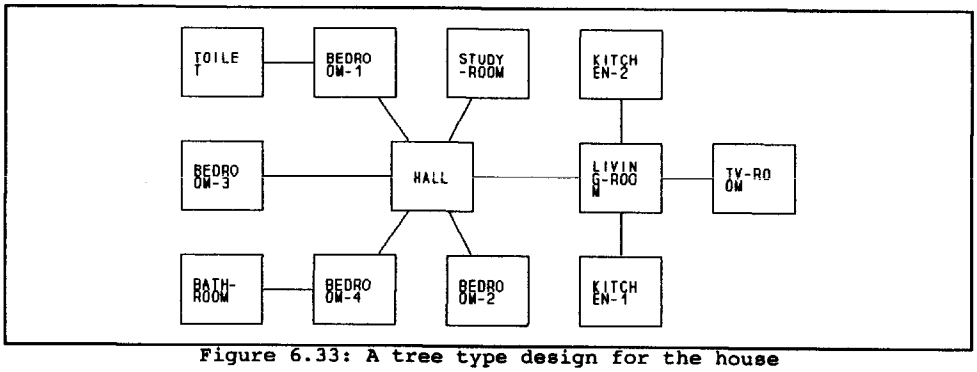


Figure 6.33: A tree type design for the house

Above design is generated under the same conditions as used for previous design, except that the knowledge-base was used.

Result of the **behavioral analysis** of the generated design:
Community utility = 776 units.
Privacy-cost = 445 units.
Circulation-cost = 1612 units.

Analysis results of different designs for the house data-set generated with respect to the combination of norms community and privacy / circulation-cost is gathered in the following table.

Design type	Community utility	Privacy cost	Circulation cost
Near-optimal	338	338	1174
Near-optimal+K.B.	776	445	1612
Tree	370	370	1206
Tree+K.B.	776	445	1612
Linear-tree	584	584	1420
Linear-tree*	1586	584	2422
Single-loaded	458	458	1294
Single-loaded*	1420	0	2256
Double-loaded	494	494	1330
Double-loaded*	876	0	1712
Star	478	478	1314
Star*	801	0	1637
Grid (3*4)	337	371	1213

The near-optimal and tree designs, in the above table, were generated under the following branching degree constraints: kitchen-2 = 1, toilet = 1, bath-room = 1, hall = 6, and all other locations = 4.

6.6.1 An analysis of TOPGENE performance

There are several ways to proceed with a performance analysis of TOPGENE in generating designs with respect to a combination of norms. We can analyze TOPGENE behavior by comparing the behavioral values of different designs with each other, or contrasting these values with behavioral values of the same designs generated with respect to single norms. TOPGENE, theoretically and practically, can generate unlimited number of designs with respect to a combination of norms. Every design, depending on the number and order of the norms defined, has different behavior. Therefore, several designs may be generated and analyzed for a set of design data. Here, we limit ourselves to an analysis of generated designed that their behavioral values are gathered in above table.

Identical conditions for different design types provide a basis for analysis of TOPGENE's performance. For example, designs exhibited in figure 6.26 and 6.27 were generated for the house and under the same conditions. The only difference between these two is design type. The former is a near-optimal design in the form of a general graph, and the latter is a tree type design. Defining a design as a near-optimal type implies the possibility of having circular paths in the design, while a tree type design does not permit such paths. The existence of circular paths in a design means more accesses between the locations in comparison to a tree type design for the same design problem. Extra accesses result in less disturbance caused by circulation flows, and consequently provide lower privacy and circulation-costs, and lower community benefit for the design. A comparison of analysis results of these two designs shows that indeed, these conditions are satisfied for these two designs. Besides, the tree type design has a higher community and intervening opportunity utilities compared to the near-optimal one. TOPGENE considers the

intervening opportunity benefits if IO locations are in-between the bracketing locations on a shortest path. This implies that an increase in accesses in a design, reduces the chances of fulfilling several in-betweenness conditions in the design. In other words, it is always harder to realise IO locations in-between the bracketing points in a design with a highly connected locations and with circular paths than in a tree type design. This is also the case for designs depicted in figures 6.26 and 6.27. The IO benefit in the latter case is 28 units in comparison to 20 units for the former one. This argument is also valid for other designs depicted in figures 6.28 and 6.29.

A second analysis may be carried out by comparisons of above table with the tables exhibited in sections 6.3 and 6.4 for the same problem but with respect to single norms privacy and community. Such comparisons show that: all designs generated with respect to the combination of norms community and privacy (or circulation-cost) has performance values for these two norms somewhere in-between the behavioral values of identical designs generated with respect to each norm separately. The only exception is the grid type design. In this case the two designs generated with respect to the privacy norm and a combination of norms almost have the same performance values. A compromise is seen in the performance of the design with respect to multiple norms in comparison with a design generated for the same set of data, but with respect to a single norm. This result shows that TOPGENE overall approach in generating different designs with respect to a combination of norms is in concordance with our intuition. Above table also confirms our expectation that:

- A compromised design with respect to all three norms should be in the form of a near-optimal or a tree type design, and not a linear-tree type design. The reason was given above.
- More compact designs lean more towards the privacy and circulation-cost norms, and not the community norm.

6.7 Evaluation of existing designs

The evaluation of existing designs with respect to social norms is executed by the evaluator module of TOPGENE. This section presents evaluation results for the two data-sets presented in the beginning of this chapter.
Evaluation of a design with respect to a set of norms, as it was described in the previous chapter, proceeds by first generating a set of sub-optimal yardstick designs, each with respect to an isolated point of view. In the next stage the analysis result of the input-design is compared with that of the generated yardstick designs.

In the following sub-sections record of evaluation of existing designs for the house, police-station, and the hospital by TOPGENE is presented. The

evaluation process for each floor of existing designs corresponding to the data sets discussed in this chapter is carried out irrespective of other floors. The reason is irregularity of connectivity pattern of these existing designs. I believe that the evaluation of a multi-stories design, as a whole, against design types discussed in this thesis, is misleading to a certain degree, unless it is very much similar to a design type discusses so far. The connectivity patterns of the multi-stories building presented in this chapter are unlike to design types discussed so far. For example, the connectivity-pattern of the hospital, as a whole, is similar to a loaded grid. This pattern does not match design types presently considered by TOPGENE. Besides, exhaustive enumeration of all design types by TOPGENE is not practical. So, evaluating these designs as a whole against any design types discussed so far is unrealistic, although possible by TOPGENE. Such an evaluation is logical only if the topology of a multi-stories building is similar to a design pattern that could be generated by TOPGENE.

I should, immediately refer to the fact that isolated evaluation of each floor is not also without a consequence. This approach also loses the touch with the reality that different floor of a design often has interaction with each other. Still, I think that separate evaluation of each floor of a design provides a better result than an evaluation based on the total design.

6.7.1 Evaluating the house

Evaluation of the existing design for the house data-set as it has been carried out by TOPGENE is presented here. TOPGENE allows user to decide the type (i.e., style) of the yardstick designs that are used for evaluation of existing designs. I have assumed that the most important social aspect for a house is privacy. A high performing design with respect to the privacy and also circulation-cost norm is a near-optimal type design. Defining a near-optimal design for other two norms, community and intervening opportunity, results in a linear-tree type design, which is high performing design with respect to these norms. The yardstick design for evaluating the house was defined as "near-optimal". Furthermore, I have assumed that the intervening opportunity requirements for the house are the same as those defined in section 6.5.

Existing topology of the house, as it has been designed by a designer, and result of the evaluation is as follows. The connectivity pattern of the yardstick designs were already exhibited in previous sections, therefore they are skipped here.

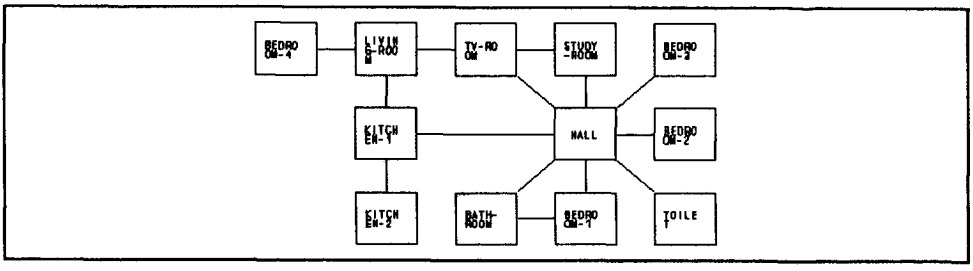


Figure 6.34: Design the house as is designed by a designer

Result of the **behavioral analysis** of designs:

Design	**community utility**	**privacy cost**	**circ. cost**	**IO utility**
Input-design:	464	149.5	1300	22
community yardstick	617	422	1453	12
privacy yardstick	617	422	1453	12
circulation-cost yardstick	1272	1074	2108	28

Evaluation result:
The *community utility* of the input-design is about 17% of the community utility of the yardstick design, and it is considered *very-bad*.
The *privacy-cost* is 65% lower than the privacy-cost of the yardstick design, and it is *excellent*.
The *circulation-cost* is about 11% lower than the circulation-cost of the yardstick design, and it is *excellent*.
The *IO utility* is 64% of the IO utility of the yardstick design, and it is *fair*.

6.7.2 Evaluating the police-station

The existing design to the police-station, depending on the interpretation of the corridor in each floor, is closely related to either a near-optimal, single-loaded or a double-loaded solution. So, it seems logical to evaluate this design against one or both designs as yardstick design(s). To show the capability of TOPGENE, and to improve insight in the existing designs of police-stations this evaluation is carried out with respect to all three yardsticks. Furthermore, each floor of the police-station is evaluated separately and irrespective of the other floor.

First run conditions:

- Corridor is taken as a single location.
- Evaluation against near-optimal yardstick designs.
- Norms: Community, privacy, and circulation-cost.

Evaluating the first-floor:

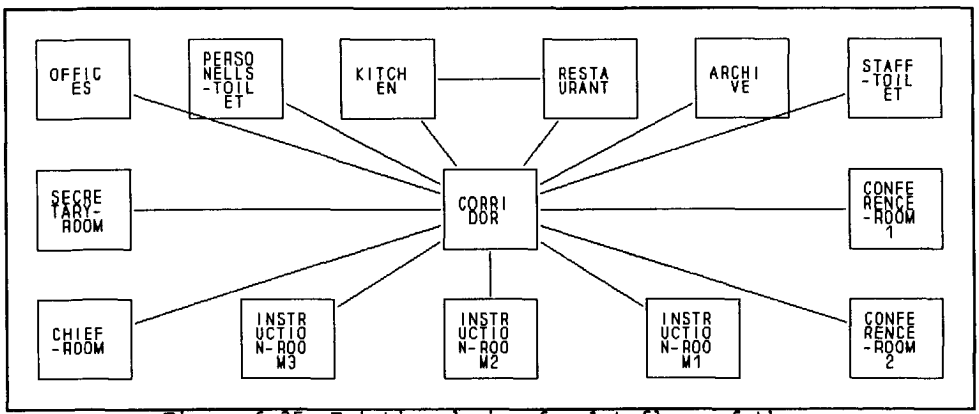


Figure 6.35: Existing design for 1st-floor of the police-station with the corridor taken as a single location

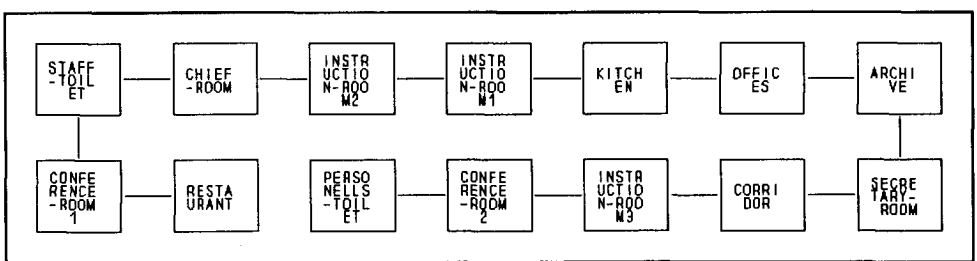


Figure 6.36: A near-optimal (linear-tree) yardstick design for 1st-floor of the police-station

Above design is generated by TOPGENE with respect to the "community" norm.

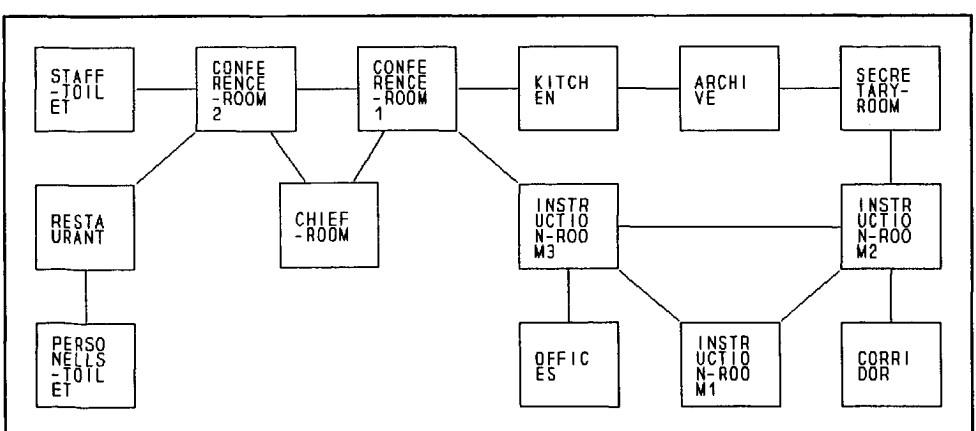


Figure 6.37: A near-optimal yardstick design for 1st-floor of the police-station

Above design is generated by TOPGENE under the following constraints:

- Norm(s) defined: "privacy" / "circulation-cost".
- The knowledge-bases of recommended and prohibited accesses were used.
- Maximum branchiness for each location = 4, except: chief-room = 2, secretary's room = 2, restaurant = 2, kitchen = 2, toilets = 1, and corridor = 10.

Result of the **behavioral analysis** of designs:

Design	community utility	privacy cost	circulation cost
Input-design	72	0	218
community yardstick	386	352	532
privacy yardstick	83	48.5	229
circulation-cost yardstick	83	48.5	239

Evaluation result:
The *community utility* of the input-design is about 19% of a near-optimal design, and it is considered *as bad*.
The *privacy-cost* is lower than a near-optimal design, and it is *excellent*.
The *circulation cost* is about 5% lower than a near-optimal design, and it is *excellent*.

Evaluating the second-floor:

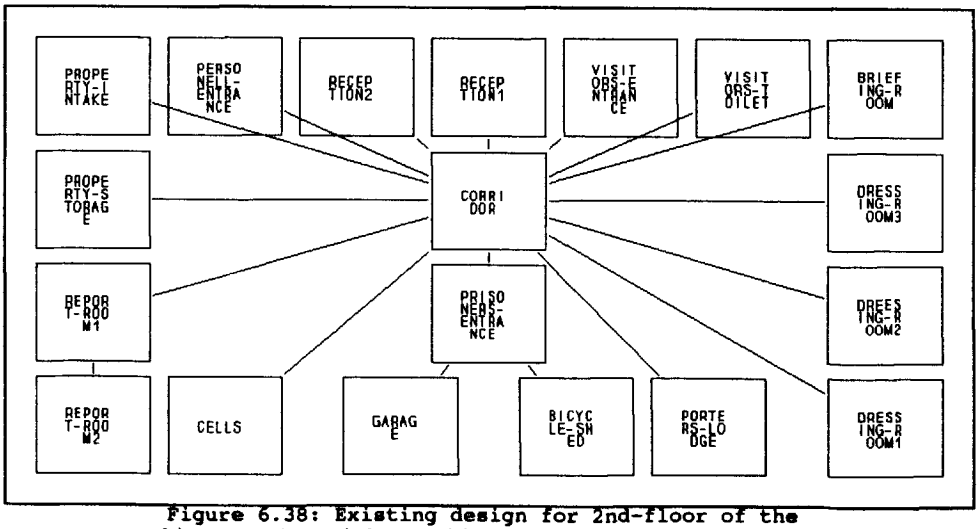


Figure 6.38: Existing design for 2nd-floor of the police-station with corridor taken as a single location

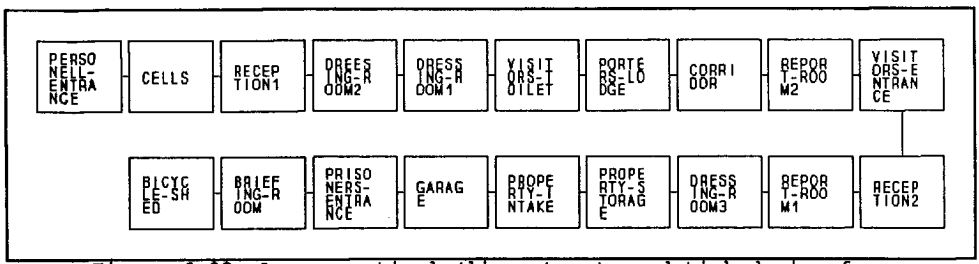


Figure 6.39: A near-optimal (liner-tree) yardstick design for 2nd-floor of the police-station

Above design is generated with respect to the "community" norm.

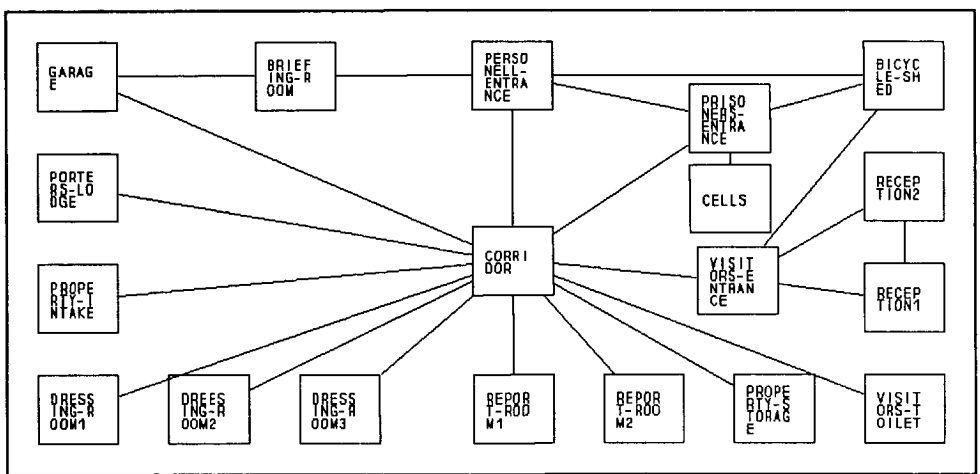


Figure 6.40: A near-optimal yardstick design for 2nd-floor of the police-station

Above design is generated under constraints of the privacy (circulation-cost) norm, the knowledge bases of accesses, and branchiness degree. The branching degree of locations were defined as 4, except for the following locations: toilets = 1, cells = 1, property-storage = 1, and corridor = unlimited.

Result of the **behavioral analysis** of designs:

Design	community utility	privacy cost	circulation cost
Input-design	99	26	253
community yardstick	572	528	726
privacy yardstick	82	30	236
circulation-cost yardstick	82	30	236

Evaluation result:
The *community utility* of the input-design is about 17% of a near-optimal design, which is considered *as bad*.
The *privacy-cost* is about 15% lower than a near-optimal design, and it is *excellent*
The *circulation-cost* is about 7% higher than a near-optimal design, and it is *excellent*.

Second run conditions:

- Multiple corridor assumption:
- Evaluation against single loaded yardstick designs.
- Points of view of evaluation (norms): Community, privacy, and circulation-cost.

Evaluating the first-floor:

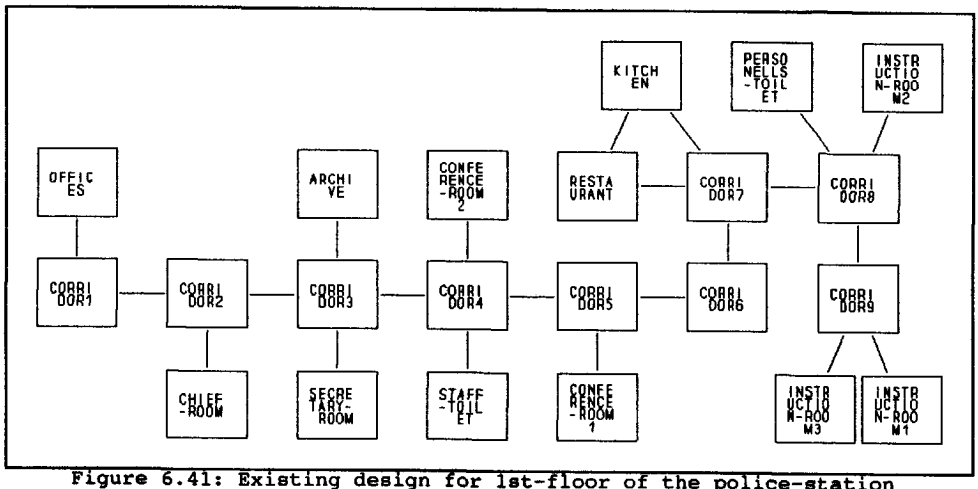


Figure 6.41: Existing design for 1st-floor of the police-station with the corridor taken as a sequence of multiple locations

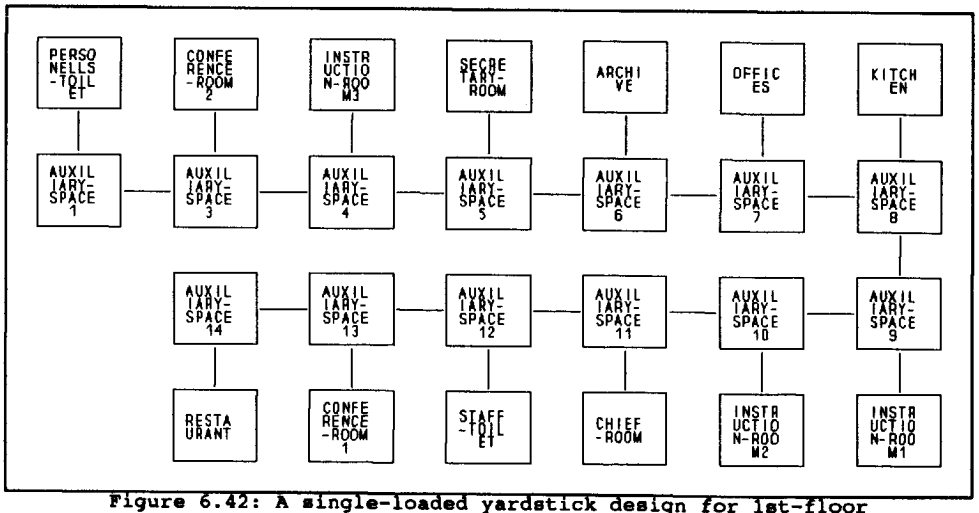


Figure 6.42: A single-loaded yardstick design for 1st-floor of the police-station

Above design is generated with respect to the community norm.

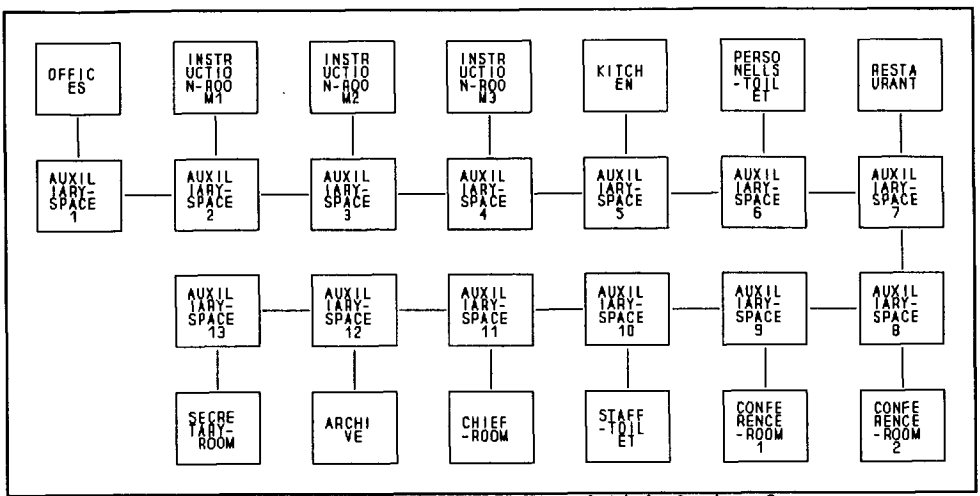


Figure 6.43: A single-loaded yardstick design for 1st-floor of the police-station

Above design is generated with respect to the "Privacy" / "circulation-cost" norm(s).

Result of the **behavioral analysis** of designs:

Design	community utility	privacy cost	circulation cost
Input-design	72	0	218
community yardstick	583	0	678
privacy yardstick	254	0	400
circulation-cost yardstick	254	0	400

Evaluation result:
The *community utility* of the input-design is about 14% of a near-optimal design, which is considered as *bad*.
The *privacy-cost* is 0 and it is *excellent*.
The *circulation-cost* is about 46% lower than the optimal design, and it is *excellent*.

Evaluating the second-floor:

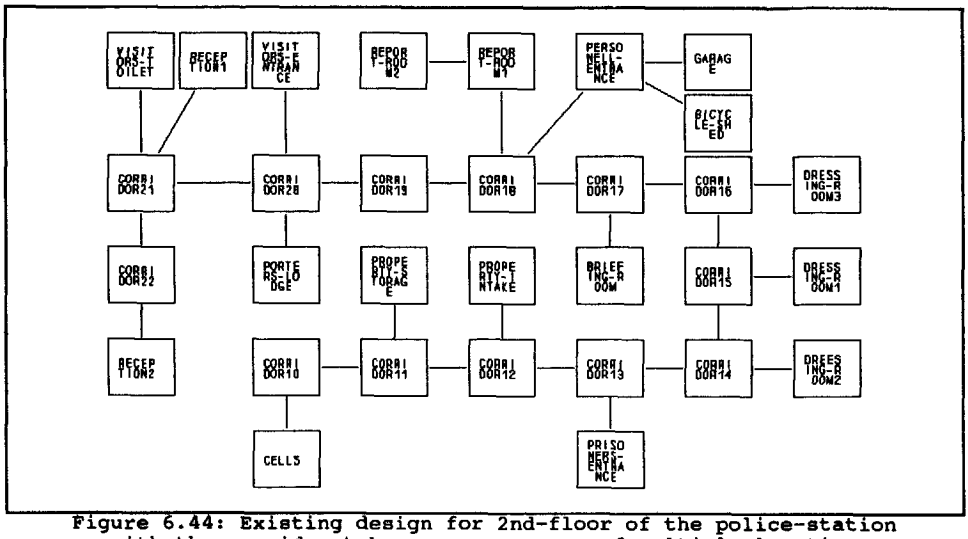


Figure 6.44: Existing design for 2nd-floor of the police-station with the corridor taken as a sequence of multiple locations

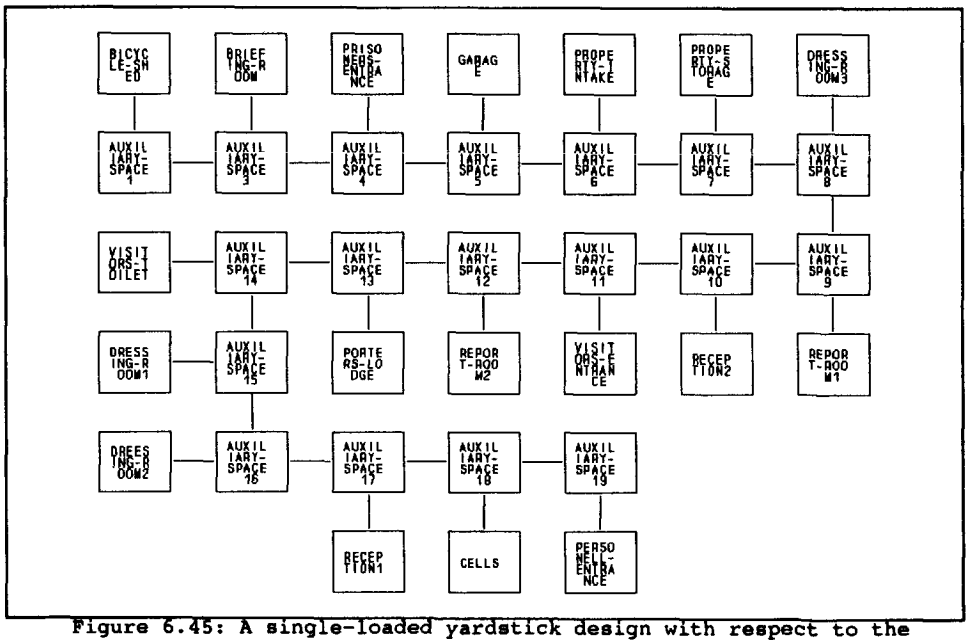
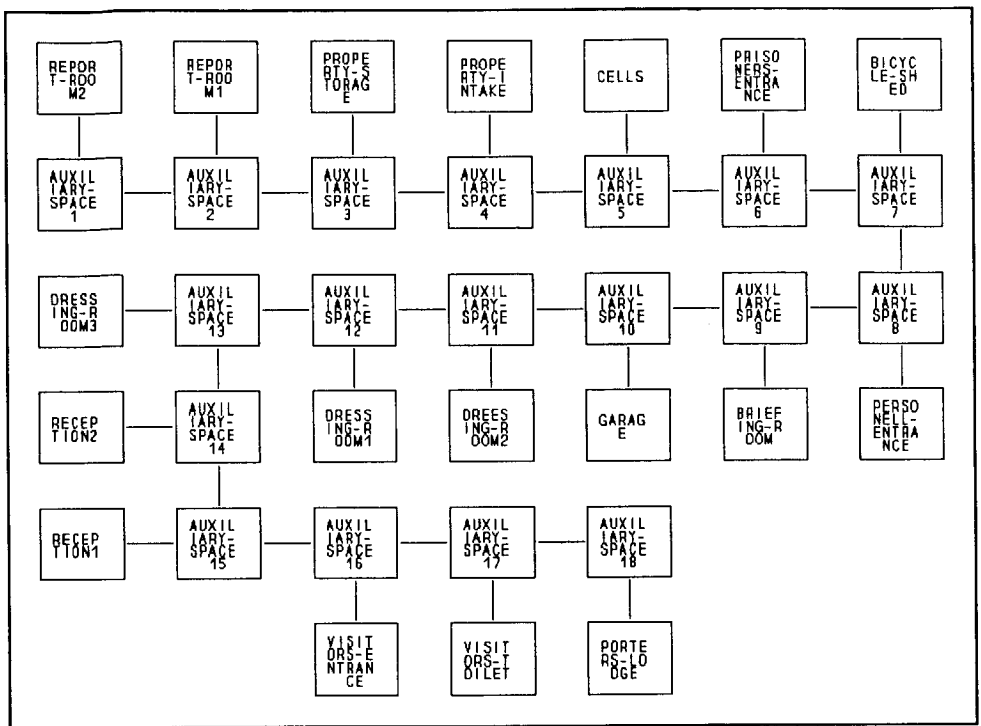


Figure 6.45: A single-loaded yardstick design with respect to the norms "community", for 2nd-floor of the police-station



6.46: A single-loaded yardstick design for 2nd-floor of the police-station

Above design is generated with respect to the privacy / circulation-cost norm(s).

Result of the **behavioral analysis** of designs:

Design	community utility	privacy cost	circulation cost
Input- design	93	23	247
community yardstick	726	0	880
privacy yardstick	398	0	552
circulation-cost yardstick	398	0	552

Evaluation result:
The *community utility* of the input-design is about 13% of a near-optimal design, which is considered *as bad*.
The *privacy-cost* is 23 units, which is higher than a near-optimal design.
The *circulation-cost* is about 55% lower than a near-optimal design, and it is *excellent*.

Third run conditions:

- Multiple corridor assumption:
- Evaluation against double-loaded yardstick designs.

- Points of view of evaluation (norms): Community, privacy, and circulation-cost.

Evaluating the first-floor:

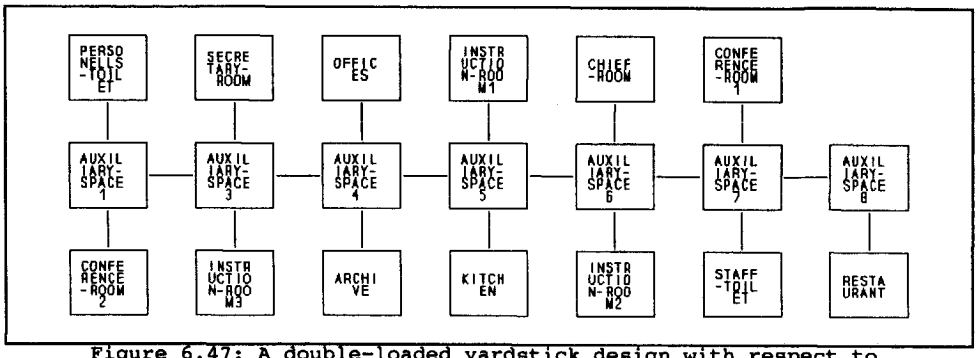


Figure 6.47: A double-loaded yardstick design with respect to the "community" norm for 1st-floor of the police-station

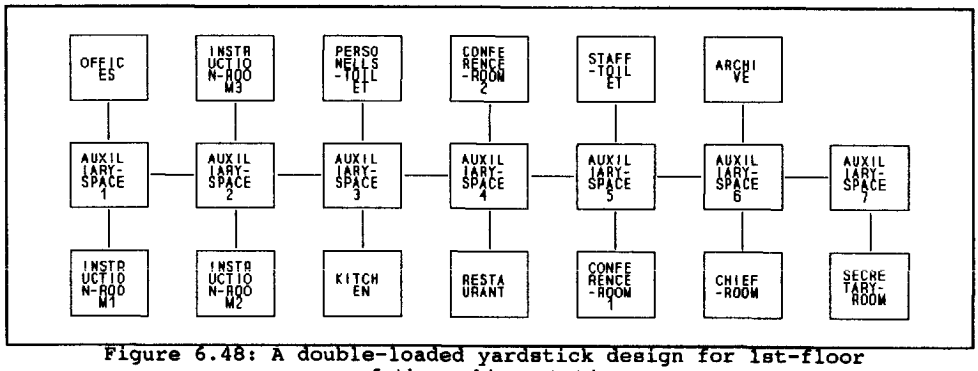


Figure 6.48: A double-loaded yardstick design for 1st-floor of the police-station

Above design is generated with respect to the Privacy / circulation-cost norm(s).

Result of the **behavioral analysis** of designs:

Design	community utility	privacy cost	circulation cost
Input- design	212	0	358
community yardstick	293	0	439
privacy yardstick	167	0	313
circulation-cost yardstick	167	0	313

Evaluation result:
The *community utility* of the input-design is about 72% of a near-optimal design, which is considered *as fair*.
The *privacy-cost* is 0 and is *excellent*.
The *circulation-cost* is about 14% higher than a near-optimal design, and it

is *excellent*.

Evaluating the second-floor:

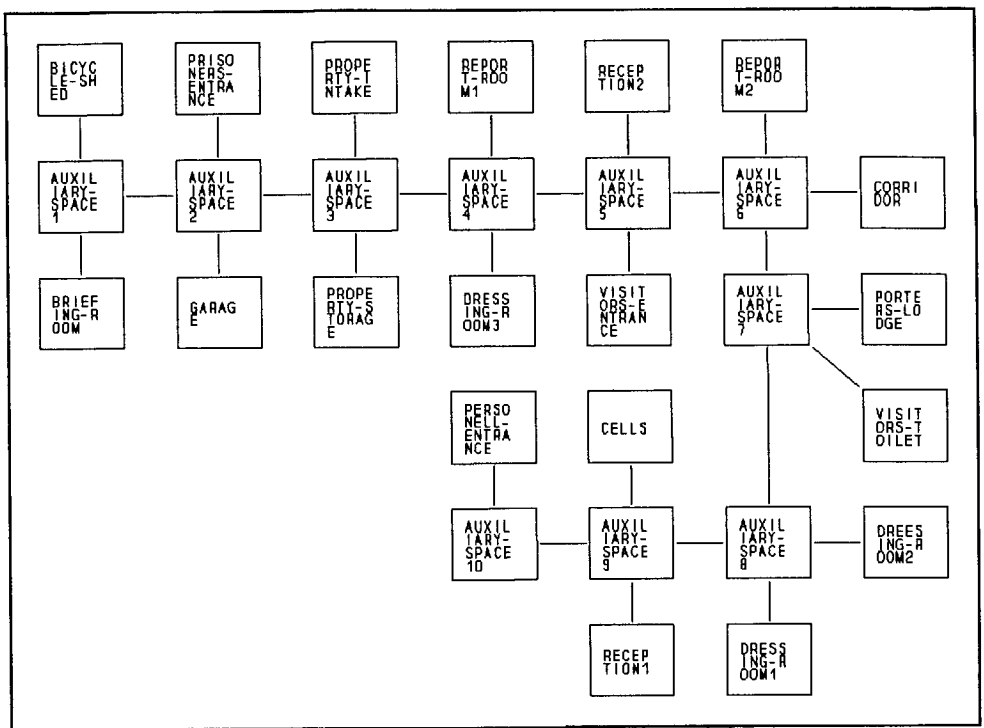


Figure 6.49: A double-loaded yardstick design with respect to the norms "community" for 2nd-floor of the police-station.

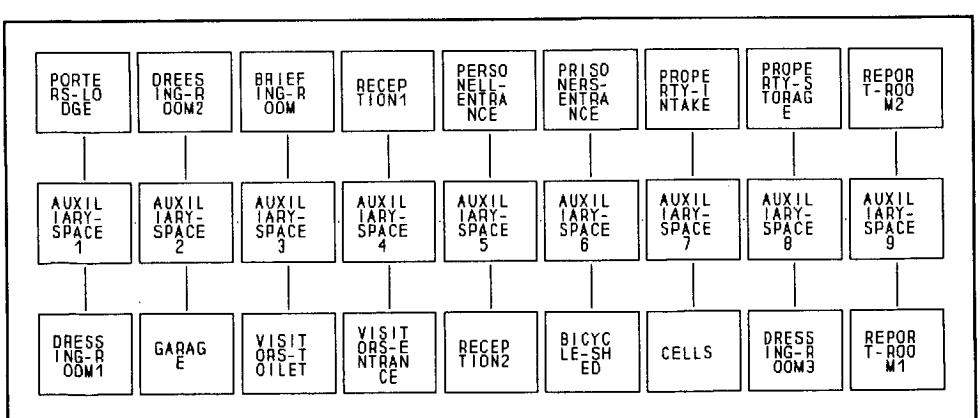


Figure 6.50: A double-loaded yardstick design with respect to the norm "privacy" "circulation-cost" for 2nd-floor of the police-station

Result of the **behavioral analysis** of designs:

Design	community utility	privacy cost	circulation cost
Input-design	267	23	421
community yardstick	368	0	522
privacy yardstick	235	0	389
circulation-cost yardstick	235	0	389

Evaluation result:
The *community utility* of the input-design is about 73% of a near-optimal design, which is considered *as FAIR*.
The *privacy-cost* is 23 units higher than near-optimal design.
The *circulation-cost* is about 8 lower than a near-optimal design, and it is *excellent*.

6.7.3 Evaluating the hospital

This section presents evaluation results of an existing design for the hospital. The existing designs for different floors of the hospital are, almost, in the form of a single-loaded or double-loaded corridors with circular-access (loops) between some locations. So, the natural way of evaluating this design is by comparing its performance against single-loaded or double-loaded design yardsticks with auxiliary locations. Following are evaluation of each floor of the hospital with respect to the social norms and against the defined yardstick designs.

Evaluating the basement:

The basement of the hospital as it was evident from the result of its Q-analysis, is a quiet floor with little interactions between its locations. This fact is reconfirmed by its performance analysis with respect to the social norms. Here is the existing design for this floor.

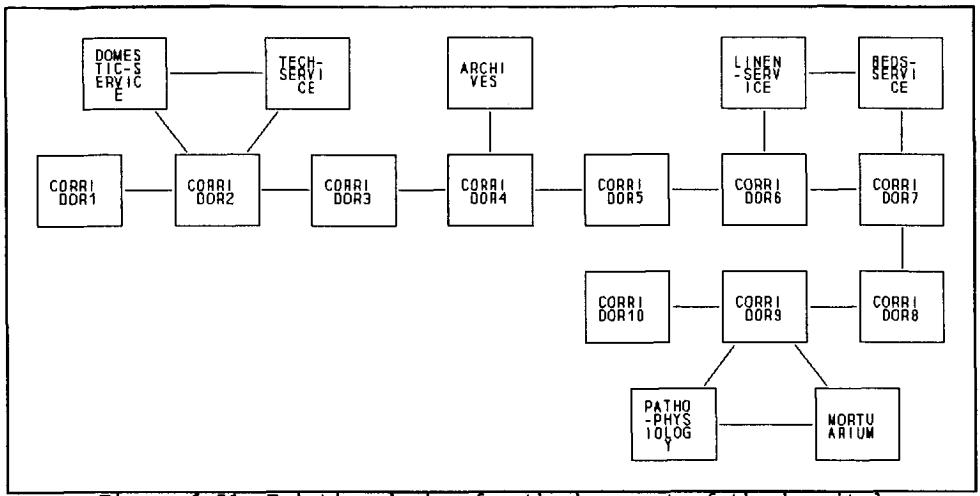


Figure 6.51: Existing design for the basement of the hospital

Evaluating the basement against a single-loaded design:

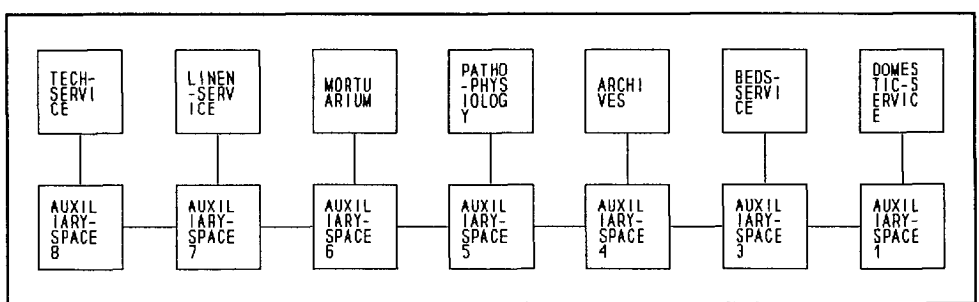


Figure 6.52: A single-loaded yardstick design with respect to the community norm generated for basement of the hospital

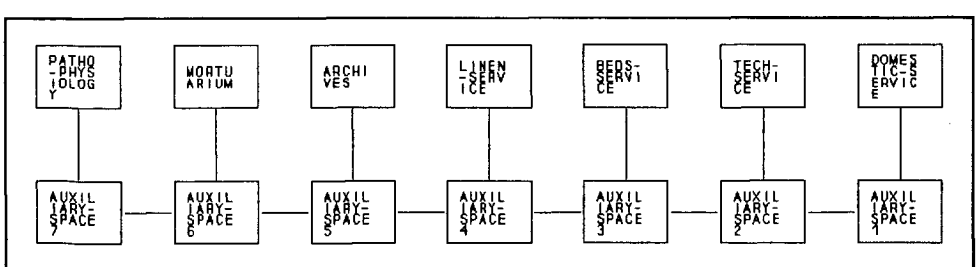


Figure 6.53: A single-loaded yardstick design with respect to the privacy / circulation-cost norm for basement of the hospital

Result of the **behavioral analysis** of designs:

Design	community utility	privacy cost	circulation cost
Input-design	0	0	10
community yardstick	0	0	45
privacy yardstick	0	0	20
circulation-cost yardstick	0	0	20

Evaluation result:
The *community utility* is 0 for both the input and the yardstick designs.
The *privacy-cost* is 0, and it is *excellent*.
The *circulation-cost* is about 50% lower than the yardstick design, and it is *excellent*.

Evaluating the basement against a double-loaded design:

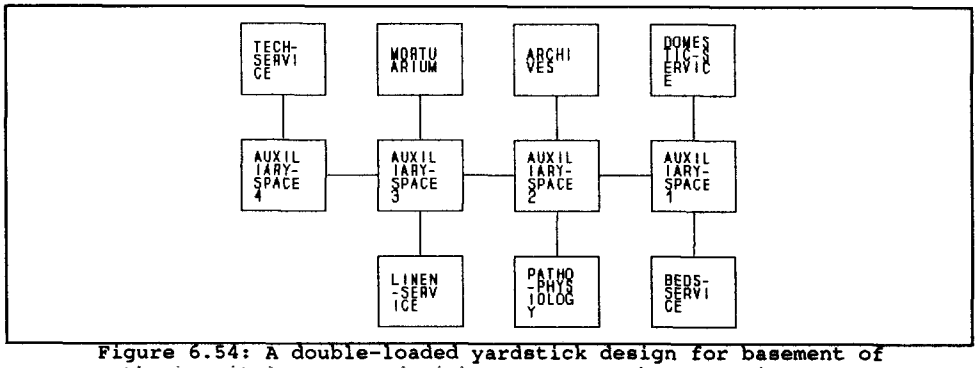


Figure 6.54: A double-loaded yardstick design for basement of the hospital generated with respect to the community norm

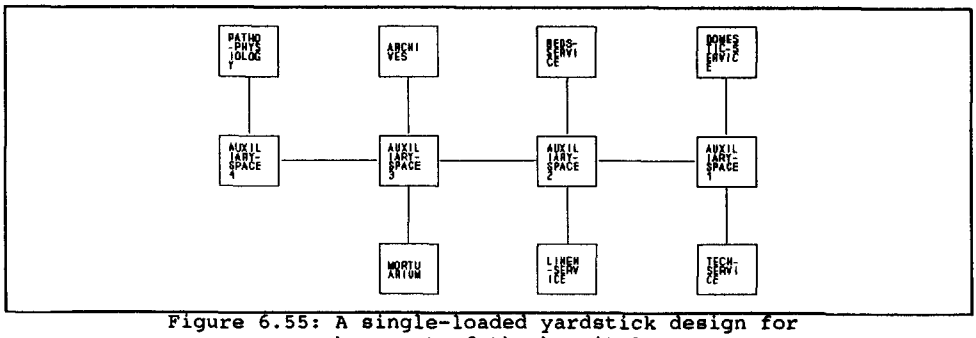


Figure 6.55: A single-loaded yardstick design for basement of the hospital

Above design is generated with respect to the privacy / circulation-cost norm(s).

Result of the **behavioral analysis** of designs:

Design	**community utility**	**privacy cost**	**circulation cost**
Input-design	0	0	10
Community yardstick	0	0	30
Privacy yardstick	0	0	15
Circulation-cost yardstick	0	0	15

Evaluation result:
The *community utility* for both the input and the yardstick design is 0.
The *privacy-cost* is 0, which is *excellent*.
The *circulation-cost* is about 23% lower than the yardstick design, and it is *excellent*.

Evaluating the ground-floor:

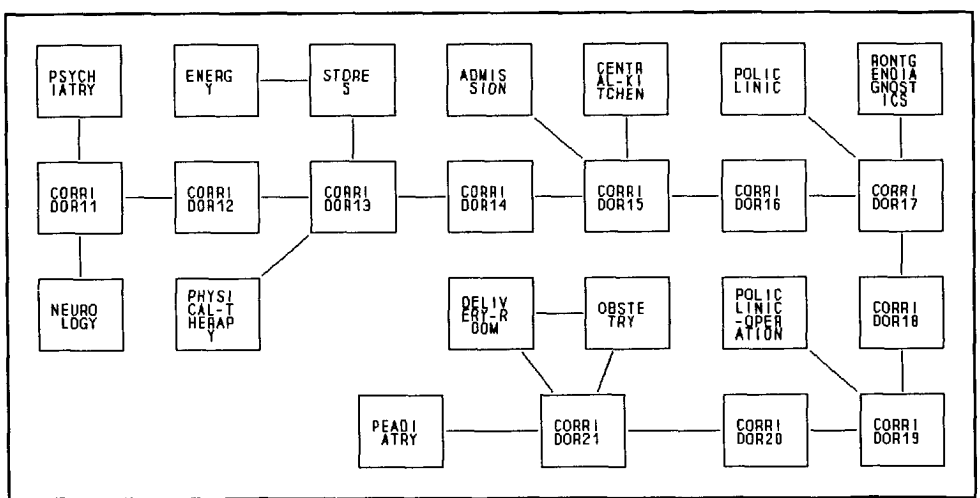


Figure 6.56: Existing design for ground-floor of the hospital

Evaluating the ground-floor against a single-loaded design:

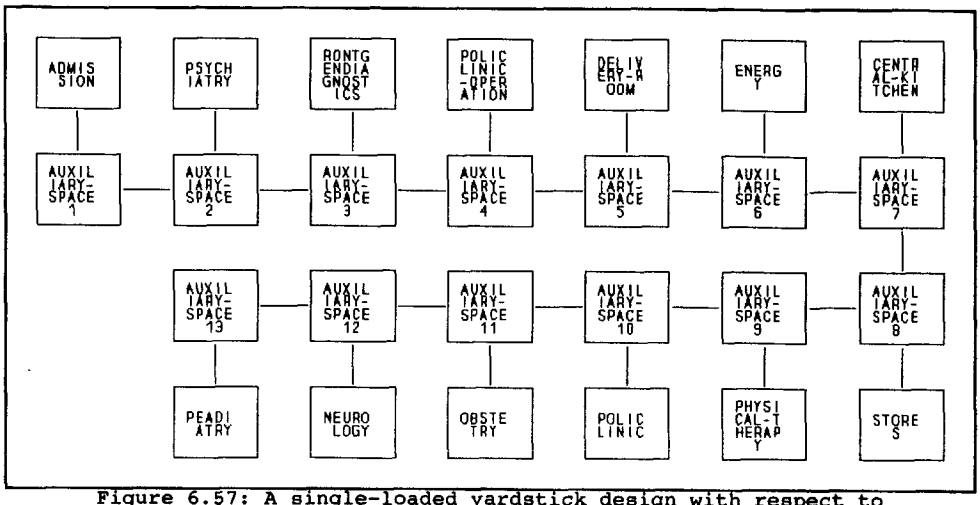


Figure 6.57: A single-loaded yardstick design with respect to the community norm generated for ground-floor of the hospital

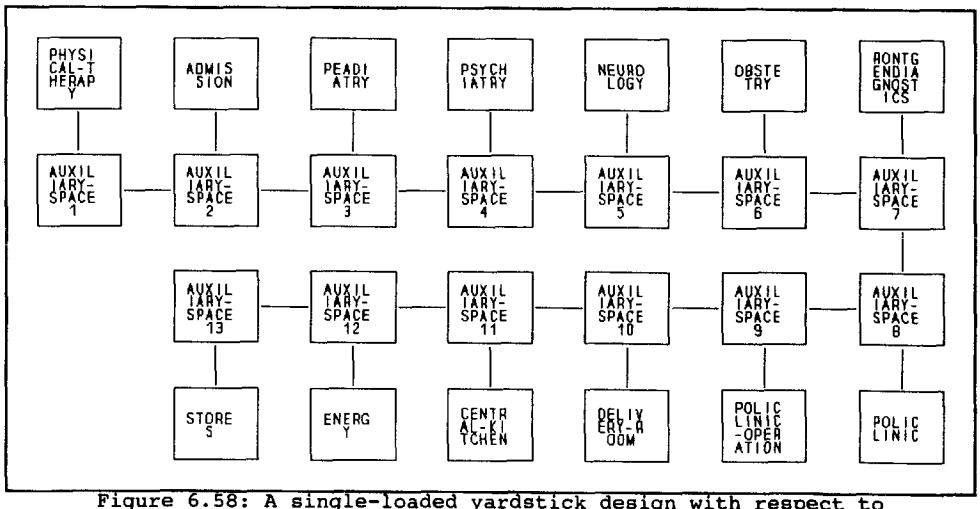


Figure 6.58: A single-loaded yardstick design with respect to the privacy / circulation-cost norm(s) for the round-floor

Result of the **behavioral analysis** of designs:

Design	community utility	privacy cost	circulation cost
Input-design	4445	0	5807
Community yardstick	5033	0	6395
Privacy yardstick	2082	0	3444
Circulation-cost yardstick	2082	0	3444

Evaluation result:
The *community utility* of the input-design is about 88% of the yardstick design, and it is *good*.
The *privacy-cost* is 0, which is *excellent*.
The *circulation-cost* is about 69% higher than the generated yardstick design, which is *medium*.

Evaluating the ground-floor against a double-loaded design:

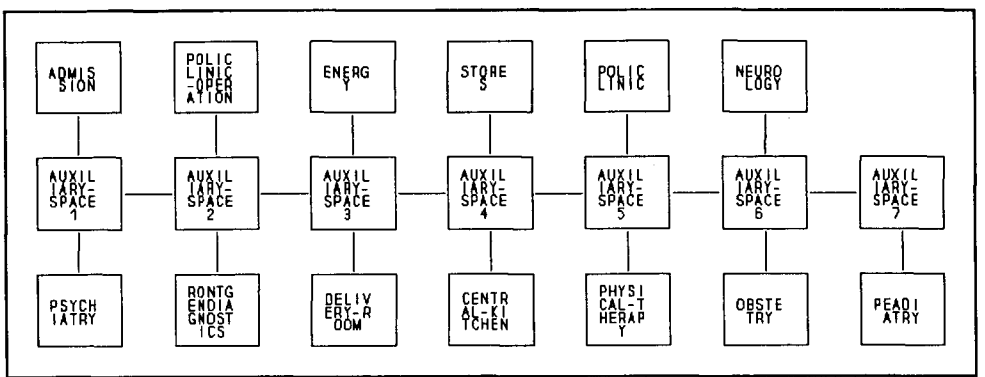


Figure 6.59: A double-loaded yardstick design with respect to the community norm generated for the ground-floor of hospital

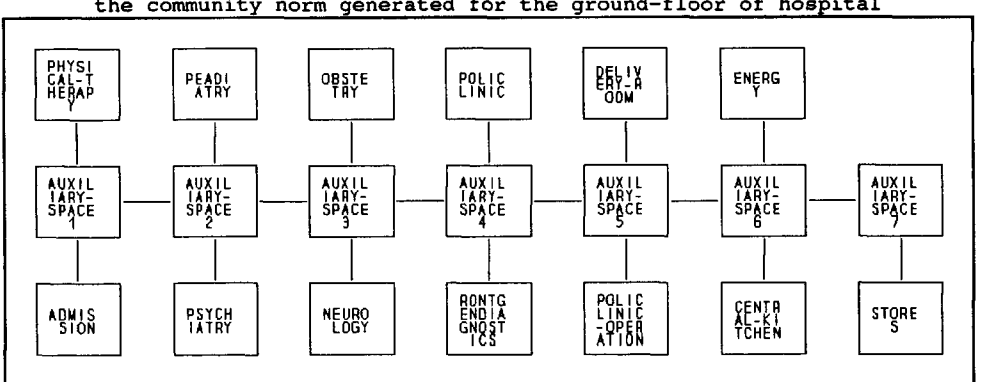


Figure 6.60: A single-loaded yardstick design with respect to the privacy / circulation-cost norm(s) for the ground-floor

Result of the **behavioral analysis** of designs:

Design	community utility	privacy cost	circulation cost
Input-design	4445	0	5807
Community yardstick	2887	0	4249
Privacy yardstick	1365	0	2727
Circulation-cost yardstick	1365	0	2727

Evaluation result:
The *community utility* of the input-design is about 154% of the yardstick design, and it is *excellent*.
The *privacy-cost* is 0, which is *excellent*.

The *circulation-cost* is about 113% higher than the generated yardstick design, which is very *bad*.

Evaluating the first-floor:

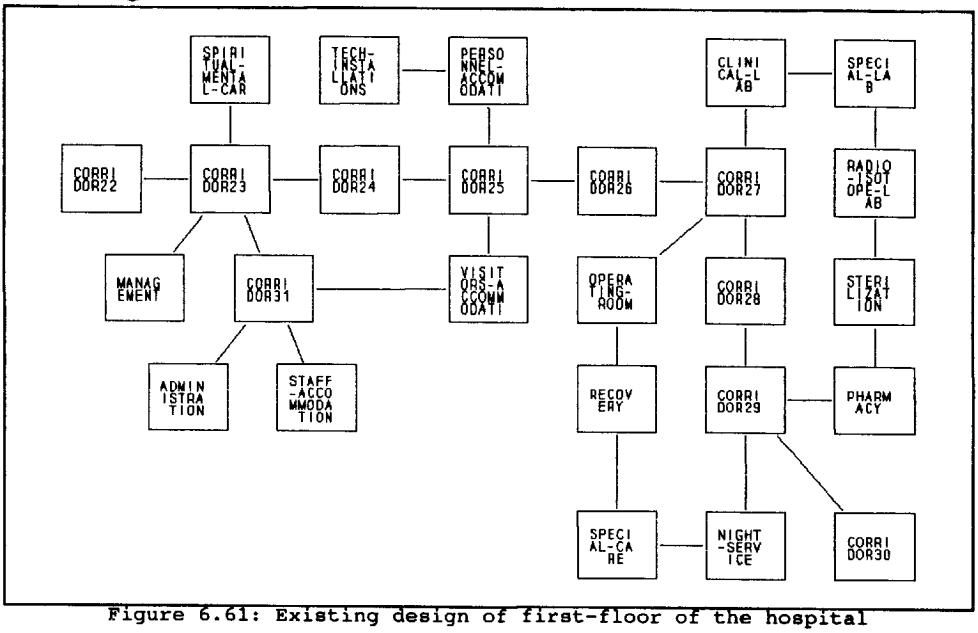


Figure 6.61: Existing design of first-floor of the hospital

Evaluating the first-floor against a single-loaded design:

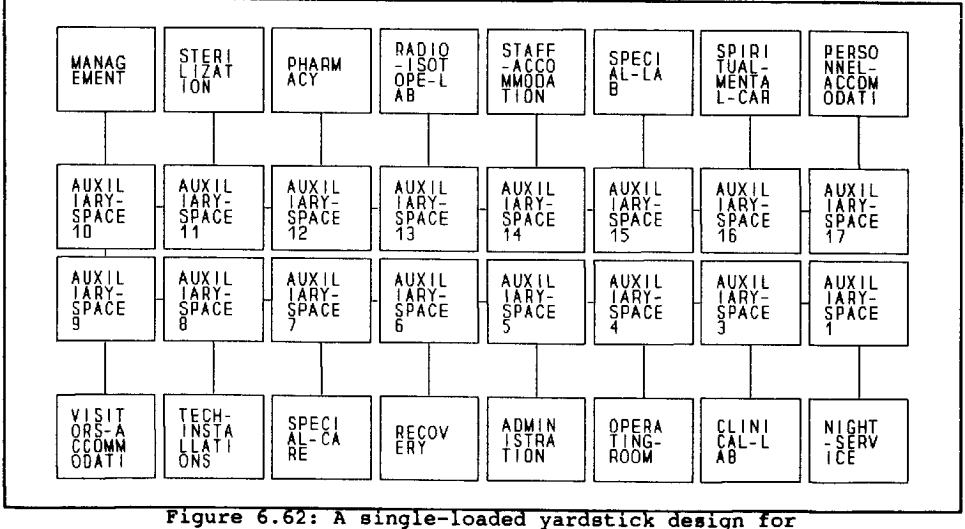


Figure 6.62: A single-loaded yardstick design for 1st-floor of the hospital

Above design is generated with respect to the community norm.

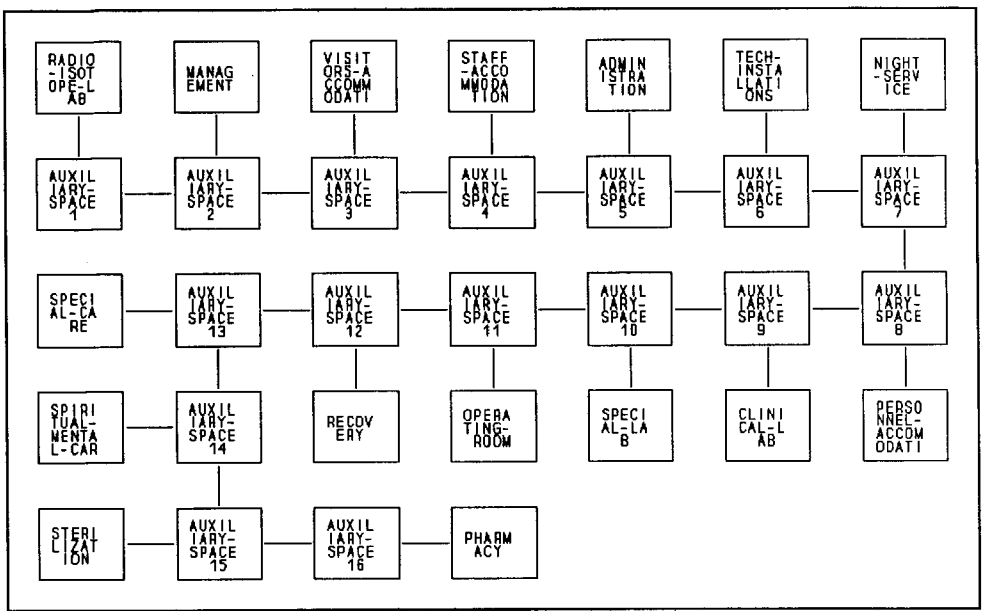


Figure 6.63: A single-loaded yardstick design with respect to the privacy / circulation-cost norm(s) for 1st-floor of the hospital

Result of the **behavioral analysis** of designs:

Design	community utility	privacy cost	circulation cost
Input-design	1474	240	2422
Community yardstick	5141	0	6089
Privacy yardstick	1814	0	2762
Circulation-cost yardstick	1814	0	2762

Evaluation result:
The *community utility* of the input-design is about 29% of the yardstick design, and it is very *bad*.
The *privacy-cost* is 240 unit, which is higher than the generated yardstick design.
The *circulation-cost* is about 12% lower than the generated yardstick design, and it is *excellent*.

Evaluating the first-floor against a double-loaded design:

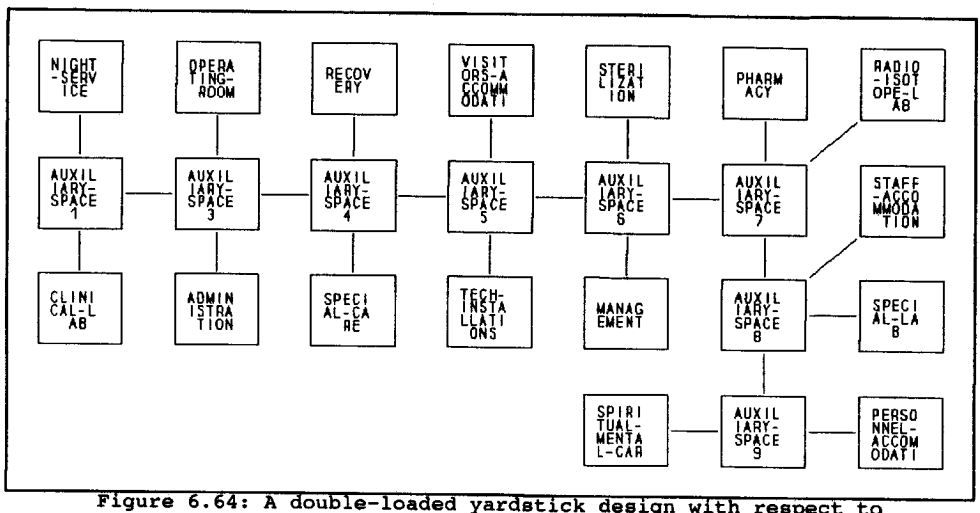


Figure 6.64: A double-loaded yardstick design with respect to the community norm for 1st-floor of the hospital

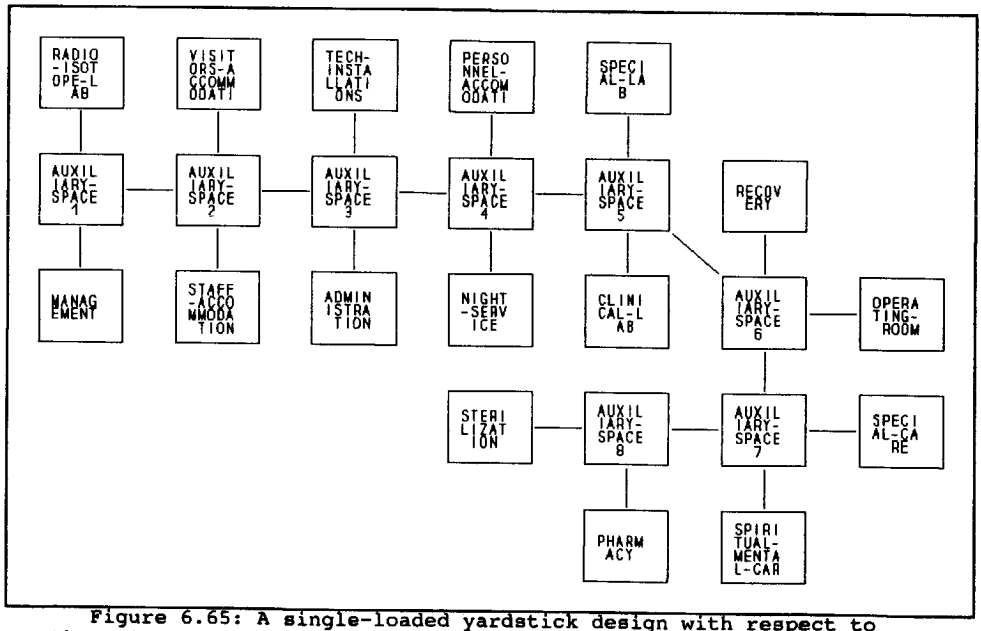


Figure 6.65: A single-loaded yardstick design with respect to the privacy / circulation-cost norm(s) for 1st-floor of the hospital

Result of the **behavioral analysis** of the designs:

Design	community utility	privacy cost	circulation cost
Input-design	1474	240	2422
Community yardstick	2703	0	3651
Privacy yardstick	1132	0	2080
Circulation-cost yardstick	1132	0	2080

Evaluation result:
The *community utility* of the input-design is about 55% of the yardstick design, and it is *medium*.
The *privacy-cost* of the input-design is 240, which is higher than the generated yardstick design.
The *circulation-cost* is about 16% higher than the generated yardstick design, and it is *excellent*.

Evaluating the second-floor:

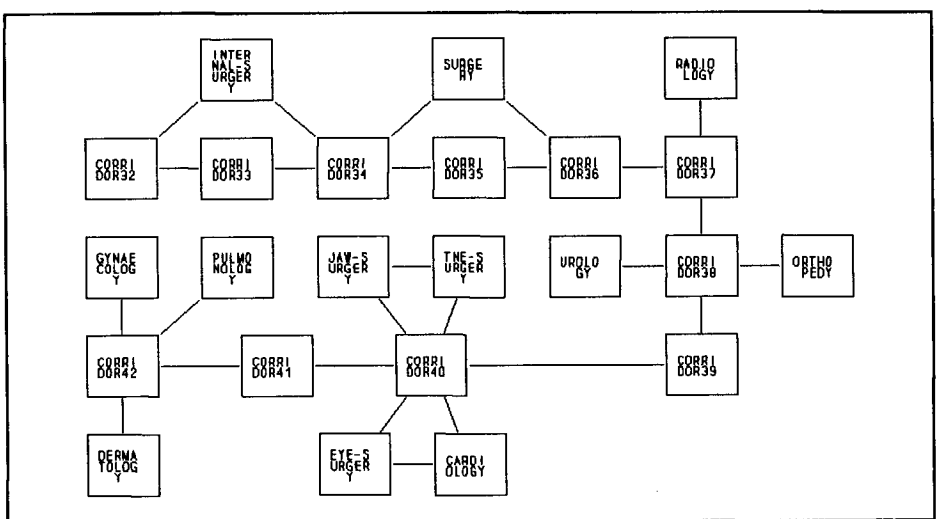


Figure 6.66: Existing design for 2nd-floor of the hospital

Evaluating the second-floor against a single-loaded design:

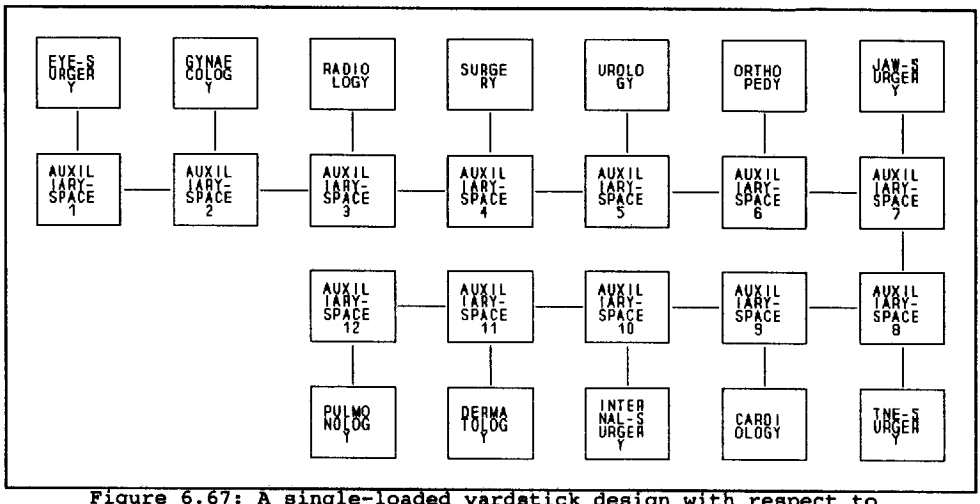


Figure 6.67: A single-loaded yardstick design with respect to the community norm for 2nd-floor of the hospital

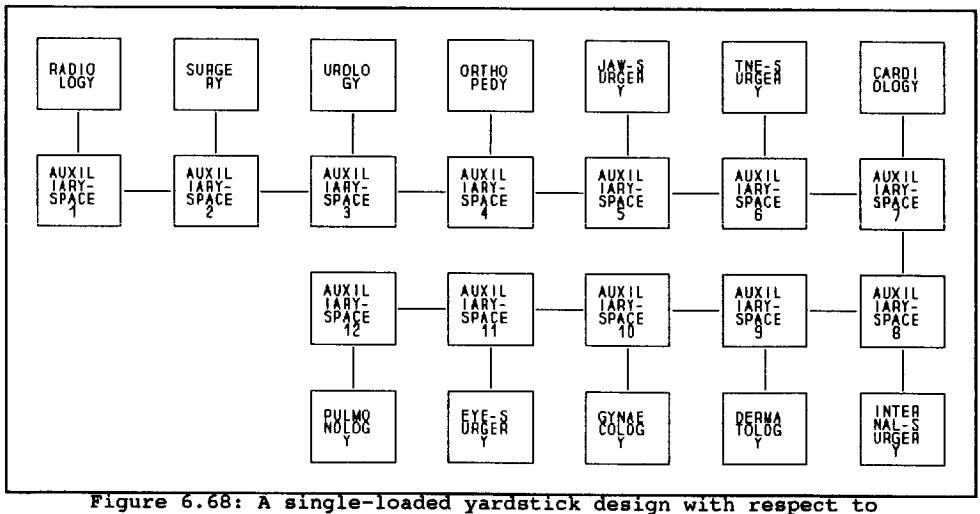


Figure 6.68: A single-loaded yardstick design with respect to the privacy / circulation-cost norm(s) for 2nd-floor of the hospital

Result of the **behavioral analysis** of the designs:

Design	community utility	privacy cost	circulation cost
Input-design	19703	278.5	21544
Community yardstick	19703	0	27057
Privacy yardstick	19509	0	26863
Circulation-cost yardstick	19509	0	26863

Evaluation result:
The *community utility* of the input-design is about 72% of the yardstick design, and it is *fair*.
The *privacy-cost* is 278 unit, which is higher than the yardstick design.
The *circulation-cost* is about 20% higher than the generated yardstick design, which is *excellent*.

Evaluating the second-floor against a double-loaded design:

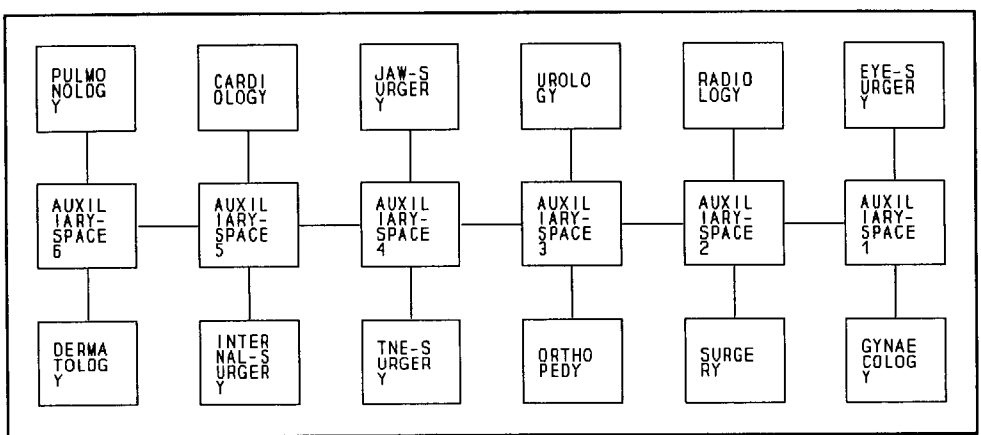


Figure 6.69: A double-loaded yardstick design for 2nd-floor of the police-station

Above design is generated with respect to the community norm.

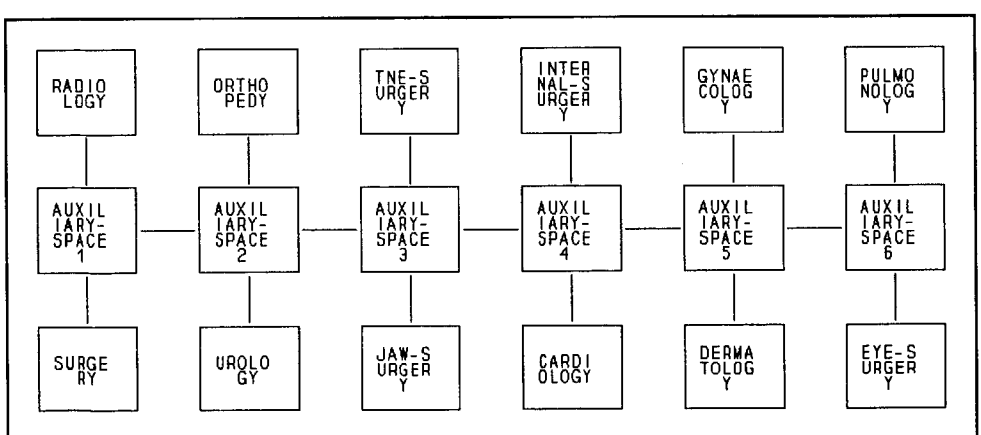


Figure 6.70: A single-loaded yardstick design for 2nd-floor of the police-station

Above design is generated with respect to the privacy / circulation-cost norm(s).

Result of the **behavioral analysis** of designs:

Design	community utility	privacy cost	circulation cost
Input-design	14190	278.5	21544
Community yardstick	11521	0	18875
Privacy yardstick	11427	0	18751
Circulation-cost yardstick	11427	0	18751

Evaluation result:
The *community utility* of the input-design is about 123% of the yardstick design, and it is *excellent*.
The *privacy-cost* of the input-design is 278.5 unit, which is higher than the generated yardstick *design*.
The *circulation-cost* is about 15% higher than the generated yardstick design, which is *excellent*.

6.8 Summary

This chapter presented examples of designs that was generated by TOPGENE based on three different design data-sets. These examples covered designs with respect to single social norms, as well as combinations of them. An analysis of complete sets of design types with respect a single or a combination of norms was presented at the end of each section. These analysis confirmed that TOPGENE as a heuristic program has a rational performance behavior , which is in conformity with our intuition and our expectation from the system. Comparisons of different design types for a data-set shows that the performance of these designs relative to each other are in harmony with their structural and operational characteristics. These two in turn, are in concordance with the norm under which they are generated.

Chapter 8 presents further testing of TOPGENE. This chapter examines TOPGENE designs against designs generated by a neural network for the same set of problems.

CHAPTER 7
NEURAL MODELING OF ARCHITECTURAL DESIGN

This chapter discusses the implementation of a special case of the architectural design (i.e., the linear-tree type design with respect to a single social norm community or privacy) in a neural network model. The idea in this chapter is borrowed from the Hopfield model implementation of the Travelling Salesman Problem (TSP). The reason is the similarity between the architectural design (AD) at topological level with the TSP. Test results are satisfactory, and exploration of larger classes of architectural design with respect to combined norms are possible.

7.1 Hopfield model

The work of Hopfield and Tanks [Hopfield82] [84] [85] [86] is classified as auto associative and evolutionary models of neural networks [Zeidenberg90]. The significance of their work is the introduction of the idea of an energy function in modeling a network. This idea relates their network to physical systems [Hopfield82].

A Hopfield net is a fully connected network, in which nodes represent "state space of knowledge". The way the system evolves is based on the magnitude of the relations between the nodes, and an equation of motion that governs the behavior of the network over time. The equation of motion is derived from an energy function equation. If the system starts with partial (random) knowledge, it eventually settles into complete knowledge through an iterative process. The initial state of the Hopfield model is a chaotic situation for the total network. After the initialization of the nodes with

the input values, the network iteratively calculates the successive values for the nodes. Each node tries to effect and turn the other nodes on. As the process continues, the effect of different weights associated with the links between the nodes tries to calm the network down, and gradually bringing it in a compromised equilibrium situation. When the process is finalized, the value of each node is checked against a threshold value for acquiring the result.

The Hopfield model is different from other neural network models in the way that it produces a solution. The nodes in the Hopfield model are all similar in terms of their positions in the total nets, and there is no distinction between them. Each node i in this model has a threshold value U_i and a step-function. A node i resets itself to a value V_i based on the following step-function rule:

$$V_i = \begin{cases} 1, & \text{if: } \sum w_{ij} v_j \geq U_i \\ 0, & \text{if: } \sum w_{ij} v_j < U_i \end{cases} \qquad \text{where, } \sum w_{ij} v_j \text{ is the input sum.}$$

The weight w_{ij} is associated to the connections between the nodes i and j, and could be symmetrical for both nodes. A node i thus calculates its output value V_i based on the weighted sum of the input signals and the threshold value by passing the sum of the weighted inputs minus their threshold value to the step function. This is the same as saying that the nodes fire only when appropriate, and not all simultaneously. In other words the system behavior is similar to local brain behavior in an asynchronous way [Zeidenberg90].

In the Hopfield model the calculation process is sequentially in one computational sweep applied to all the nodes, and checked if they fire or not. For example the initial inputs to the network are supplied for all the nodes at once, and the process continues until it reaches a stable state in which further improvement is not evident in the status of the network. The output of the Hopfield model is derived upon the final value of the whole network.

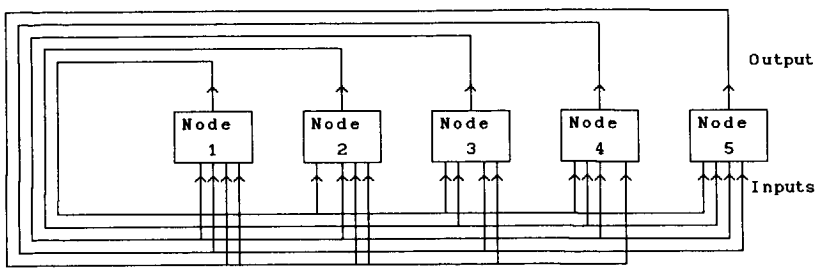


Figure 7.1: The Hopfield Model with 5 neurons

7.2 Hopfield model for the Travelling Salesman Problem (TSP)

Hopfield demonstrated his model through the use of the TSP problem. His approach towards modeling this problem was based on the representation of a n city problem with an n by n permutation matrix in which a row represents a city and a column the position of a city on the tour. For example, the representation of a 4-cities B-D-A-C-E tour will look like:

	Position				
City	1	2	3	4	5
A	0	0	1	0	0
B	1	0	0	0	0
C	0	0	0	1	0
D	0	1	0	0	0
E	0	0	0	0	1

In the actual implementation, such a matrix represents the voltage for the neurons, in a way that each entry v_{xi} denotes the output voltage of a neuron corresponding to the x-th row and i-th column. This matrix is continuously updated according to the governing equation of motion until a termination criterion is confronted after which the voltage for each neuron (i.e, matrix elements) is interpreted as 0 or 1 for inferring the tour sequence (i.e topology) for the problem.

7.3 The TSP energy function

The evolutionary behavior of the network is governed by an energy function in which the minimum corresponds to the best solution. Such an energy func-

tion should consider the topological property of a closed tour on the permutation matrix as well as the minimum distance condition on the tour. To ensure these two conditions, Hopfield gave the following equation for the energy function:

$$E = (A/2) \sum_x \sum_i \sum_{j \neq i} V_{xi} V_{xj} + (B/2) \sum_i \sum_x \sum_{y \neq x} V_{xi} V_{yi} + (C/2) \left(\sum_x \sum_i V_{xi} - n\right)^2$$

$$+ (D/2) \sum_x \sum_{y \neq x} \sum_i d_{xy} V_{xi} \left(V_{y,i+1} + V_{y,i-1}\right) \qquad (1)$$

In this equation:

- The first term is 0, if each row in the matrix contains a single 1.
- The second term is 0, if each column in the matrix contains only a 1.
- The third term is 0, if the total number of 1s in the matrix is exactly n.
- The fourth term is minimum if a shortest tour is chosen a solution.

Furthermore, d_{xy} is the distance between two cities x and y, and A, B, C, and D are positive parameters. These parameters are important in terms of the behavior of the network and must be chosen as appropriate as possible to the problem. Experimenting with the network and deep understanding of its behavior helps in appropriate determination of these parameters for a particular problem. The values of these parameters in the Hopfield model are A=B=500, C=200, and D=500.

Hopfield used the derivative of the energy function to arrive at an expression for determining the weights for the connection between a neuron pair $(x_i\ y_j)$. The connection weights are then used to generate the equation of motion (2) for the input u_{xi} to a particular neuron N_{xi}.

$$\frac{d}{dt} u_{ij} = -\left(\frac{u_{ij}}{\tau}\right) - A \sum_{j \neq i} V_{xj} - B \sum_{y \neq x} V_{yi} - C\left(\sum_x \sum_j V_{xj} - n\right) - D \sum_y d_{xy} \left(V_{y,i+1} + V_{y,i-1}\right) \qquad (2)$$

This equation (2) contributes to the determination of the input value of each neuron (3) at each iteration, which in turns determines the output voltage of a neuron according to the equation (4) acting as a step-function:

$$u_{xi}^{t+1} = u_{xi}^{t} + \frac{d}{dt} u_{xi}^{t} \qquad (3)$$

$$v_{xi} = g(u_{xi}) = \frac{1}{2} \left[1 + \tanh\left(\frac{u_{xi}}{u_0}\right)\right] \qquad (4)$$

where, $u_0 = 0.02$

The initial value of the neurons is chosen so that the total values sums up to n (i.e., the number of the neurons). However, for keeping the network from being trapped in an initial equilibrium situation, random noise is added to each neuron value as a starting force.

Hopfield and Tank [Hopfield85] proposed erroneously that a value of 1 can be used for interval time τ without loss of generality; but, Wilson and Pawley [Wilson88] warned that τ=1 causes destruction of the initial voltage value of each neuron, and the oscillation of the neuron voltage without ever converging. This is quit clear from the fact that if we substitute equation (2) in equation (3), then we have:

$$u_{xi}^{t+1} = u_{xi}^{t} - [\frac{du_{xi}^{t}}{\tau} + \text{other terms}] \qquad (5)$$

and if we take τ as 1, then the first term annihilates the second term. For this reason Wilson and Pawley proposed a fraction value such as 10^{-5} for τ.

7.4 Generating linear-tree designs with respect to a norm

In this section the implementation issues concerning generating linear-tree designs with respect to a single norm is discussed. The combination of norms is a separate issue, and is subject to further investigation. The norms privacy and circulation costs are in harmony with each other and demands the same treatment, while the norm community conflicts with both of them. The final point to be made is that since we have been after an approach rather than a working system, all parameters discussed in the formal definition of the norms, except the flow is assumed to be 1. So, the input to the model is a matrix representing the flow-rate between different location-pairs in a building. Obviously, the effect of other parameters may be enforced upon the matrix before its input to the network.

To represent the design, we will use the Hopfield approach for the travelling salesman problem. The network for a design problem with n numbers of locations is represented as a n by n matrix that its rows denote the locations of the building and columns denote the position of locations with respect to each other.

7.4.1 Heuristics

Remind that a design optimized with respect to community norm is a pattern labeled with the activities in such a way that it creates maximum flow overlap on each location, while the other two norms demand on the contrary. These properties of designs have important rôles in the modeling of the problem in a network. The maximization or minimization of flow overlap and consideration of design at topological level brings about a considerable resemblance between the ADP and the TSP. In the case of the TSP one is looking for the shortest closed path (a tour) that passes all cities, while in the case of ADP the search will be directed towards an open path with minimum or maximum flow overlaps. Besides the heuristics discussed may be included in the network to enforce early settlement of the network to a solution. For example based on the heuristics discussed we can assign the most interactive locations as the end locations of a linear-tree design in the case of the community norm. The least interactive location-pairs may be chosen for the case of the privacy norm. This rule of thumbs decide on the positions of two locations before the network start a computation.

7.4.2 The initial condition

Theoretically the initial input voltage values of the neurons should be taken in a way that there is no bias in favor of any particular solution. According to Hopfield and Tank these values should be chosen in a way that the sum of the final voltage value is approximately equal to n, the number of neurons in the network. that is:

$$\sum_{x} \sum_{i} V_{xi} = n \qquad (6)$$

From equation (6) we can derive the initial value of each neuron as:

$$u^{0}_{xi} = -\frac{u_0}{2} \ln (n - 1), \text{ where } u_0 = 0.02 \qquad (7)$$

This insures that the initial voltage to all neurons sums up to n. A small uniform random noise must be added to each neuron in order to keep the network from being trapped in an unstable initial equilibrium situation. This random noise for each neuron was added as is recommended by Wilson and Pawley [Wilson88] as follows:

$$-0.1\, u_0 \leq \delta u_{xi} \leq 0.1\, u_0 \qquad (8)$$

7.4.3 Energy functions

It was mentioned that in general our design problem is similar to the TSP, if a linear-tree type design is in prospect. This type of design is topologically the same as solutions to the TSP. We have to describe each model in terms an energy function that embodies the properties of the designs in mind. The TSP energy function, thus, can provide us with the initial idea for devising such energy functions. I propose the following energy function for our design problems with respect to the community norm:

$$E_c = \frac{A}{2} \sum_x \sum_i \sum_{j \neq i} V_{xi} V_{xj} + \frac{B}{2} \sum_x \sum_{y \neq x} \sum_i V_{xi} V_{yi} + \frac{C}{2} \left(\sum_x \sum_i V_{xi} - n \right)^2$$

$$+ \frac{D}{2} \left(\sum_x \sum_{y \neq x} \sum_i \sum_{j \neq i} f_{xy} (|i - j| - 1) V_{xi} V_{yj} \right)^{-1} \qquad (9)$$

As in the TSP, the first term, here, forces a single 1 in a row, the second term is for having a single 1 in each column, and the third term is additional constraint for having exactly n 1s (i.e., n neurons on) at the final stage. The fourth term, which is different from the corresponding term in the TSP, embraces other topological properties of designs with respect to the community norm. This term enforces the right labeling on the pattern in order to have a maximum community utility (i.e., flow overlaps) on the design pattern. The reason that we are using the reciprocal of the fourth term is that AD with respect to community norm is a maximization problem, while TSP is a minimization problem. This term gets large when interactive location pairs are far apart and gets small as the network try to locate them as far as possible. f_{xy} in this term is the actual flow potential (input) between the two locations x and y. These four terms together lead the system towards a desired solution with respect to the community norm, by deriving the energy value of the network towards a minimum value in a stable condition.

A, B, C and D are all positive constants that play a considerable rôle in the model. Hopfield and Tank used A=B=D=500, and C=200 for their model. These values were found inappropriate to our model. Experiments with the network revealed that the following values were generally in favor of the convergence of the network towards a solution for $n \leq 15$. I am, thus, proposing a problem dependent D parameter for the model.

A = 500

B = 500

C = 100

D = 100,000 * n

The derivative of the energy function can be used to derive the strength of connection between a neuron pair x_i and y_j. This derivative has rôle in generating the equation of motion (10) for the input u_{xi}, to a particular neuron.

$$\frac{du_{xi}}{dt} = -\frac{u_{xi}}{\tau} - A\sum_{j\neq i} V_{xj} - B\sum_{y\neq x} V_{yi} - C\left(\sum_x\sum_j V_{xj} - n\right)$$

$$+ D\left(\sum_{y\neq x}\sum_{j\neq i} f_{xy}(|i-j|-1)\, V_{yj}\right)\left(\sum_x\sum_{y\neq x}\sum_i\sum_{j\neq i} f_{xy}(|i-j|-1)\, V_{xi}V_{yj}\right)^{-2} \qquad (10)$$

$$\text{where } V_{xi} = g(u_{xi}) = \frac{1}{2}\left(1 + \tanh\left(\frac{u_{xi}}{u_0}\right)\right) \qquad (11)$$

$$\text{and } u_{xi}^{t+1} = u_{xi}^{t} + \frac{du_{xi}}{dt} \qquad (12)$$

where, u_{xi}^{t+1} is the input voltage of the neuron N_{xi} on the next interval.

Similarly, we can describe the energy function and its corresponding equation of motion for a design with respect to the privacy and/or circulation-cost norm(s). This energy function is similar to the previous function for community norm except for the fourth term. We have to devise the energy function so that it corresponds to a linear-tree solution that has most interactive activities close to each other, and there is minimum flow overlap (intervention) on the pattern. Equation (13) is the energy function for the privacy / circulation-cost sub-module.

$$E_P = \frac{A}{2}\sum_x\sum_i\sum_{j\neq i} V_{xi}V_{xj} + \frac{B}{2}\sum_x\sum_{y\neq x}\sum_i V_{xi}V_{yi} + \frac{C}{2}\left(\sum_x\sum_i V_{xi} - n\right)^2$$

$$+ \frac{D}{2}\sum_x\sum_{y\neq x}\sum_i\sum_{j\neq i} f_{xy}(|i-j|-1)\, V_{xi}\, V_{yj} \qquad (13)$$

Where A, B, C and D are all positive parameters, and f_{xy} is the flow potential between the locations x and y. Experiments with the network showed that the following value were generally appropriate for this model.

A = 500

B = 500

C = 200

D = 100 / n

Again the derivative of the energy function is used to fix the strength of connection between a neuron pair x_i and x_j. The equation of motion for the privacy and/or circulation-cost norms(s) (14) for the input u_{xi} to a particular neuron N_{xi} is:

$$\frac{du_{xi}}{dt} = -\frac{u_{xi}}{\tau} - A\sum_{j\neq i} V_{xj} - B\sum_{x\neq y} V_{yi} - C\Big(\sum_{x}\sum_{j} V_{xj} - n\Big) - D\sum_{y\neq x}\sum_{j\neq i} f_{xy}(|i-j|-1)V_{yj} \qquad (14)$$

$$\text{where, } V_{xi} = g(u_{xi}) = \frac{1}{2}\left[1 + \tanh\left(\frac{u_{xi}}{u_0}\right)\right] \qquad (15)$$

$$\text{and } u_{xi}^{t+1} = u_{xi}^{t} + \frac{du_{xi}}{dt} \qquad (16)$$

7.4.4 Why dynamic parameters?

Experiments, also analysis of the network's behavior showed that choosing the right parameters is extremely important in well performance of the network. Wrong choices for the parameters have a range of unwanted affects, consisting of:

- A partial solution (The network does not completely converge).
- A poor solutions that is not optimal or is in the neighborhood of an optimal solution.
- Excessive iterations.
- Chaotic behavior in the network (e.g., oscillation of the neurons' values).

For example, choosing a large set of parameters (e.g., A=B=1000, C=500, and D=10000) for the community module, or a set of too small parameters (e.g., A=B=C=D=100) for the privacy mostly resulted in partial solutions with often missing end locations. A fixed value for D parameter, as in the case of Hopfield model, also caused oscilation or excessive iterations specially for large size problems.

A close look at the fourth terms in our cases signifies that the value of these terms decreases in the case of the community norm, and increases in the case of the privacy norm. The rates of changes, here, are higher than the rate of changes of similar terms in the TSP. The reason is that the fourth term in TSP relies on the value of 2n neurons, while in the AD the fourth term is calculated based on the values of all neurons in the network (i.e., n^2). Consequently, the rate of change of the fourth term for different problem sizes in AD is much higher than the TSP. Leaving the fourth term without compensation of appropriate values of the D parameter for different size of problems results in the loss of the influence of the fourth term. To tune the D parameter I altered the parameter D with the changes in problem size. Limited experiments with the network revealed that D=100,000(n) for the community module, and D=100/n for the privacy module are appropriate choices. It is possible to performance of the network by trying to find values of these parameters automatically without human intervention or the influence of human judgment.

7.4.5 Updating rules

As was described previously the equation of motion for a neuron u_{ij} at a time t+1 is determined by equation (3) as follows:

$$u_{xi}^{t+1} = u_{xi}^{t} + \frac{d}{dt} u_{xi}^{t}$$

where, $\frac{du_{xi}^{t}}{dt} = (\frac{u_{xi}}{\tau})$ + other terms/τ was appropriate for our case.

where τ is the time interval for the recalculation of neuron behavior. This parameter was set to 10^5 for our model.

7.4.6 Termination criteria

In order to stop the network at appropriate time three termination criteria are suggested by Wilson and Pawley [Wilson88]:

1- The discovery of a valid tour. This condition occurs when there is only a single 1 in each row and a single 1 in each column of the solution matrix. To detect this condition a binary threshold test

can be applied to the neuron's voltage. For example any neuron that its value is greater than 0.1 can be considered as *on* and otherwise as *off*. This test detects the convergence of the network to a solution, and halts it from further process.

2- The second termination criterion is the freezing condition. This is the case when the network reaches a rather stable state and no considerable change is possible in the status of the network. Such condition can be detected by testing the changes in the output values of the neurons. In our case the changes in the value of neurons are tested against a 10^{-35} and the network is halted if the change in the value of every neuron is less than this value.

3- The third stopping criterion was the time-out test. The network was stopped after 200 iterations, and the output values of neurons were interpreted for a solution pattern. This value was set to 1000 in the implementations of the TSP by Hopfield and Tank [Hopfield85], and of the Wilson and Pawley [Wilson88]. However, in our model, because of the relative good performance of the network for both problems I reduced this value to 200.

7.5 Algorithms for generating linear-tree designs

Based on the implementation details, discussed so far, I now present the algorithm for solving the community norm. For clarity, the variables in this algorithm are printed in italic font.

1- Input n (the number of locations).
2- Set the parameters value:
  - Input the flow between each location-pair.
  - Set *timed-out* value.
  - *u0* := 0.02.
  - *u_init* := - (*u0* * log (*n* - 1.0))/2.0.
3- Initialize each neuron's input voltage:
  u_volt_{xi} := *u_init* + random-noise.
4- $Normalized\_flow_{xi}$:= $input\_flow_{xi}$ / *Maximum_flow*.
5- Add heuristics to the normalized-flow to accelerate convergence
  (Choose two most interactive locations as the end locations).
6- *time_out* := FALSE.
7- *freeze* := FALSE.
8- *Valid_solution* := FALSE.
9- WHILE (!*freeze* && !*Valid_solution* && !*timed_out*):
  9.1- Calculate the output voltage for each neuron.

- $V_{xi} := (1.0 + \tanh(u\_volt_{xi} / u0)) / 2.0$.

9.2- *freeze* := TRUE.

9.3- FOR each $neuron_{xi}$ Do:

$$terms = A\sum_{j\neq i} V_{xj} + B\sum_{y\neq x} V_{yi} + C\left(\sum_{x}\sum_{j} V_{xj} - n\right) - D\left(\sum_{y\neq x}\sum_{j\neq i} f_{xy}(|i-j|-1)\, V_{yj}\right)\left(\sum_{x}\sum_{y\neq x}\sum_{i}\sum_{j\neq i} f_{xy}(|i-j|-1)\, V_{xi}V_{yj}\right)^{-2}$$

- *old_motion* := $motion_{xi}$.
- $motion_{xi} := (- u\_volt_{xi} - terms) / 1E+5$.
- IF: $(|motion_{xi} - old\_motion| < 1E-35)$.
  THEN: *freeze* := FALSE.

9.4- END_FOR.

9.5- Calculate the new input voltage for each neuron.

- *Old_volt* := u_volt_{xi}.
- $u\_volt_{xi} := u\_volt_{xi} + motion_{xi}$.
- IF: $(|u\_volt_{xi} - Old\_volt| < 1E-35)$.
  THEN: *freeze* := FALSE.

9.6- Calculating the energy function.

- *Old_energy* := *Current_energy*.
- $$Current\_energy := \frac{A}{2}\sum_{x}\sum_{i}\sum_{j\neq i} V_{xi}V_{xj} + \frac{B}{2}\sum_{x}\sum_{y\neq x}\sum_{i} V_{xi}V_{yi} + \frac{C}{2}\left(\sum_{x}\sum_{i} V_{xi} - n\right)^2 + \frac{D}{2}\left(\sum_{x}\sum_{y\neq x}\sum_{i}\sum_{j\neq i} f_{xy}(|i-j|-1)\, V_{xi}V_{yj}\right)^{-1}$$
- IF: $(freeze \;\&\; (|Old\_energy - Current\_energy| < 1E-35))$.
  THEN: *freeze* := FALSE.

9.7- Check for convergence of the network to a solution.

- IF: - A single neuron in each row has a voltage greater than a threshold (i.e., $V_{xi} > 0.1$).
  - A single neuron in each column has a voltage greater than the threshold value.
  THEN: - *Valid_solution* := TRUE.
- IF: ((*Valid_solution* = TRUE) OR (*freeze* = TRUE) *OR* (*iteration* > *timed-out*)).
  THEN: Interpret and output the design and its evaluation value.
- IF: *freeze* = TRUE.
  THEN: print "The process was frozen.".
- IF: *Valid_solution* = TRUE.
  THEN: print "The process was timed out".
- IF: (*Valid_solution* = TRUE).
  THEN: print "The process was successful".

9-8- END-WHILE.

10- END

The algorithm for generating AD with respect to the norm privacy / circulation-cost is the same as the above algorithm except the steps that calculate the equation of motion and the energy function. The energy function (13) and the equation of motion (14) for these norms resemble very much the corresponding equations for the TSP. However, one must keep in mind that in the TSP the search is for a closed path, while in the AD case we are interested in Linear-tree solutions (i.e., open paths). For this reason, in the TSP problem the subscripts for calculating the equation of motion must be taken as modulo n, so that the n-th city is adjacent to the first. In the AD case this condition is neglected.

The computational complexity of the model for each iteration is in the order of $O(n^4)$. The worst situation for the model is calculation of the fourth term for the equation of motion and the energy function. This approach has a polynomial time complexity of order 4 as opposed to a combinatorial approach for solving the same problems which has an exponential time complexity.

7.6 Experiments and test results

The Hopfield model for the AD discussed in this paper was implemented in the programing language C and was run on an IBM/PS2 model 80 computer. In this section, after giving a test example, the results of the random tests run for up to 10 locations are summarized in different tables. The resulting designs are not judged whether they are optimal or not, since first, optimal designs are not known except for a small number of locations, and second, satisfactory solutions suffices design problems. Intuitive examination of the results, and a comparison of results of the same problem with respect to conflicting norms shows that they are satisfactory and in the neighborhood of the optimal solutions. Test results are presented in terms of the number of iterations. Greater efficiency could be achieved by parallel implementation of the model.

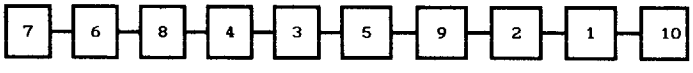
Table 1 shows randomly generated numbers representing flow potentials between 10 locations or activities of some kind in a building. This matrix was used as input to the network. The resulting design patterns are

exhibited in figures 4 and 5. I am discarding the index of group formation as well as the unit utilities for different locations. These factors do not have any effect on the model nor on the performance of the network, since it is always possible to pre-process the flow matrix with regard to these factors before its input to the network.

The evaluation of design patterns gives a total of 67461 community utility for the design with respect to the community norm, and a total of 44274 units cost for the privacy / circulation-cost norm(s). Furthermore in both cases the network converged fast and with a small number of iterations.

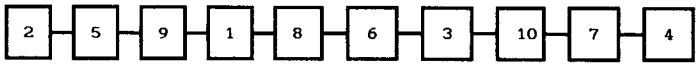
Locations	Locations (activities)									
	1	2	3	4	5	6	7	8	9	10
1	0									
2	539	0								
3	885	655	0							
4	231	2	658	0						
5	578	348	4	82	0					
6	924	694	351	428	160	0				
7	270	41	697	775	506	891	0			
8	617	387	43	121	853	238	277	0		
9	963	733	389	467	199	584	623	316	0	
10	309	312	736	814	545	930	969	662	8	0

Table 7.1: A randomly generated matrix representing the flow between different locations (activities) in a building



Total community utility: 67461.
Number of iterations: 53.

Figure 7.2: A linear-tree design with respect to the community norm



Total privacy cost: 44274.
Number of iterations: 41.

Figure 7.3: A linear-tree design, with respect to the privacy / circulation-cost norm(s)

The following tables summarize the result of runs for randomly generated tests with sizes from 3 to 14 locations. These tables indicate that the

model is efficient and converges fast to a solution. Tables 7.2 and 7.3 show that the convergence efficiencies of the community and privacy modules are more than 83% and 88% respectively. This does not mean that in the rest of the cases the model in incapable of converging to solutions. The threshold value for the timed-out is small in comparison with the Hopfield model which were set to 1000. If the timed-out threshold is increased, then a higher convergence percentage is expected.

Similarly, tables 7.3 and 7.5 show that for both norms more than 80 percent of the cases converged to a solution in less than 100 iterations. This performance is also remarkable. Of course the performance of the model drops considerably when the problem size increases. This deficiency may be overcame by further investigation for finding more appropriate parameters for problems of larger sizes.

# of locations	Total run	# of time converged	# of time Frozen	# of time timed-out
3	50	50	0	0
4	50	50	0	0
5	50	50	0	0
6	50	50	0	0
7	50	50	0	0
8	50	50	0	0
9	50	50	0	0
10	50	40	0	10
11	50	37	0	13
12	50	29	0	21
13	50	23	0	27
14	50	22	1	27
Total	600	501	1	98
%	100	83.5	0.2	16.3

Table 7.2: Convergence efficiency of the community module in terms of the number of solutions found

# of locations	Total run	≤ 100 iterations	> 100 itereration
3	50	50	0
4	50	50	0
5	50	50	0
6	50	49	1
7	50	49	1
8	50	46	4
9	50	43	7
10	50	38	12
11	50	36	14
12	50	38	12
13	50	21	29
14	50	21	29
Total	600	491	109
%	100%	81.8%	18.2%

Table 7.3: Convergence efficiency of the community module in terms of the number of iterations

# of locations	Total run	# of time converged	# of time Frozen	# of time timed-out
3	50	50	0	0
4	50	50	0	0
5	50	50	0	0
6	50	50	0	0
7	50	50	0	0
8	50	50	0	0
9	50	49	0	1
10	50	41	0	9
11	50	43	0	7
12	50	38	0	12
13	50	31	1	18
14	50	29	1	20
Total	600	531	2	67
%	100	88.5	0.3	11.2

Table 7.4: Convergence efficiency of the privacy / circulation-cost module in terms of the number of solutions found

# of locations	Total run	≤ 100 iterations	> 100 iterations
3	50	50	0
4	50	50	0
5	50	50	0
6	50	50	0
7	50	50	0
8	50	50	0
9	50	42	8
10	50	34	16
11	50	35	15
12	50	35	15
13	50	25	25
14	50	26	24
Total	600	497	103
%	100	82.8	17.2

Table 7.5: Convergence efficiency of the privacy / circulation-cost module in terms of the number of iterations

Another attempt was made to investigate the performance of the modules in terms of the quality of the solutions. The results are reflected in the following two tables. These tables which are based on the first 50 solutions found for problems of different sizes show the qualities of test runs in terms of the nearness of the solutions to the best solutions. Test results show that the modules rarely converge to a best solution, but about 80 percents of the solutions found are in the 10 percent range of the best solution. Again, one should note that because of the computational complexity of the problems no attempt was made to verify if the best solutions were optimal solutions, and if not what was their distance from the optimal ones.

% distance from the best solution found / Problem size	% Of solutions falling within specific distance from the best solutions. 0%	5%	10%	15%	20%
3	100	100	100	100	100
4	100	100	100	100	100
5	14	100	100	100	100
6	28	100	100	100	100
7	4	50	76	98	100
8	2	32	60	90	96
9	2	74	100	100	100
10	2	58	72	94	100
11	2	16	66	100	100
12	2	14	34	84	100
13	2	12	60	100	100
	23.5	59.6	79	97	99
	Average solutions falling within specific distances from the best solutoions.				

Table 7.6: Statistics for solutions falling within specific distance from the best solution for the community module

% distance from the best solution found / Problem size	% Of solutions falling within specific distance from the best solutions. 0%	5%	10%	15%	20%
3	100	100	100	100	100
4	82	82	82	82	82
5	18	100	100	100	100
6	24	100	100	100	100
7	10	10	46	72	82
8	4	8	48	70	90
9	2	34	74	74	84
10	2	74	88	97	99
11	2	72	100	100	100
12	2	58	100	100	100
13	2	48	98	100	100
	22.5	62	85	90	94
	Average solutions falling within specific distances from the best solutoions.				

Table 7.7: Statistics for solutions falling within specific distance from the best solution for the privacy module

7.7 Summary

This chapter discussed the neural networks implementation of architectural designs with respect to community and privacy norms. These problems are similar in nature, but mathematically more complex than the travelling salesman problem. Hopfield and Tank chose the TSP problem to introduce their neural network model. Wilson and Pawley [Wilson88] criticized Hopfield and Tank model by saying that their underlying "method is unreliable and does not offer much scope for improvement". My experiments with the adapted version of Hopfield model for AD introduced in this work show that Hopfield model is a promising approach for optimizing architectural designs. The experiments with the network showed that the network generate reasonable and satisfactory design solutions with respect to social norms community and privacy / circulation-cost norm(s).

From a technical point of view, this chapter provided an experimental ground to discover the rôle of variable parameters for the Hopfield model. Both Hopfield and Pawley used constant parameters for different problem sizes. A close examination of this network reveales that constant parameters for problems of different size cannot guarantee the convergence of the network to optimal solutions for all cases.

This fact was more evident in the case of our AD in which value of the fourth term was under the influence of all neurons in the network, and demanded different compensation for different problem size with respect to the D parameter. This terms has a weaker effect in the total behavior of the network for the case of the travelling salesman problem. The solution to this problem is the tuning of parameters of the network to comply with the size of the input problems. Different behaviors of neurons in networks of different sizes demand different-size parameters. Something that was not under immediate effect in the Hopfield implementation of the TSP problem.

Next chapter examines the performance behavior of TOPGENE as a heuristic program by comparing sets of designs generated for the same sets of data by both TOPGENE and the neural network.

CHAPTER 8
A TESTING OF TOPGENE AND FUTURE WORKS

This chapter, presents:

- A comparison of design solutions by TOPGENE and the neural network (NN) for the same problems.
- Limitation of both approaches and future works.

8.1 A comparative look at TOPGENE's and NN's solutions

This section compares solutions generated by the TOPGENE and the NN for the same set of problems. This comparison is based on the limited capability of the NN implementation of our architectural design problem (ADP). The present version of the NN is only generating linear-tree sub-optimal designs with respect to an isolated norm community or privacy. The linear-tree type design happens to have a high degree of complexity, because of the high degrees of interactions that is foreseen between location-pairs of such a design type.

The main ideas behind comparing output of these two systems are, first, a partial evaluation of TOPGENE as a heuristic system in generating designs, and second, discovering some of the shortcomings associated with both approaches.

The NN model accepts the actual or expected flows between location pairs of a design as input, and produces a linear-tree type design in terms of its connectivity pattern. A more sophisticated version of the NN, necessarily should include the Q-analysis module for derivation of expected flow between locations of a design, if a wider range of input data-sets, or if a more row data-set, such as activities and actors responsible for the

activities, should be accepted by the NN.

The input flow tables for the NN, exhibited in the next sections, are randomly generated. Each table is proceeded by the following information for the community norm, and is repeated for the privacy / circulation-cost norm:

- The norm used for generating a design,
- Number of test runs on data,
- Average number of number of iterations for test runs,
- Percent of the NN solutions that converged to the best performing solution,
- The best performing design generated by the NN,
- Result of the performance analysis of the NN's solution,
- TOPGENE's solution for the same problem, and finally,
- Result of the performance analysis of TOPGENE's solutions.

The test runs are carried out for design problems from 4 locations up to 13 locations. Here is the first test run and its results:

The input flows between locations:

	1	2	3	4
1	0	355	701	48
2	355	0	394	740
3	701	394	0	87
4	48	740	87	0

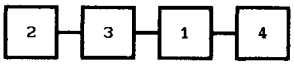
Norm: community.

Number of runs: 50.

Average number of iterations: 30.

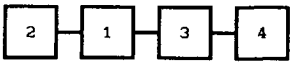
Percent of runs that converged to the best solution: 100%

Best solution found by the NN:



Total community utility: 1922.

Solution generated by the TOPGENE:



Total community utility: 1922.

Norm: privacy / circulation-cost.

Number of runs: 50.

Average number of iterations: 15.

Percent of runs that converged to the best solution: 84%

Best solution found by the NN:

Total privacy cost: 538.

Solution generated by the TOPGENE:

Total privacy cost: 538.

The input flow between locations:

	1	2	3	4	5
1	0	443	453	464	474
2	443	0	485	495	506
3	453	485	0	517	527
4	464	495	517	0	538
5	474	506	527	538	0

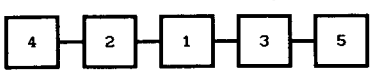
Norm: community.

Number of runs: 50.

Average number of iterations: 35.

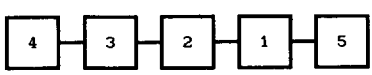
Percent of runs that converged to the best solution: 14%

Best solution found by the NN:



Total community utility: 5083.

Solution generated by the TOPGENE:



Total community utility: 5050.

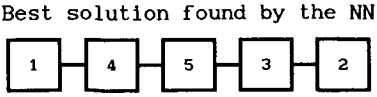
Norm: privacy / circulation-cost.

Number of runs: 50.

Average number of iterations: 40.

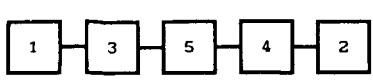
Percent of runs that converged to the best solution: 18%

Best solution found by the NN:



Total privacy cost:4722.

Solution generated by the TOPGENE:



Total privacy cost: 4724.

The input flow between locations:

	1	2	3	4	5	6
1	0	227	248	269	290	311
2	227	0	332	354	375	396
3	248	332	0	417	438	459
4	269	354	417	0	480	502
5	290	375	438	480	0	523
6	311	396	459	502	523	0

Norm: community.

Number of runs: 50.

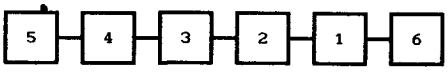
Average number of iterations: 35.

Percent of runs that converged to the best solution: 28%

Best solution found by the NN:

Total community utility: 8405.

Solution generated by the TOPGENE:



Total community utility: 8310.

Norm: privacy / circulation-cost.

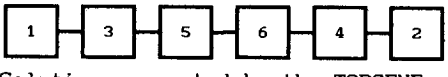
Number of runs: 50.

Average number of iterations: 53.

Percent of runs that converged to the best solution: 24%

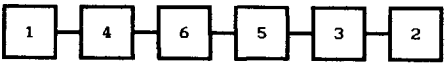
Total privacy cost: 6542.

Best solution found by the NN:



Total privacy cost: 6542.

Solution generated by the TOPGENE:



Total privacy cost: 6545.

The input flow between locations:

	1	2	3	4	5	6	7
1	0	363	709	56	402	748	95
2	363	0	441	787	133	480	826
3	709	441	0	172	519	865	211
4	56	787	172	0	558	904	250
5	402	133	519	558	0	596	943
6	748	480	865	904	596	0	289
7	95	826	211	250	943	289	0

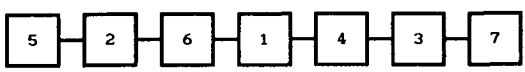
Norm: community.

Number of runs: 50.

Average number of iterations: 41.

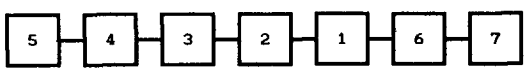
Percent of runs that converged to the best solution: 4%

Best solution found by the NN:



Total community utility: 21079.

Solution generated by the TOPGENE:



Total community utility: 19000.

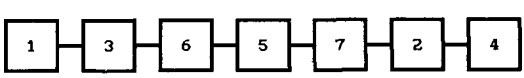
Norm: privacy / circulation-cost.

Number of runs: 50.

Average number of iterations: 59.

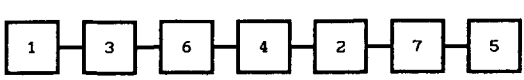
Percent of runs that converged to the best solution: 10%

Best solution found by the NN:



Total privacy cost: 11981.

Solution generated by the TOPGENE:



Total privacy-cost: 12447.

The input flow between locations:

	1	2	3	4	5	6	7	8
1	0	328	674	21	367	713	59	406
2	328	0	752	98	445	791	137	484
3	674	752	0	830	176	523	869	215
4	21	98	830	0	561	908	254	600
5	367	445	176	561	0	947	293	639
6	713	791	523	908	947	0	986	332
7	59	137	869	254	293	986	0	678
8	406	484	215	600	639	332	678	0

Norm: community.

Number of runs: 50.

Average number of iterations: 50.

Percent of runs that converged to the best solution: 2%

Best solution found by the NN:

Total community utility: 37732.

Solution generated by the TOPGENE:

Total community utility: 36349.

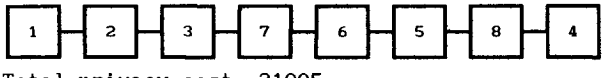
Norm: privacy / circulation-cost.

Number of runs: 50.

Average number of iterations: 59.

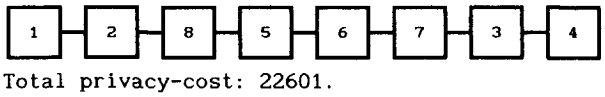
Percent of runs that converged to the best solution: 4%

Best solution found by the NN:



Total privacy cost: 21995.

Solution generated by the TOPGENE:



Total privacy-cost: 22601.

The input flow between locations:

	1	2	3	4	5	6	7	8	9
1	0	995	5	16	26	37	47	58	69
2	995	0	79	90	100	111	121	132	142
3	5	79	0	153	164	174	185	195	206
4	16	90	153	0	216	227	238	248	259
5	26	100	164	216	0	269	280	290	301
6	37	111	174	227	269	0	312	322	333
7	47	121	185	238	280	312	0	343	354
8	58	132	195	248	290	322	343	0	364
9	69	142	206	259	301	333	354	364	0

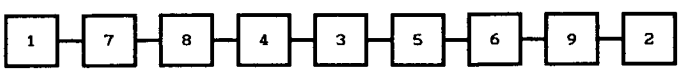
Norm: community.

Number of runs: 50.

Average number of iterations: 60

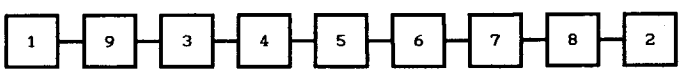
Percent of runs that converged to the best solution: 2%

Best solution found by the NN:



Total community utility: 20064.

Solution generated by the TOPGENE:



Total community utility: 19633.

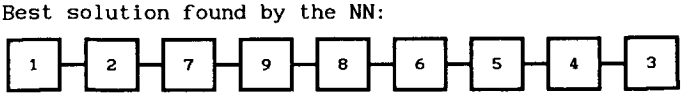
Norm: privacy / circulation-cost.

Number of runs: 50.

Average number of iterations: 62.

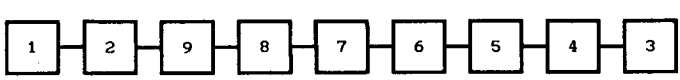
Percent of runs that converged to the best solution: 2%

Best solution found by the NN:



Total privacy cost: 11161.

Solution generated by the TOPGENE:



Total privacy-cost: 11231

The input flow between locations:

	1	2	3	4	5	6	7	8	9	10
1	0	826	836	847	858	868	879	889	900	910
2	826	0	921	932	942	953	963	974	984	995
3	836	921	0	6	16	27	37	48	58	69
4	847	932	6	0	80	90	101	111	122	132
5	858	942	16	80	0	143	153	164	175	185
6	868	953	27	90	143	0	196	206	217	227
7	879	963	37	101	153	196	0	238	249	259
8	889	974	48	111	164	206	238	0	270	280
9	900	984	58	122	175	217	249	270	0	291
10	910	995	69	132	185	227	259	280	291	0

Norm: community.

Number of runs: 50.

Average number of iterations: 67.

Percent of runs that converged to the best solution: 2%

Best solution found by the NN:

Total community utility: 69204.

Solution generated by the TOPGENE:

Total community utility: 66792

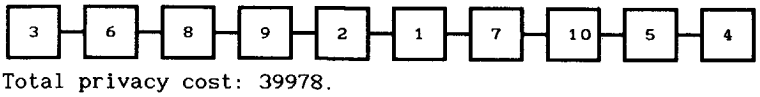
Norm: privacy / circulation-cost.

Number of runs: 50.

Average number of iterations: 62.

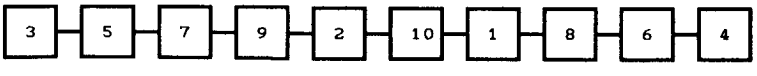
Percent of runs that converged to the best solution: 2%

Best solution found by the NN:



Total privacy cost: 39978.

Solution generated by the TOPGENE:



The input flow between locations:

	1	2	3	4	5	6	7	8	9	10	11
1	0	197	229	260	292	324	356	387	419	451	482
2	197	0	514	546	578	609	641	673	704	736	768
3	229	514	0	799	831	863	895	926	958	990	21
4	260	546	799	0	53	85	117	148	180	212	243
5	292	578	831	53	0	275	307	338	370	402	434
6	324	609	863	85	275	0	465	497	529	560	592
7	356	641	895	117	307	465	0	624	656	687	719
8	387	673	926	148	338	497	624	0	751	782	814
9	419	704	958	180	370	529	656	751	0	846	877
10	451	736	990	212	402	560	687	782	846	0	909
11	482	768	21	243	434	592	719	814	877	909	0

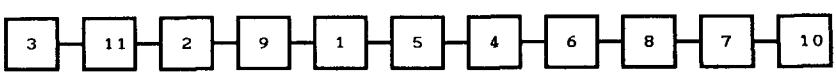
Norm: community.

Number of runs: 50.

Average number of iterations: 67.

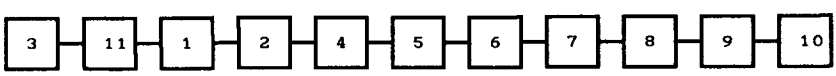
Percent of runs that converged to the best solution: 2%

The design solution:



Total community utility: 106510.

Solution generated by the TOPGENE:



Total community utility: 104007

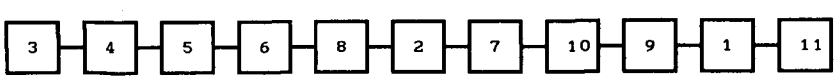
Norm: privacy / circulation-cost.

Number of runs: 50.

Average number of iterations: 74.

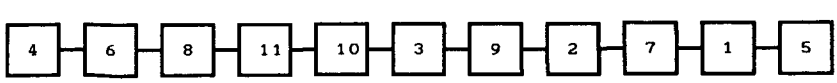
Percent of runs that converged to the best solution: 2%

The design solution:



Total privacy cost:78701.

Solution generated by the TOPGENE:



Total privacy-cost: 74846

The input flow between locations:

	1	2	3	4	5	6	7	8	9	10	11	12
1	0	438	469	501	533	564	596	628	659	691	723	755
2	438	0	786	818	850	881	913	945	977	8	40	72
3	469	786	0	103	135	167	198	230	262	294	325	357
4	501	818	103	0	389	420	452	484	516	547	579	611
5	533	850	135	389	0	642	674	706	737	769	801	833
6	564	881	167	420	642	0	864	896	928	959	991	23
7	596	913	198	452	674	864	0	54	86	118	150	181
8	628	945	230	484	706	896	54	0	213	245	276	308
9	659	977	262	516	737	928	86	213	0	340	372	403
10	691	8	294	547	769	959	118	245	340	0	435	467
11	723	40	325	579	801	991	150	276	372	435	0	498
12	755	72	357	611	833	23	181	308	403	467	498	0

Norm: community.

Number of runs: 50.

Average number of iterations: 75.

Percent of runs that converged to the best solution: 2%

The design solution:

Total community utility: 131156.

Solution generated by the TOPGENE:

Total community utility: 127825

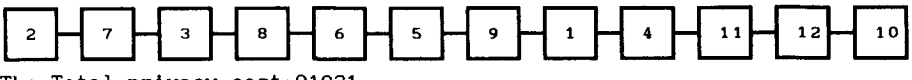
Norm: privacy / circulation-cost.

Number of runs: 50.

Average number of iterations: 81.

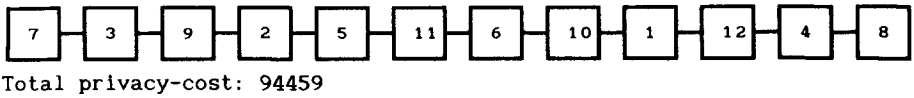
Percent of runs that converged to the best solution: 2%

The design solution:



The Total privacy cost:91031.

Solution generated by the TOPGENE:



Total privacy-cost: 94459

The input flow between locations:

	1	2	3	4	5	6	7	8	9	10	11	12	13
1	0	308	340	372	404	435	467	499	530	562	594	626	657
2	308	0	689	721	752	784	816	847	879	911	943	974	6
3	340	689	0	38	69	101	133	164	196	228	260	291	323
4	372	721	38	0	355	386	418	450	482	513	545	577	608
5	404	752	69	355	0	640	672	704	735	767	799	830	862
6	435	784	101	386	640	0	894	925	957	989	21	52	84
7	467	816	133	418	672	894	0	116	147	179	211	242	274
8	499	847	164	450	704	925	116	0	306	338	369	401	433
9	530	879	196	482	735	957	147	306	0	464	496	528	560
10	562	911	228	513	767	989	179	338	464	0	591	623	655
11	594	943	260	545	799	21	211	369	496	591	0	686	718
12	626	974	291	577	830	52	242	401	528	623	686	0	750
13	657	6	323	608	862	84	274	433	560	655	718	750	0

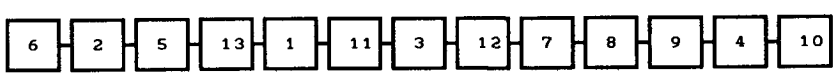
Norm: community.

Number of runs: 50.

Average number of iterations: 68.

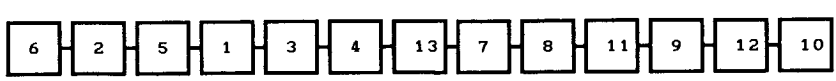
Percent of runs that converged to the best solution: 2%

The design solution:



Total community utility: 167695.

Solution generated by the TOPGENE:



Total community utility: 167023

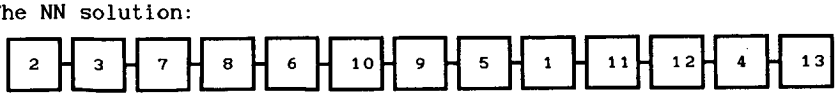
Norm: privacy / circulation-cost.

Number of runs: 50.

Average number of iterations: 75.

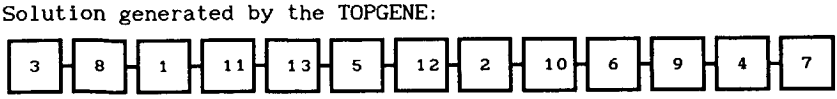
Percent of runs that converged to the best solution: 2%

The NN solution:



Total privacy cost:127322.

Solution generated by the TOPGENE:



Total privacy-cost: 131759

The following tables compare performances of designs generated by both systems:

Problem size	Performance value (utility) of the NN best solution	% runs converged to best	Performance value of the TOPGENE's ...	relative distance in % of TOPGENE solutions from the NN solutions
4	1922	100	1922	0
5	5083	14	5050	0.6
6	8405	28	8310	1.1
7	21079	4	19000	9.8
8	37732	2	36349	3.6
9	20064	2	19633	2.1
10	69204	2	66792	3.4
11	106510	2	104007	2.3
12	131156	2	127825	2.5
13	167695	2	167023	0.4
			Ave.	2.58

Table-1: Comparison of the performance of TOPGENE and the NN with respect to the community norm

Problem size	Performance value (cost) of the NN best solution	% runs converged to best	Performance value of the TOPGENE's ...	relative distance in % of TOPGENE solutions from the NN solution
4	538	84	538	0.0
5	4722	18	4724	0.0
6	6542	24	6545	0.0
7	11981	10	12447	3.9
8	21995	4	22601	2.8
9	11161	2	11231	0.6
10	39978	2	40380	1.0
11	78701	2	74846	-4.9
12	91031	2	94459	3.8
13	127322	2	131759	3.5
				Ave.1.07

Table-2: Comparison of the performance of TOPGENE and the NN with respect to the privacy norm

Above test-runs show that:

- Except for small sized problems, the NN solutions are better performing than TOPGENE's solutions.
- Only in one case (i.e., table-1, problem size 7) the solution generated by TOPGENE is about 10% worse than a solution generated by the NN for the same problem.
- On the average, the performance behavior of designs generated by TOPGENE are not more than 2.5 percent worse than the solutions generated by the NN for the same set of data. Considering the speed of TOPGENE in generating designs relative to the NN, this deviation is very low and negligible.
- The problem size 11 in table-2 shows that it is even possible for the TOPGENE to perform better than the NN in terms of the quality of a solution to a design problem.
- The NN implementation of the ADP problems, in general converges faster than the original Hopfield and Tanks NN model for the travelling salesman problem (TSP) in terms of the number of iterations that it takes to produce a solution. The NN converges to a solution on the average with 50 iterations. This is a very low number of iterations.
- Because of the computational complexity of the ADP, the time that one iteration takes is time-wise more costly than the TSP problem. For example, for a design with 10 to 15 locations the time that it takes for the NN to converge to a solution is between one to two hours on an IBM-PS2 model 80 machine. This in comparison with TOPGENE performance also is a long time. TOPGENE produces a solution for the same problems in an interval between one to two minutes depending on the type of solution required. The NN implementation of the ADP problem in software is much slower than the TOPGENE approach.

Based on above facts, and considering that, first, TOPGENE is a heuristic program that relies on rules of thumbs for generating any designs, and second, the NN is a computational system, one can conclude that TOPGENE is a remarkably fast performing system. In addition, one must remember that because of the possibility of integrating TOPGENE with knowledge bases of recommended and prohibited accesses in a building, TOPGENE non-linear designs are more realistic than a neural network designs. The most that a

neural network similar to one discusses in last chapter can do is to strive for an optimized design regardless of whether it is practical on not. Something common between the NN and an analytical optimization method for generating a design.

8.2 Limitations of TOPGENE and future works

There are limitations with TOPGENE, as a functioning system, some of which can be eliminated by its expansion. There are many possibilities for increasing the effectiveness and efficiency of TOPGENE as a spatial reasoner or as a topological pattern generator. One of the most important measures of the success of TOPGENE is the ease with which such extensions can be made. There are also differences between the theory behind TOPGENE, as discussed in [Tzonis87], and the implementation. In this section I discuss these differences, and propose possible extensions to the system.

Path finding and division of flow on them

Path finding is one of the bottlenecks in TOPGENE. As, it was discussed in the implementation chapter, TOPGENE needs to detect paths between locations for two purposes. First, for keeping track of the shortest paths between locations in partial designs while configuring a design, and second, once a design is configured, TOPGENE has to analyze and calculate performance behavior of the design with respect to the social norms, and possibly produce a diagnosis report for its behavior. The computational complexity of the path finding was eased by devising an special algorithm for keeping track of distances in growing partial designs, as was discussed in chapter 4. The complexity of best known algorithms for generating paths between all location pairs in a graph is n^3 [Buckley90] [Gibbons85]. This complexity, although for structurally complicated designs, may be time consuming but is bearable if it has to be calculated once for a design, and not for every partial design as in TOPGENE. People, however take the shortest path towards their destination if they know it and if it is a permitted path. This rule also is applicable in a building as well.

TOPGENE may apply this human heuristic dividing flows on paths based on statistical or probabilistic assumptions. Because of modularity of the

TOPGENE this is not a hard task to fulfill.

Increasing the system efficiency

A major thrust of this work was to find an efficient method for solving a class of design problems which are computationally intractable. The goal has played a major part in deciding on the search strategy that enable the system to perform well in terms of the computational efficiency. TOPGENE is written in GCLISP and runs on an IBM PS2 machine. From the early stages of the system implementation, several key decision was made and methods adopted to increase the efficiency of the system. This included devising a new graph manipulation algorithm, discussed in chapter 4, and taking advantage of existing information within the system to speed up the system. Example of the second case was the use of distance matrix for speeding up findings of the shortest paths in a partial design. This decision speeds up the system, time-wise, by a factor of 10 to 15.

The slow performance in matrix manipulation of most of the LISP interpreters, including GCLISP, is bottle neck in TOPGENE, specially for problems of large size. For example, TOPGENE may take several minutes to pre-process data (i.e., Q-analysis) for a design problem of 20 to 30 locations. This problem is believed to ease if TOPGENE is interfaced with other programming languages such as C, for handling of sub-processes such as Q-analysis, that heavily rely on numerical calculation.

Integration of a data-base of precedents of architectural designs

A data-base of architectural designs including their connectivity patterns is a natural extension to the TOPGENE. Such an extension to TOPGENE necessarily demands study on the choices of numerical indices, the same as discussed in chapter 4, for indexing and retrieval of the precedents. I strongly suggest a sub-set of the topological indices (TIs) discussed in appendix-A for this purpose. The hierarchical model for measuring complexity of graphs proposed by Bonchev [Bonchev87a], discussed in section 4.1.4 is one of the best choices. This model can capture most of the structural (topological) properties of a building in a vector type index that can be used for indexing and retrieval of precedents of architectural design. One should note that the integration of precedents of designs necessarily would not undermine the current status of TOPGENE in terms of its algorithms and

methods that it uses in generating new designs. The reasons are, first, there is always a need for analytical treatment of data on design problems dealing with social norms with a method such as Q-analysis; second, it is always possible that a precedent solution for a design problem does not exist, and that it necessarily should be generated directly, based of the data presented to the system; and third, a data base of precedents of design may only help in the design generation activities, and not the evaluation. The evaluation problem, thus, needs the current approach by TOPGENE in any cases.

Introduction of non-uniform utility / costs to the system

TOPGENE, currently assumes uniform utilities / costs associated with behavior of different parts of a building. This, although may be true in some situations, but in general, a designer may consider different utilities and costs associated with performances of different parts of a design with respect to the social norms. Addition of this capability is a possibility that also is in conformity with the current architecture of TOPGENE. Only, different indices have to be used for calculating the performance behavior of different locations.

Focusing social points of view on micro level rather than for the totality of a design

The current version of the TOPGENE, assumes uniformly similar social requirements for a design in its totality, and consequently for each location of a design. It is quite natural that a designer demands different performance from different parts of a building with respect to the social norms. Here, as in the previous case, a more complicated analysis of design data is needed. TOPGENE based on this approach would have to process data related to parts of a design with different behaviors separately, proceeds design for each part, and combine the sub-designs found, to form a total design. Such an approach is very much akin to the model of design based on decomposition of the problem, discussed in chapter 2.

Another addition is enabling the system to respond to a different type of problem formulation based on unlike social requirements from various parts of a building. This is different from the case discussed above. In previous case, the requirements of a design were the same for various parts,

while the indices of social behaviors were different. Here, unlike performance is demanded for different parts of a design regardless of the input data and indices of social behaviors.

Introduction of the time constraint to the system

Flow-rates between different locations of a building are in reality a time dependent phenomenon. A flow rate between two specific locations changes over time. For example, the flow rate in one part of a building may be higher during the day than night, and another part of the building may behave on the contrary. TOPGENE currently assumes constant flow-rate between different locations in any times, or as one may interpret, it considers the average flow rate, or total flow-rate for a time-interval.

Involvement of this timing factor is another extension possibility for the system. Such an extension also requires only changes to the data preparation and data analysis part of the system and not in its general approach.

Adding friendlier input user interface

A friendly interface plays an important rule in acceptance of a system by its users. TOPGENE currently supports a graphical interface for input and automatic display of connectivity patterns of buildings. The system also includes window facilities for inputting other design data for generating topological patterns. However, because of the reliance of the system on a fairly vast amount of information for process, thinking about a friendlier user interface worth the effort.

Incorporation of explanation facilities

TOPGENE, on a limited base, supports and navigate the user on entering data to the system, and almost has no facilities for explaining its course of actions at micro levels. The system, is perfectly capable of providing logical explanations for the actions that it takes based on the mathematical information that it collects about a building. Adding (optional) explanation facilities to the system is a user's favorite.

Processing of location pairs with the same priority

TOPGENE cannot give processing priority to location pairs within the same cluster (i.e., priority in terms of flow generation potential). This is not

always healthy in terms of the outcome of a design. For example, suppose that we have the following location pairs in a cluster to be processed:
(toilet conference-room)
(chief-room conference-room)
(chief-room secretary-room)

Handling the first location pair, for example with respect to the privacy norm, After will create new connection (access) between the conference-room and the toilet, and the branchiness degree of the conference-room is increased by 1. In the next iteration, the location pair (chief-room and conference-room) must be processed. Now, if the limit for branching degree of the conference-room is reached, then TOPGENE will avoid creating new access between the conference-room and the chief-room. It is obvious that if a designer had to make an access choice between the toilet and the conference-room to the chief-room, then he would have chosen the conference-room. Two possibilities exist for overcoming this problem:

- Letting the system interact with the user while confronting such a situation.
- Enriching the system with a larger knowledge base, and stronger knowledge of architectural design.

A solution based on the second proposal is discussed in the next section.

Enhancing the knowledge-bases with class descriptions and flow weights

The knowledge bases of recommended and prohibited accesses in buildings proved to play an important rôle in generating more realistic and practical designs. These knowledge bases, integrated into the system in final design stages of TOPGENE, currently consists of simple assertions concerning the recommended and prohibited accesses in buildings. The efficiency and productivity of these knowledge bases may be further improved by taking advantage of AI representation techniques such as class description and including a broader knowledge of architectural design at topological level. Class description of recommended and prohibited links helps in both broadening and deepening the reasoning process of TOPGENE in a sense that ,first , it provides a means for deeper classification of access knowledge of design, and second, classification in turn provides the system with broader reason-

ing possibilities.

Another improvement in this direction is the enhancement of the recommended and prohibited accesses with the weights (i.e., potential flows between the location pairs) associated with them. These weights may play rôle in generating designs with different properties in terms of optimality or realism. As an example, consider the following set of recommended accesses in a building.

```
(access hall living-room)
(access hall bed-room)
(access hall kitchen)
(access kitchen-1 kitchen-2)
(access living-room kitchen-1)
(access living-room kitchen-2)
```

These recommended accesses are available to the TOPGENE in order of their priorities with respect to the flow generation potential between the location pair in each access. However, the actual flow between the location pairs is not evident to TOPGENE. The Q-analysis process presently simply ranks recommended accesses extracted from the knowledge base before passing them to an appropriate task-executer to be processed. This process becomes more attractive if weights are kept in each assertion reflecting a recommended or prohibited link. In such case, a wiser mechanism for dealing with the knowledge of access in a building is possible. For example, weights allow a better switching control between different processes that work on the relations between different locations of a building in terms of flow degrees (weights) between them.

Let's consider above recommended accesses enhanced with flow weights as follows:

```
(access hall living-room 214)
(access hall bed-room 210)
(access hall kitchen 108)
(access kitchen-1 kitchen-2 55)
(access living-room kitchen-1 55)
(access living-room kitchen-2 40)
```

Now, if we suppose that a location pair with maximum priority in the list of

prohibited accesses for the same building has a flow degree of 100. A TOPGENE task-executer processing above recommended accesses is preferred to stop, when (hall kitchen) is processed and switch to processing non-recommended links having a hire priority. The current version of TOPGENE may not make such decisions because of the absence of the flow weights in the partially hierarchical clusters of location pairs. One should note that the implementation of this idea in fact requires a trade-off mechanism between the optimality and realism of a design.

Error detection and error correction

Detecting and correcting erroneous data is necessary part of any system requiring outside data for consumption. For example, if the input data contradict system requirements, it should be detected or the user must be warned against the consequences. In general, error correction depends on detecting where errors are introduced into the system. To make the system robust, such an error correction is integrated into the TOPGENE. But, since TOPGENE is adaptable to both being a deductive knowledge-based system or an interactive system, thought should always be given on the detection and correction of mistakes on data entry points.

8.3 Limitations of the NN approach and future works

The neural modeling step towards the architectural problem was taken at the final stages of TOPGENE implementation. This approach was not intended to be finalized and complete. There are many unresolved situations in this respect. Here are notes on the shortcomings associated with the NN:

Generating all design types

The current NN implementation of the architectural design problem is only capable of generating linear-tree designs with respect to a single norms community, privacy, or circulation-cost. The possibilities of generating other types of designs are open to quest. I should mention that, on the contrary to what may seems, other types of design such as a general tree or near-optimal types do not have a high priority for this approach. The linear-tree type was the most important of all with this respect. The reason is that, first, linear-tree type designs are the optimal type design for the

community norm; second, computationally this type of design is one of the most intractable types with respect to all norms. In fact, detection of a near-optimal design in the form of a general graph or a tree graph for the privacy and the circulation-cost norm turns into the problem of detecting a minimum flow spanning graph (or tree) that is also planar. This was already discusses in chapter 5. Implementation of such an algorithm in a NN does not seem as important as the linear-tree type designs. One reason is existence of algorithms for solving such problems, which do not exist for the previous case.

Design with respect to the intervening opportunity norm

Another shortcoming of the NN is generating designs with respect to the intervening opportunity norm. The energy functions devised for other norms are valuable bases for discovering a new energy function for optimizing designs with respect to this norm. Interesting enough is the fact that an optimal solution with respect to this norm also is in the form of a linear-tree type design. Linear-tree type design has potential for producing maximum in-betweenness condition which is valuable to the intervening opportunity norm.

Design with respect to a combination of norms

Generating designs with respect to a combination of social norms is another deficiency in the NN. I believe, once the strategies for generating sub-optimal designs with respect to single norms are complete, generating designs with respect to a combination of them are not a hard task to accomplish. The most important aspect of the neural network used in this work, as described in chapter 7, are the control strategy of a network for accomplishing a task. The main contributor to the control mechanism of Hopfield model is the equation of motion. A natural suggestion for a control strategy for a similar network capable of solving a design problem with respect to a combination of norms is a control mechanism that combines the control mechanism discovered for each norm separately. This strategy, already experienced my myself works well and produces result. This is the same as letting all equation of motions related to norms govern the network simultaneously, and compete with each other to derive the network towards their own goal. The ultimate wining neurons will be those having or gaining upper hand with

respect to a norm during the process. Optional weights also may be attached to each equation of motion as a secondary control in elevating the effects of some norms in the outcome of a design with respect to the other norms.

I used a combination of equations of motion for the community and privacy / circulation-cost norms to create a new network generating linear-tree design. The initial guess was that such a network, due to conflict between these norms, will not settle down to a solution and endlessly will continue to oscillate; but, the results were on the contrary. The network converges to solutions with behaviors that are indeed somewhere in-between the behavior of two designs generated with respect to each norm separately. Continuation of this approach and further enhancement with the intervening opportunity norm is a worthwhile attempt.

Here, is an example of a design generated by the NN with respect to the combination of norms privacy and community. The generated designs with respect to each norm separately also are presented for comparison of their behavioral values with behavioral values of the design with respect to combination of the norms.

The input flow between locations:

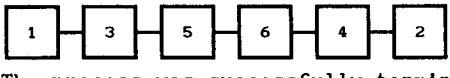
0	468	500	532	564	595
468	0	627	659	690	722
500	627	0	754	785	817
532	659	754	0	849	881
564	690	785	849	0	912
595	722	817	881	912	0

Design with respect to the community norm:

The process was successfully terminated after 30 iterations.

Total community utility: 15169. units.

Design with respect to the privacy norm:



The process was successfully terminated after 45 iterations.

Total privacy cost: 12379. units.

Design with respect to combination of the norms community and privacy / circulation-cost:

The values of parameters for this test run was:

A = 500.00, B = 500.00, C = 150.00, D_c = 1.000.000n, and D_p = 100 / n, where D_c and D_p are D parameters for the community and privacy / circulation-cost energy functions, respectively.

The process was successfully terminated after 51 iterations.

Total community utility: 14791.00 units.

Total privacy cost: 14791.00 units.

This example, and several other test runs, show that combining the energy functions in the NN is a possibility for generating design with respect to multiple norms. The community utility and privacy cost for the third solution are both 14791 units. This value is somewhere in-between the 15169 units utility and 12379 units cost for designs with respect to single norms community and privacy, respectively. These values show that the third solution, having more tendency towards the design with respect to the community norm, but is indeed a design with a behavioral value obtained by a trade off between the behavioral value of the two designs with respect to a single norm.

If such a network, in a long run, shows bias towards a norm, then this problem may be eliminated by tuning the network by changing the parameters values, or as was discussed earlier, by adding weigh options to corresponding equation of motions.

CHAPTER 9
CONCLUDING REMARKS

This chapter discusses the characteristics of TOPGENE, the contributions of the work carried out, and presents some final remarks regarding TOPGENE's potential. In the final section I suggest two other possibilities for solving the design problem discussed in the work.

9.1 The characterization of TOPGENE

AI has its own set of standards for judging theories. An AI theory is *broad* if it describes a broad class of knowledge and inference. It is *coherent* if its sub-parts fit together well. It is *resonant* if it introduces ideas that are relevant to other AI programs. It is *robust* if it can be extended to handle more knowledge and more inference, and if it will interface cleanly with other AI theories. TOPGENE stands relatively high with respect to the AI standards.

In any integrated computer system, many or most of the design decisions, including the program specifications, are made largely to fit in with other design decisions. In learning from such systems, it is important to decide what features are valid outside the system, and what is their scope. At most specific level, TOPGENE builds abstract graphs and uses Q-analysis method to infer the flow potentials between the locations in a design and cluster them in a partially hierarchical manner. TOPGENE is also an AI system that reasons about the structural-functional interrelationship. Several aspects of TOPGENE are relevant to spatial reasoning in general. For example there is not much difference in optimizing urban plans with respect to social norms than optimizing a building with this respect. An urban

planning problem with respect to the same social norms discussed in this work can directly be presented to TOPGENE for a solution.

TOPGENE is also characterized as a deep reasoning system. It produces inference based on mathematical modeling and a relatively deep knowledge of architectural design compared to shallow knowledge in the form of inference rules.

TOPGENE, apparently a complicated system based on methods brought together from many disciplines, is a coherent system. All its algorithms and data structures interface with each other in a clear way. Each module of TOPGENE uses neatly one or more aspects of connectivity patterns of a building that are in principle relevant.

Emphasizing coherence usually limits the scope of the problem that is to be studied, and the system to be developed. Coherence limits expansions of a system unless full implications for representation and inferences on expanded parts are understood. TOPGENE, however, has a high degree of flexibility and robustness in terms of the extensions. This shown in the proposals of chapter 8. TOPGENE may be extended beyond its present capabilities without needs for a great deal of changes in its data structures and reasoning processes.

It is more difficult to assess the significance of TOPGENE to the knowledge based systems. TOPGENE presently takes advantage of a simple knowledge base of recommendations and prohibitions of access in buildings, and uses them in conjunction with its deep reasoning process. A more sophisticated version of TOPGENE should use a wider knowledge-base of design, and an inference method that has more significance to the knowledge based systems technology. However, TOPGENE has proved that domain knowledge plays a significant rôle in generating realistic architectural designs.

TOPGENE,s competence is generalizable in the following sense:

- The kernel of the system consists of connectivity patterns of a building with activities and actors attached to its locations. Matrices and other suitable graph representation techniques are chosen to represent and manipulate the kernel information for deriving implicit information vital for reasoning. Such information includes the underlying relationships between the activities and

actors within a building, which in turn cause the circulation flow of people and objects in the building, and consequently results in interactions between the actors. The interactions between the actors results in the social performance of a building. Other implicit information includes paths between locations, information for hierarchical abstractions etc.

- Hierarchical abstraction is used, for abstracting the relationship information from fine-grain level to a higher and more abstract level. This task is accomplished by the Q-analysis method.
- TOPGENE contains micro-level knowledge of architectural design, and also general heuristic rules applicable to other domains with the same underlaying properties. The model can be used to analyze the behavior of other systems similar to buildings.

9.2 Contributions

The main focus of this work has been discovering ways for tackling a design problem characterized as intractable. The work has fulfilled this objective by addressing several issues surrounding the goal, and has made several contributions on the course of its quest for unknowns:

The problems tackled in this work raise a number of issues of interest to the AI and architectural design communities. The principal issues addressed in this study can be summarized as follows:

- The development and testing of TOPGENE was definitely worthwhile. Without TOPGENE the theory of architectural design at topological level presented would have been considerably more nebulous and less coherent. The need for much of the details of representation, algorithms, techniques used, and many of the normalcy conditions could not have been seen a priori; but once seen they could be justified on theoretical grounds.
- TOPGENE is an example of how ideas from different disciplines may be brought together in a design automation process.
- The development of TOPGENE, the neural network implementation of ADP, and ideas discussed in the next section, all lay down examples of an AI approach in dealing with intractable problems, such as a design

problem, which is generally dealt with by intuition and is considered hard to automate. TOPGENE is an example of heuristic programming in dealing with intractability, while the neural network is a proof of the usefulness of neural modeling techniques in design.

- TOPGENE showed, once more, the rôle of domain heuristics in reducing the complexity by integrating them into the state-space of search.
- TOPGENE has shown the rôle of existing graph theoretical works and algorithms in automating architectural designs at topological level.
- The work has brought into attention the significance of topological indices (TIs) in design and development of precedents-based automated systems of architectural design. TIs were also used in TOPGENE to diagnose eccentricity of a location with respect to other locations in a design.
- Abstraction levels help in problem solving. TOPGENE provides abstraction decomposition of the design problem, to achieve design generation, and to evaluate existing design problems. TOPGENE has shown that abstraction and hierarchies plays an important rôle in reducing search efforts. Abstraction of data in TOPGENE is carried out by the Q-analysis method. The work proves that this method has an important rôle in abstraction, externalization of hidden design information, measuring dynamic aspects of buildings such as flow potential between location pairs, and in hierarchical clustering of objects with respect to a common attribute of them.
- Knowledge plays an important rôle in architectural design. The use of heuristics together with mathematical modeling of building behavior do not suffice for generation of realistic designs. Without design knowledge it is hard to generate realistic designs, and without mathematical modeling it is hard to judge the quality of generated designs or existing designs. These are other lessons learned from the development of TOPGENE.
- TOPGENE achieves some generality by applying mathematical notions in a relatively general way. The proposed methodology has general applicability in similar design problems such as in urban design.
- Development of a new algorithm for detecting new tracks and keeping track of distances in dynamically-growing graphs.

- The rôle of domain constraints (e.g., planarity of topological patterns), in reducing the states in the search space.
- Viewing architectural norms as constraints, and relaxing them when conflicts occur.

9.3 Other possibilities

During the work, several possibilities for tackling the architectural design problem at topological level was considered worthwhile, two of which are briefly described in the following sub-sections.

9.3.1 Automatic discovery of designs

The first possibility to be discussed is based on automatic discovery of designs with limited number of locations that have certain behavior with respect to the social norms. These designs, can be indexed and stored in a data base of solutions to be used for generating larger scale designs. The discoverer must be focused on enumerating all possible designs for problem of small sizes and select those with certain potential behavior for future use. Enumeration of small scale designs at topological (connectivity) level means enumeration of all possible planar graphs labeled with the activities related to the design. Generating all possible labeled graphs, even for graphs of up to 8 nodes is a complicated and time consuming task; but, if one is enumerating these solutions only once for the purpose of discovering the best solutions, then the suggested approach is worth consideration.

The discovered solutions, then, must be indexed according to their behavioral properties with respect to the social norms, and stored in a data base for use in generating more sophisticated designs.

Appendix C exhibits paper and pencil simulation of the automatic discovery process for cases of design problems with a few number of nodes. To save the time, the analysis results of the solutions are suppressed and the proposed solutions are judged and selected intuitively.

A system based on a data base of precedents of small scaled designs probably must use a model of design process based on the problem decomposition, discussed in chapter 2, to generate designs of larger sizes

with respect to the social norms. A design generation process based on this approach, decomposes a large scale design into sub-problems of specific characteristics matching one or more sub-problems. The solutions of the sub-problems are discovered and catalogued, the ready made solutions are selected, and composes into a global design for the whole problem.

This approach, has to take advantage of a data abstraction method such as Q-analysis used in conjunction with TOPGENE, for inferring hidden information from a design and hierarchical clustering of design activities (locations) for further processes. Hierarchical abstraction of a design provides a ground for decomposition of a design problem into smaller scaled problems. Data abstraction, as in the case of TOPGENE, is also a necessity for diagnosis and evaluation of the selected sub-designs, composed designs, and existing designs.

9.3.2 Search for fixed points

Another possibility based on the current capabilities and underlaying methods in TOPGENE is a system using a search method targeted for a set of fixed points corresponding to different social behaviors of a design. This method best can be described by looking at the following figure. In this figure, four curves Uco Cpr Ccc Uio are assumed to correspond to the community utility, privacy cost, circulation cost, and intervening opportunity utility of sub-optimal designs with respect to these norms. These curves may be assumed to correspond to a design formed on an iterative improvement basis from a null design up to a complete design. A null design, as indicated in the figure, has zero cost or utility associated with its behavior, and as a design grows towards a complete design consisting of all activities (locations) involved in the design statement the cost or the utility associated with it presumably increases. Each final complete sub-optimal design, thus have a total behavioral value corresponding to the norm under which it is generated. Based on the fact that some of these norm are in conflict, a design with respect to all these norms would have to have a behavior comprising of compromise between the behavioral values of sub-optimal designs. A natural decision on where to fix the compromising points are points A, B, C, and D, half-way from the optimal points on the curves

(values). But, were are these optimal points, when the sub-optimal solutions with respect to each norm is not known or it is hard to find? A fast solution to this problems is TOPGENE. The behavioral values of sub-optimal designs generated by TOPGENE could be taken as behavioral values of the actual sub-optimal designs with respect to these norms.

TOPGENE, for the proposed approach, can be also revised to generate a globally optimal design with respect to a combination of norms. The revised system, then, would start from a null partial design with its behavioral values with respect to the social norms set to zero, and it will iteratively improve the formed partial design by selecting, on each iteration, an appropriate unprocessed location. The goal of such a process would be to generate a design that has performance values as close as possible to the points A, B, C, and D on the curves corresponding to different social norms.

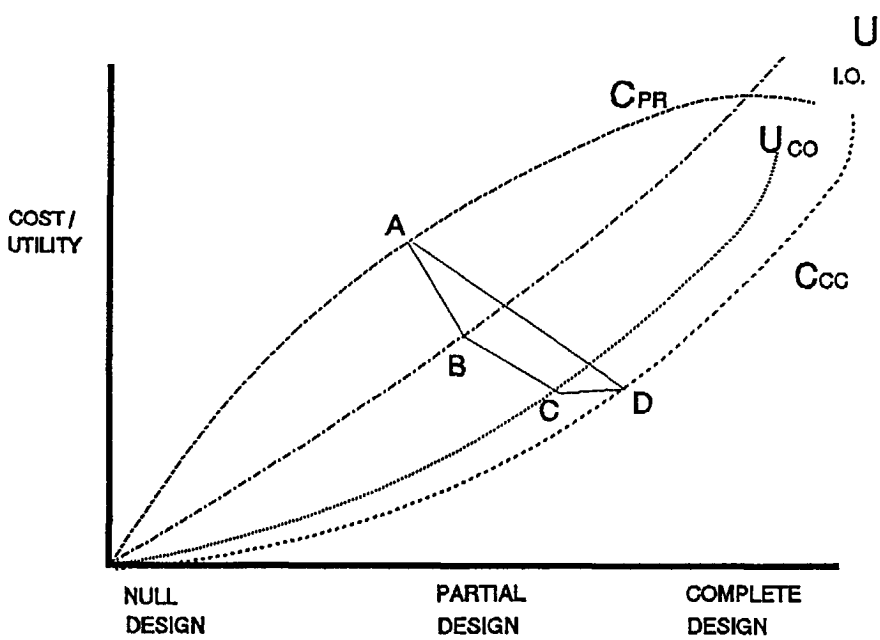


Figure 9.1 Curves representing performance values of sub-optimal designs with respect to social norms

The problems associated with this approach are as follows:

- The sub-optimal solutions generated by TOPGENE are based on heuristic strategy adapted by the system. They are only in the neighborhood of the optimal solutions. This means that the fixed points found, in turn, would be in the neighborhood of the actual fixed points.
- Careful thought is needed for adapting a search strategy while building partial designs towards a complete design. Such a search might not be allowed to run into complete enumeration of all possibilities.
- A final design, although a reasonable one in terms of trade-offs made between a set of conflicting norms, but may not a realistic one unless, as in the case of TOPGENE, the system is integrated with a knowledge base of recommended and prohibited accesses in buildings.

REFERENCES

A

[Aarts89]
Aarts, E., and J. Korst, **Simulated Annealing and Boltzmann Machines, A Stochastic Approach to Combinatorial Optimization and Neural Computing,** New York: John Wiley & Sons, 1989.

[Adler87]
Adler N., and L. Kovacvicn-Beck, **The Correlation between Physical Properties and Topological Indices of N-Alkanes,** *Graph Theory and Topology in Chemistry,* Amsterdam: Elsevier, pp. 194-200, 1987.

[Aho et al 74]
Aho, A., Hopcroft, J.E., and J.D. Ullman, **The Design and Analysis of Computer Algorithms,** Amsterdam: Addison-Wesley Publishing Co, 1974.

[Alexander77]
Alexander, C., **Notes on the Synthesis of Form,** Cambridge, Massachusetts: Harvard university Press, 1977.

[Atkin74a]
Atkin, R.H., **Mathematical Structure in Human Affairs,** New york: Crane, Russak and Co., Inc., 1974.

[Atkin74b]
Atkin, R.H., **An Approach to Structure in Architectural and Urban Design, 1. Introduction and Mathematical Theory,** *Environment and Planning B,* vol. 1, pp. 51-67, 1974.

[Atkin74c]
Atkin, R.H., **An Approach to Structure in Architectural and Urban Design, 2. Algebraic representation and local structure,** *Environment and Planning B,* vol. 1, pp. 173-191, 1974.

[Atkin75]
Atkin, R.H., **An Approach to Structure in Architectural and Urban Design, 3. Illustrative examples,** *Environment and Planning B,* vol. 2, pp. 21-57, 1975.

[Atkin77]
Atkin R.H., **Combinatorial Connectivities in Social Systems, An Application of Simplicial Complex Structures to the Study of Large Organizations,** Stuttgart: Birkhauser Verlag, 1977.

B

[Bachant84]
Bachant, J., and J. McDermott, **R1 revisited: Four years in the trenches,** *AI Megazine,* vol. 5, no. 3, pp. 21-32, 1984.

[Balaban87]
Balaban A.T., **Numerical Modeling of chemical Structures: Local Graph Invariants and Topological Indices,** *Graph Theory and Topology in Chemistry,* Amsterdam: Elsevier, pp. 159-176, 1987.

[Barker89]
Barker, V.E., and D.E. O'Conner, **Expert Systems for Configuration at Digital: XCON and Beyond,** Communication of the ACM, vol. 32, no. 3, pp. 298-318, 1989.

[Baybars80]
Baybars I., and C.M. Eastman, **Enumerating Architectural Arrangements by Generating their Underlaying Graphs,** London: *Environment and Planning B,* vol. 7, pp. 289-310, 1980.

[Bainekg83]
Bainekg, L.W., **Selected Topics in Graph Theory,** New york: Academic Press Inc., 1983.

[Berwick71]
Berwick, R., **Optimization of the Topological Organization of a Building, (1971),** reprinted in: *Frames, Plans, Representation,* Netherlands: Technical University of Delft, pp. 170-192, 1987.

[Bobrow86]
Bobrow D.G, Mittal, S., and M.J. Stefik, **Expert Systems: Perils and Promise,** Communication of ACM, vol. 29, no. 9, pp. 880-894, 1986.

[Bonchev87a]
Bonchev, D., and O.E. Polansky, **On the Topological Complexity of Chemical Systems,** *Graph Theory and Topology in Chemistry,* Amsterdam: Elsevier, pp. 126-157, 1987.

[Bonchev87b]
Bonchev D., O. Mekenyan and O.E. Polansky, **Some Relations Between the Wiener Number and the Number of Self-Returning Walks in Chemical Graphs,** *Graph Theory and Topology in Chemistry,* Amsterdam: Elsevier, pp. 209-218, 1987.

[Brachman85]
Brachman, R.J., Fikes, R.E., and H.J. Levesque, (Eds.), **Readings in Knowledge representation,** Los Altos, California: Morgan Kaufman Publishing Inc., 1985.

[Broadbent88]
Broadbent, G., **Design in architecture, Architecture and the Human sciences,** London: David Fulton Publishers Ltd., 1988.

[Buchanan84]
Buchanan, B, and E.H. Shortliffe, **Rule Based Expert Systems: The MYCIN Experiment of the Stanford Heuristic Programming Project,** Amsterdam: Addison- Wesley, 1984.

[Buckley90]
Buckley, F., and F. Harary, **Distances in Graphs,** Amsterdam: Addison-Wesley Publishing Co., 1990.

C

[Campbell86]
Campbell, J., **Principles of Artificial Intelligence,** *Artificial Intelligence, Principles and applications,* Yazdani, M. (Ed.), Amsterdam: Chapman and Hall Ltd., 1986.

[Carbonell83a]
Carbonell, J., **Derivational Analogy and its Role in Problem Solving,** *The Proceedings of National Conference on Artificial Intelligence,* The American Association for Artificial Intelligence (AAAI), pp. 64-69, 1983.

[Carbonell83b]
Carbonell, J.G., **Learning by Analogy: Formulating and Generalizing Plans from Past Experience,** *Machine Learning, An Artificial Intelligence Approach,*vol. I, Palo Alto, California: Tioga Publishing Co., 1983.

[Carbonell83c]
Carbonell, J.G., **An Overview of Machine Learning,** *Machine Learning, An Artificial Intelligence Approach,* vol. I, Palo Alto, California: Tioga Publishing Co., 1983.

[Charniak80]
Charniak, E., Riesbeck C.K., and D.V. McDermott, **Artificial Intelligence Programming,** New Jersey: Lawrence Erlbaum Associates, Publishers, 1980.

[Charniak85]
Charniak, E., Riesbeck, C.K., and D.V. McDermott., **Introduction to Artificial Intelligence,** Amsterdam: Addison-Wesley Publishing Co., 1985.

[Christofides75]
Christofides, N., **Graph Theory, An Algorithmic Approach,** New york: Academic Press Inc., Ltd., 1975.

[Cohen86]
Cohen P.R., **Heuristic reasoning about Uncertainty: An AI approach,** London: Pitman Publishing Ltd., 1986.

[Coyne88]
Coyne, R.D., **Logic Models of Design,** London: Pitman Publishing Ltd.,

1988.

[Coyne90]
Coyne, R.D., Rosenman, M.A., Radford, A.D., Balachandran, M., and J.D. Gero, **Knowledge-Based Design Systems,** Amsterdam: Addison-Wesley Publishing Co., 1990.

D

[Davis82]
Davis, R. H., **Diagnosis Based On Description of Structure and Function,** *The Proceedings of National Conference on Artificial Intelligence,* The American Association for Artificial Intelligence (AAAI), 1982.

[Davis et al 82]
Davis, R., and D.B. Lenat, **Knowledge-Based Systems In Artificial Intelligence,** New York: McGraw-Hill International Book Company, 1982.

[Davis83]
Davis, R., **Negotiation as a Metaphor for Distributed Problem Solving,** *Artificial Intelligence,* vol. 20, pp. 63-109, Amsterdam: North-Holland Publishing Co., 1983.

[Davis84]
Davis, R., **Diagnostic Reasoning Based on Structure and Behaviour,** *Artificial Intelligence,* vol. 24, pp. 374-410, Amsterdam: North-Holland publishing Co., 1984.

[Dial79]
Dial, R., Glover, F., Karney D., and D. Klingman, **A Computational Analysis of Alternative Algorithms and Labeling techniques for finding Shortest Path Trees,** *Networks,* vol. 9, pp. 215-248, John Wiley & Sons, Inc., 1979.

[Dougherty88]
Dougherty E.R., and C.R. Giardina., **Mathematical Methods for Artificial Intelligence and Autonomous Systems,** Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1988.

[Durbin89]
Durbin, R., **Network Models and the Nervous System,** *The Computing Neuron,* Durbin, R., Mial, C., and G. Mitchison (Eds.), Amsterdam: Addison-Wesley Publishing Co., 1989.

E

[Eastman73]
Eastman, C.M., **Automatic Space Planning,** *Artificial Intelligence,* vol. 4, pp. 41-64, Amsterdam: North-Holland Publishing Co., 1973.

[Evans68]
Evans T. G., **A Program for the solution of a class of Geometric-**

Analogy, Intelligence-Test Questions, *Semantic Information Processing,* M. Minsky (Ed.), Cambridge, Massachusetts: MIT Press, 1968.

[Even79]
Even, S., **Graph Algorithms,** London: Pitman Publishing Ltd., 1979.

F

[Farbestein69]
Farbestein, J., **Some aspects of Activity and Spatial Analysis in the Micro- region,** Cambridge, Massachusetts: Harvard University, Graduate School of Design, Thesis, 1969.

[Findler81]
Findler N.V., **Analogical Reasoning in Design Process,** *Design Studies,* vol. 2, no. 1, Business Press Ltd., 1981.

[Fisher66]
Fisher, G.J., and O. Wing, **Computer Recognition and Extraction of Planar Graphs from the Incident Matrix,** IEEE Transaction on Circuit Theory, vol. Ct- 13, no. 2, pp. 154-197, 1966.

[Fisher87]
Fisher , M. A., and O. Firschein., **Intelligence: the Eye, the Brain, and the Computer,** Amsterdam: Addison-Wesley Publishing Co., 1987.

[Flament63]
Flament, C., **Application of Graph Theory to Group Structure,** Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1963.

[Flegg74]
Flegg, H.G., **From Geometry to Topology,** London: The English University Press Ltd., 1974.

[Flemming78]
Flemming, U., **Wall representations of rectangular dissections and their use in automated space allocation,** *Environment and Planning B,* vol. 5, pp. 215- 232, 1978.

G

[Galle81]
Galle, P., **An Algorithm for Exhaustive Generation of Building Floor Plans,** *Communication of ACM,* vol. 24, no. 12, 1981.

[Gary79]
Gary, M., and D.S. Johnson, **Computers and Intractability, A guide to the Theory of NP-Completeness,** San Francisco: W.H. Freeman and Co., 1979.

[Genesereth82]
Genesereth, M.R., **Diagnosis using Hierarchical Design Models,**

Proceedings of National Conference on Artificial Intelligence (AAAI), 1982.

[Genesereth84]
Genesereth, M.R., **The use of Design Descriptions in Automated Diagnosis,** *Artificial Intelligence,* vol. 24, pp. 411-436, Amsterdam: North-Holland Publishing Co. 1984.

[Genesereth87]
Genesereth, M.R., and N.J. Nilsson, **Logical Foundations of Artificial Intelligence,** Los Altos, California: Morgan Kaufman Publishers, Inc., 1987.

[Georgeff83]
Georgeff, M.P., **Strategies in Heuristic Search,** *Artificial Intelligence*, vol, 20, pp. 393-425, 1983.

[Gero90]
Gero, J., **Design Prototypes: A Knowledge Representation Schema for Design,** AI Magazine, pp. 27-36, Winter 1990.

[Gibbon85]
Gibbons, A., **Algorithmic Graph Theory,** Massachusetts: Cambridge University Press, 1985.

[Gillard78]
Gillard, J., **LAYOUT - A Hierarchical Computer Model for the Production of Architectural Floor Plans,** *Environment and Planning B,* vol. 5, pp. 233-241, 1978.

[GCLISP87]
Golden Common LISP Programmer Reference Manual, U.S.A.: Gold Hill Computers, Inc., 1987.

[Goel89]
Goel V. and P. Pirolli, **Design with Information Processing Theory: The Design Problem space,** AI Magazine, vol. 10, no. 1, 1989.

[GoldWorks87]
GoldWorks Expert System Reference Manual, U.S.A.: Gold Hill Computers, Inc., 1987.

[Gordon84]
Gordon, J., and E.H. Shortliffe, **A Method for Managing Evidential Reasoning in a Hierarchical Hypothesis Space,** California: Stanford University, Departments of Medicine and Computer Science, Report no. Stan-CS-84-1023., 1984.

[Gross87]
Gross, J. L., and T.W. Tucker, **Topological Graph Theory,** New York: John Wiley & Sons, 1987.

H

[Haggett69]
Haggett, P. and R.F. Chorley, **Network Analysis in Geography,** London: Edward Arnold., 1969.

[Hanson87]
Hanson M.P. and Rouvray D.H., **The Use of Topological Indices to Estimate the Melting point of Organic Molecules,** *Graph Theory and Topology in Chemistry,* Amsterdam: Elsevier, pp. 201- 208, 1987.

[Harrary71]
Harrary, F., **Graph Theory,** Amsterdam: Addison-Wesley Publishing Co., 1971.

[Harfmann87]
Harfmann, A.C., **The Rationalizing of Design,** *Computability of Design,* Kalay, Y. E. (Ed.), New York: John Wiley & Sons., 1987.

[Hashimshony86]
Hashimshony R., and J. Roth., **ALG: a Model for Generating Alternative Layout Graphs under Architectural Constraints,** *Computer-aided Design.* vol. 18, no. 8, pp. 431-436, 1986.

[Hashimshony88]
Hashimshony R., and J. Roth., **Algorithm in graph theory and their use for solving problems in architectural design,** *Computer-aided Design.* vol. 20, no. 7, pp. 373-381, 1988.

[Hayes-Roth85]
Hayes-Roth, B., **A Blackboard Architecture for Control,** *Artificial Intelligence,* vol. 26, no. 3, pp. 251-321, 185.

[Herik88]
Van den Herik, H.J., **Expert Configuring,** *USER!, Proceedings*, Simulation councils, pp. 255-257, San Diego, USA, 1988.

[Hoekstra90]
Hoekstra90, J., **Simulating Modular Neural Networks Using 400 processors,** *IEEE symposium on Neural networks,* Delft, The Netherlands: Delft University of Technology, 1990.

[Hopfield81]
Hopfield, J.J., **Neurons with graded response have collective computational properties like those of two-states neurons**, Proceedings of the National Academy of Sciences, pp. 3088-3092, 1981.

[Hopfield82]
Hopfield, J.J., **Neural networks and physical systems with emergent collective computational abilities**, Proceedings of the National Academy of Sciences, USA, vol. 79, pp. 2554-2558, 1982.

[Hopfield85]
Hopfield, J.J., and D.W. Tank, **Neural Computation of Decisions in**

Optimization Problems, Biol. Cybern., vol. 52, pp. 141-152, 1985.

[Hopfield86]
Hopfield, J.J., and D.W. Tank, **Computing with Neural Circuits: A Model,** Science, vol. 233, pp. 625-633, August 1986.

I

[Iwasaki86]
Iwasaki, Y., and H. A. Simon, **causality in device Behavior,** *Artificial Intelligence,* vol. 29, pp. 3-32, 1986.

J

[Jackson74]
Jackson, P.C., **Introduction to Artificial Intelligence,** New York: Petrocelli Books, 1974.

[Jackson86]
Jackson, P., **Introduction to Expert Systems,** Amsterdam: Addison-Wesley Publisher Ltd., 1986.

[Jayakumar89]
Jayakumar, K., Thulasiraman, K., and M.N.S. Swamy, **$O(n^2)$ Algorithms for Graph Planarization,** *Lecture notes in Computer Science (344),* J. Van Leevwen (Ed.), Berlin: Springer-Verlag, 1989.

[Johnson87]
Johnson M., Naim, M., Nicholson, V., and C.-C. Tsai, **Unique Mathematical Features of the Substructure Metric Approach to Quantitative Molecular Similarity Analysis,** *Graph Theory and Topology in Chemistry,* Amsterdam: Elsevier, pp. 219-225, 1987.

[Jonathan87]
Jonathan L. G., and T.W. Tucker, **Topological Graph Theory,** New york: John Wiley & Sons., 1987.

K

[Kalay87]
Kalay, Y. E. (Ed.), **Computability of Design,** New York: John Wiley & Sons, 1987.

[Karni86]
Karni, S., **Analysis of Electrical Networks,** New York: John wiley & Sons, 1986.

[Kier76]
Kier, L.w, and Lowel H. Hall, **Molecular Connectivity in Chemistry and Drug research,** New York: Academic Press Inc. Ltd., 1976

[King87]
King, R.B. and D.H. Rouvray, (Eds.), **Graph Theory and Topology in Chemistry**, Amsterdam: Elsevier, 1987.

[Klahr86]
Klahr, P., and Waterman, D.A., (Eds.), **Expert Systems Techniques, Tools and Applications**, Amsterdam: Addison-Wesley Publishing Co., 1986.

[Klein87]
Klein, D., and T. Finin, **What's in a Deep Model?, A characterization of Knowledge Depth in Intelligent safety Systems**, IJCAI 87, vol. 1, pp. 559-562, 1987.

[Klein89]
Klein, m., and S.C.-Y. Lu, **Conflict Resolution in Cooperative Design**, *Artificial Intelligence in Engineering*, Southampton UK: Computational Mechanics Publications, vol. 4, no. 4, pp. 168-180, 1989.

[Korf77]
Korf, R.E. **A Shape Independent Theory of space Allocation**, *Environment and Planning B*, vol. 4, no. 1, pp. 37-50, 1977.

[Koppelaar90]
Koppelaar, H., **Automatic Discovery**, *IEEE symposium on Neural networks*, Delft, The netherlands: Delft University of Technology, 1990.

[Krishnamurti78]
Krishnamurti, R., and P.H. O'N Roe, **Algorithmic aspects of plan generation and enumeration**, *Environment and Planning B*, vol. 5, pp. 157-177, 1978.

[Kuipers84]
Kuipers, B. **Commonsense Reasoning about Causality: Deriving Behaviour from Structure**, *Artificial Intelligence*, vol. 24, pp. 169-203, Amsterdam: North- Holland Publishing Co. 1984.

L

[Lampel67]
Lampel, A., Even, S., and I. Cederbaum, **An Algorithm for planarity Testing of Graphs**, *Theory of Graphs*, International Symposium: Rome, July 1966, P. Rosenthal (Ed.), New York: Gordon and Breach, pp. 215-232, 1967.

[Langley87]
Langley, P., Simon H.A., Bradshaw G.L., and J.M. Zytkow, **Scientific Discovery, Computational Explosions of the Creative Process**, Cambridge, Massachusetts: The MIT Press, 1987.

[Lenat82]
Lenat D.B., **The Nature of Heuristics**, *Artificial Intelligence*, vol. 19, pp. 189-249, Amsterdam: North-Holland, 1982.

[Li-Min85]
Li-Min Fu and B. G. Buchanan, **Inductive Knowledge Acquisition for Rule-Bases Expert Systems,** California: Stanford University, Department of Computer Science, 1985.

[Lindsay85]
Lindsay, R.K., Buchanan, B., E. Feigenbaum, and J. Lederberg, **Application of Artificial Intelligence for Organic Chemistry,** Los Altos, California: Morgan Kaufman Publishers, Inc., 1985.

M

[Maher85]
Maher, M.L., and S. Fenves, **HI-RISE: An Expert System for Preliminary Structural design of High Rise Buildings,** *Knowledge Engineering in Computer- Aided Structural Design,* Gero, J. (Ed.), Amsterdam: North-Holland, pp. 125- 146, 1985.

[McCarthy59]
McCarthy, J., **Programs with Common Sense (1959),** *Semantic Information processing,* Minsky, M. (Ed.), Cambridge, Massachusetts: MIT Press, pp. 403-417, 1968.

[McCarthy81]
McCarthy, J., and P. J. Hayes, **Some Philosophical Problems from the Standpoint of Artificial Intelligence,** (1969), *Readings in Artificial Intelligence,* Webber, B.L., and N.J. Nilsson (Eds.), Palo Alto, California: Tioga Publishing Co., 1981.

[McDermott82]
McDermott, J., **R1: A Rule-based Configurer of Computer Systems,** *Artificial Intelligence,* vol. 19, no. 1, 1982.

[Michalski83]
Michalski, R.S., **A Theory and Methodology of Inductive Learning,,** *Machine Learning, An Artificial Intelligence Approach,* vol. I, Michalski, R.S. Carbonell J.G., and T.M. Mitchell (Eds.), Palo Alto, California: Tioga Publishing Co., 1983.

[Michalski86]
Michalski, R.S., **Understanding the Nature of Learning: Issues and research directions,** *Machine Learning, An Artificial Intelligence Approach,* Michalski, R.S. Carbonell J.G., and T.M Mitchell (Eds.), vol. II, Palo Alto, California: Tioga Publishing Co., 1986.

[Minsky68]
Minsky, M. (Ed.), **Semantic Information Processing,** Cambridge, Massachusetts: MIT Press, 1968.

[Minsky75]
Minsky, M., **A Framework for Representing Knowledge,** *The Psychology of Computer Vision,* Winston P.H. (Ed.), New York: McGraw-Hill Book Company, 1975.

[Mitchell75]
Mitchell, W.J., **The theoretical foundation of computer-aided architectural design,** *Environment and Planning B,* vol. 2, pp. 127-150, 1975.

[Mitchell et al 76]
Mitchell, W.J., Steadman, I.P., and R.S. Liggett, **Synthesis and optimization of small rectangular floor plans,** *Environment and Planning B,* vol. 3, pp. 37-70, 1976.

[Mitchell et al 86]
Mitchell, T.M., R.M. Keller, and S.T. Kedar-Cabelli, **Explanation-based generalization: a unified view,** *Machine Learning,* vol. 1, no. 1, pp. 47-80, 1986.

[Mittal85]
Mitall, S., Dym, C., and M. Morjaria, **Pride: An Expert System for the Design of Paper Handling Systems,** *Application of Knowledge Based Systems to Engineering Analysis and Design,* C.L. Dym (Ed.), New york: American Society of Mechanical Engineers, 1985

[Mostow85]
Mostow, J., **Towards Better Models of the Design Process,** *AI Magazine,* vol. VI, pp. 44-56, 1985.

[Mostow87]
Mostow, J., and N. Bhatnagar, **Failsafe- A Floor Planner that Uses EBG to Learn from its Failures,,** *IJCAI,* pp. 249-255, 1987.

N

[Newell69]
Newell, A., **Heuristic Programming: ill-structured problems,** *Progress in Operations Research,* Aronofsky, J. (Ed.), vol. 3, pp. 363-413, New York: John Wiley, 1969.

[Nicholson87]
Nicholson V., Tsai, C.-C., Johnson, M., and M. Naim, **A subgraph isomorphism Theorem for Molecular Graphs,** *Graph Theory and Topology in Chemistry,* Amsterdam: Elsevier, pp. 226-229, 1987.

[Nilsson71]
Nilsson, N. J., **Problem Solving Methods in Artificial Intelligence,** New york: McGraw-Hill Book Co., 1971.

[Nishiziki88]
Nishiziki, T., and N. Chiba, **Planar Graphs: Theory and Practice,** Amsterdam: North-Holland Publishing Co., 1988.

O

[Ozawa81]
Ozawa, T., and H. Takahashi, **A Graph-Planarization Algorithm and its**

Application to Random Graphs, *Lecture notes in Computer Science (108),* Saito, N., and T. Nishiziki (Eds.), Berlin: Springer-Verlag, 1981.

P

[Papadimitriou82]
Papadimitriou, C.H.,and K. Steiglitz, **Combinatorial Optimization,Algorithms and Complexity,** Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1982.

[Parrello88]
Parrello, B. **The (Almost) Birth of an Expert System,** *AI expert,* pp. 60-64, January 1988.

[Pople84]
Pople, H.E. **CADUCEUS: An Experimental expert System for Medical Diagnosis,** *The AI Business, The Commercial use of Artificial Intelligence,* Winston, P.H., and K. A. Prendergast, (Eds.), Cambridge, Massachusetts: MIT Press, 1984.

R

[Reingold77]
Reingold E. M., Nievergelt, J., and D. Narsingh, **Combinatorial Algorithms, Theory and Practice,** Englewood Cliffs, N.J.: Prentice-Hall Inc., 1977.

[Reinolds80]
Reinolds, T. **Computer Methods for Architects,** London: Butterworth & Co. Publishers Ltd., 1980.

[Rich83]
Rich, E, **Artificial Intelligence**, Singapore: McGraw-Hill Book Co., 1983.

[Roch78]
Roch, J. **Interactive Space Layout: A Graph Theoretical Approach,** *Design Automation Conference precedings 15,* IEEE/ACM, SIGDA, pp. 152-157, 1978.

[Rosenman90]
Rosenman, M.A., **Application of Expert Systems to Building Design Analysis and Evaluation,** *Building & Environment,* vol. 25, no. 3, pp. 221-233, 1990.

[Rouvray86]
Rouvray D.H., **Predicting Chemistry from Topology,** Scientific American, pp.36-43, 1986.

[Roth82]
Roth J., R. Hashimshony, and A. Washman, **Turning a Graph into a Rectangular Floor Plan,** *Building & Environment.* vol. 17. No 3, pp. 163-

173, 1982.

[Roth88]
Roth J., R. Hashimshony, and A. Washman., **Algorithms in Graph Theory and their use for Solving Problems in Architectural Design,** Computer-aided Design, vol. 20, no. 7, 1988.

S

[Schubert76]
Schubert, L.K. **Extending the Expressive Power of Semantic Networks,** *Artificial Intelligence*, vol. 7, pp. 163-198, Amsterdam: North-Holland Publishing Company, 1976.

[Seidl86]
Seidl, R., **Graphs, Analogy, and Representation,** Delft, The Netherlands: Technical University of Delft, 1986.

[Shortliffe76]
Shortliffe, E. H., **Computer-Based Medical Consultations: EMYCIN,** New York: American Elsevier, 1976.

[Simon71]
Simon H. A., **The Sciences of Artificial,** Second Edition, 1981, Cambridge, Massachusetts: MIT Press, 1988.

[Simon73]
Simon H.A., **The Structure of Ill-structured Problems,** *Artificial Intelligence,* vol. 4, pp. 181-201, 1973.

[Slagle71]
Slagle, J. R., **Artificial Intelligence: The Heuristic Programming Approach,** New York: McGraw-Hill Book Co., 1971.

[Slatter87]
Slatter, P. E., **Building Expert System: Cognitive Emulation,** London: Ellis Horwood Ltd., 1987.

[Sriram87]
Sriram S., **knowledge-Based Approaches for Structural Design,** Southampton UK: Computational Mechanics Publications, 1987.

[Steele84]
Steele Jr., Guy L., **Common LISP The Language (Reference manual),** United State of America: Digital Equipment Corporation, 1984.

[Steadman76]
Steadman J. P., **Graph-theoretic Representation of Architectural Arrangement,** *The Architecture of Form,* March, L. (Ed.) London: Cambridge University Press, 1976.

[Steadman83]
Steadman J. P., **Architectural Morphology, An Introduction to the**

Geometry of Building Plans, London: Pion publication Ltd., 1983.

[Struss88]
Struss, P., **Extensions to ATMS-based diagnosis,** *Artificial Intelligence in Engineering: Diagnosis and Learning,* Gero, J.S. (Ed.), Amsterdam: Elsevier, 1988.

T

[Tzonis75]
Tzonis, A., and O. Salama, **Problems of Judgment in Programmatic Analysis in Architecture: The synthesis of partial evaluations,** (1975), reprinted in: *Frames, Plans, representation,* Delft, The Netherlands: Technical University of Delft, 1987.

[Tzonis85]
Tzonis, A., and G.R. Scherpbier, **Intelligent Architect Project,** The Netherlands: Technical University of Delft, 1985.

[Tzonis87]
Tzonis, A., **Frames, Plans, representation,** Netherlands: Technical University of Delft, 1987.

[Temperley81]
Temperley, H. N. V., **Graph Theory and Applications,** London: Ellis Horwood Ltd., 1981.

[Trinajstic83]
Trinajstic, N., **Chemical Graph Theory,** vol 1, Boca Raton, Florida: CRC Press Inc., 1983.

[Tsai87]
Tsai C.-C., Johnson, M., Nicholson, V., and M. Naim, **A Topological Approach to Molecular-Similarity Analysis and its Application,** *Graph Theory and Topology in Chemistry,* Amsterdam: Elsevier, pp. 231-235, 1987.

W

[Watanabe89]
Watanabe, H., **Heuristic graph displayer for G-Base,** Int. J. Man-Machine Studies, vol. 30, pp. 287-302, Academic Press Ltd., 1989.

[Wilensky84]
Wilensky, R., **LISPcraft,** New York: W. W. Norton & Co., 1984.

[Wilf86]
Wilf H. S., **Algorithms and Complexity,** Englewood Cliffs, N.J.: Prentice-Hall Inc., 1986.

[Willoughby71]
Willoughby, T., **Evaluating circulation performance,** *Architectural*

Design, vol. 5, p. 314, 1971.

[Wilson85]
Wilson, R. J., **Introduction to Graph Theory,** London: Longman Scientific & Technical, 1985.

[Wilson88]
Wilson, G. V., and G.S. Pawley, **On the stability of the Travelling Salesman Problem Algorithm of Hopfield and Tank,** Biological Cybernetics, vol. 58, pp. 63-70, 1988.

[Winograd75]
Winograd, T., **Frame Representations and the Declarative / Procedural Controversy,** *Representation and Understanding,* D. G. Bobrow & Collins (Eds.). New York: Academic Press Inc. Ltd., 1975.

[Winston79]
Winston, P.H, **Learning by Understanding Analogies,** MIT AI Memo 520, 1979.

[Winston81]
Winston, P.H., **Learning New Principles From Precedents and Exercises: The Details,** MIT AI Memo 632, 1981.

[Winston83]
Winston, P.H., **Learning Physical Descriptions from Functional Definitions, Examples and Precedents,** MIT AI Memo 679, 1983.

[Winston84a]
Winston, P.H., **Artificial Intelligence,** Amsterdam: Addison-Wesley, 1984.

[Winston84b]
Winston, P.H., and K.A. Prendergast, (Eds.), **The AI Business, The Commercial use of Artificial Intelligence,** Cambridge, Massachusetts: MIT Press, 1984.

[Winston85]
Winston, P.H., **Learning Structural Description from Examples,** *Reading in Knowledge Representation,* Brachman, R., and H. Levesque (Eds.), Los Altos, California: Morgan Kaufman Publishers, Inc., 1985.

[Winston89b]
Winston, P.H., and B.K.P. Horn., **LISP,** Amsterdam: Addison-Wesley Publishing Co., 1989.

Y

[Yazdani86]
Yazdani, M., **Artificial Intelligence, Principles and applications,** Amsterdam: Chapman and Hall Ltd., 1986.

Z

[Zeidenberg90]
Zeidenberg, M., **Neural Networks Models in artificial Intelligence,** London: Ellis Horwood Ltd., 1990

APPENDIX-A

EXAMPLES OF TOPOLOGICAL INDICES (TIs)

This appendix presents the definition and properties of a number of topological indices (TIs), identified in relation to the structural characteristics of graphs. TIs, introduces in chapter 4, are of extreme use in design and development of precedent based automatic systems capable of indexing, storing, and retrieving existing architectural design solutions.

A.1 TIs based on the structural connectivity in a graph

Most of the topological indices introduced in literatures are either based on the connectivity relationships, or the topological distances in graphs. This implies that a large number of TIs are derivable from the adjacency matrix, and the distance matrix corresponding to a graph.

A.1.1 The (global) connectivity index

The simplest topological index, reflecting the topological complexity of a graph, is the number of its components. A connected graph is intuitively more complex than a disconnected one. Therefore, such an index should have an inverse relation with number of components of a graph. A (global) connectivity index for a graph, thus, can be defined [Bonchev87] as:

$$C_{con}(G) = \frac{1}{k}, \text{ where k is the number components of G}$$

Clearly, the relation $0 < C_{con} \leq 1$ always hold for a graph G, where C_{con} (G) is equal to 1 if G is connected, and $C_{con}(G)$ decreases with increase in the number of components of G.

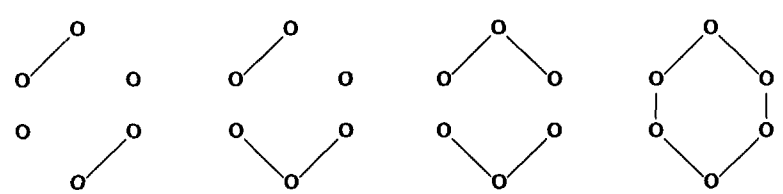


Figure A.1: Connectivity index for different graphs of the same order

A.1.2 The connectivity index β.

Another measure of connectivity in a graph is the β-index. This index is a simple yardstick for measuring the connectivity of a connected graph G β-index relates the number of vertices n to the number of edges m in G. One way of calculating the β-Index is by dividing m with n [Haggett69].

$$\beta = m / n$$

A graph with a high β-index is structurally more complex than a graph with a lower β-index. furthermore:

If β-index < 1.0, then the graph is a tree.
If β-index = 1.0, then the graph has only one circuit.
If β-index > 1.0, then the graph is a general graph.

This index, thus, can be used for distinguishing basic classes of graphs, and estimating their structural complexities.

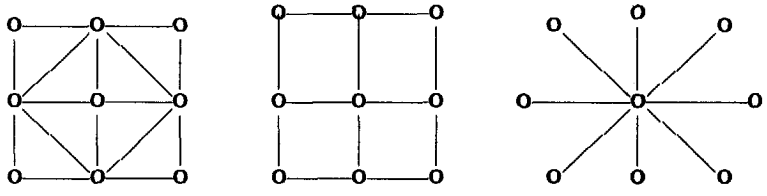


Figure A.2: Connectivity complexity of graphs measured by β-index

A.1.3 Euler's formula and Cyclomatic number

Euler [Trinajstic83a] discovered that for any connected planar graph with n vertices, m edges, and f faces, the relation m-n-+2=f always holds. This relation for disconnected planar graphs with k components is m-n+k+1=f. Based on the above relation we have the cyclomatic-number f(G), which is the number of face-cycles in a graph:

$$f(G) = m - n + k$$

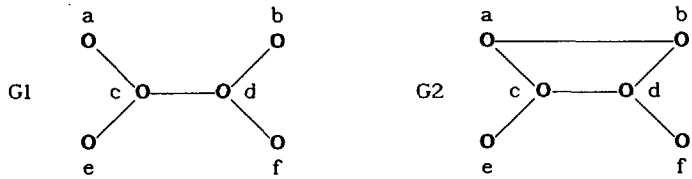
This formula holds and yields more structural information (e.g., number of cycles) in a graph than the connectivity indices discussed.

A.1.4 Randic's molecular-connectivity index

This index, introduced by M. Randic in 1975 [Balaban87], is based on the topological concept of vertex degree. This index is a valuable index in chemistry that is used in developing new drugs, modeling toxicity, and even predicting taste and smell of new substances yet to be engineered [Rouvray83]. This index is defined as follows:

$$X(G)=\sum_{1}^{m} [d_i(v) * d_j(v)]^{-(1/2)}, \text{ for } i \neq j.$$

Where m is the number of edges in a graph G, and $d(v_i) * d(v_j)$ is the multiplication of branching degree of adjacent vertices in G. Randic's index increases with increase in branchiness degree in G.



$$X(G1)=4*(\frac{1}{\sqrt{1*3}})+(\frac{1}{\sqrt{3*3}})=2.64$$

$$X(G2)=2*(\frac{1}{\sqrt{1*3}})+(\frac{1}{\sqrt{3*3}})+(\frac{1}{\sqrt{2*2}})+2*(\frac{1}{\sqrt{2*3}})=3.62$$

Figure A.3: Randic's connectivity index

Randic's connectivity index can be generalized for all paths of various length in a graph [Trinajstic83b] as follows:

$$X_p(G)=\sum_{paths} [d(v_1)*d(v_j)...*d(v_{t+1}]^{-(1/2)} \quad 1\#J\# \ ... \ \#t+1$$

where, $d(v_1) * ...* d(v_{t+1})$ are the vertex degrees in a path p.

Derivable from this formula are the zero-order connectivity index, and the first-order connectivity index (the original Randic's connectivity index):

$$X_0(G) = \sum^{n} [d(v_i)]^{-(1/2)}, \text{ where n is the number of vertices in G.}$$

A.1.5 Adjacency index

The vertex adjacency is also fundamental, after the connectivity index, in determination of structural complexity in a graph. Adjacency of a graph is represented by a square matrix having entries $a_{ij}=1$ for the pairs of adjacent vertices and 0 otherwise. Such a matrix is symmetric if the graph representing is undirected. The total adjacency of a graph G, denoted by A(G) is the sum of entries in the adjacency matrix that represents G:

$$A(G) = \sum a_{ij} = \sum d(v_i)$$

A(G) is equal to the sum of all vertex degrees in G. If G is a simple connected graph (no loops and multiple edges), then, The A(G) range for connectedness is:

$$m \leq A(G) \leq 2m \qquad (m = \text{total number of edges})$$

The m bound corresponds to simple connected digraphs, and the 2m bound corresponds to undirected graphs. A(G) is generally higher for undirected graphs than for directed graphs with the same number of vertices and edges. A(G) for completely connected graphs (k-graphs) are:

$A(G_k) = n(n-1)$, where n is the number of nodes in G_k.

A(G) is a primary topological index which increases with the number of cycles in a graph.

With above points in mind, the following inequalities for graphs with constant number of vertices holds:
A(acyclic graphs) < A(cyclic graphs)
A(digraph) ≤ A(undirected graphs)
A(simple graphs) ≤ A(multi-graphs)

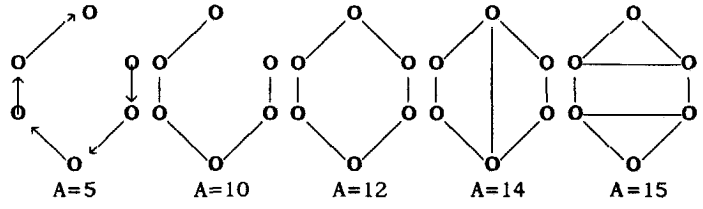


Figure A.4: Increase in complexity, reflected by total adjacency

A.1.6 Relative adjacency(RA) index
The total adjacency of graphs reflects the size of the graph besides their adjacency relations. This means that the adjacency relations of graphs of different size cannot be compared. The reason is that a comparison will be misleading because of the size effect. To avoid the size effect from the adjacency measure, relative adjacency is recommended. Relative adjacency of a graph G is obtainable by normalization of its total adjacency by total adjacency of complete graph of the same size (G_k).

$$RA(G) = \frac{A(\ )}{A(G_k)} = \frac{A(G)}{n(n-1)}$$

where, $0 \le RA(G) \le 1$ for simple graphs.

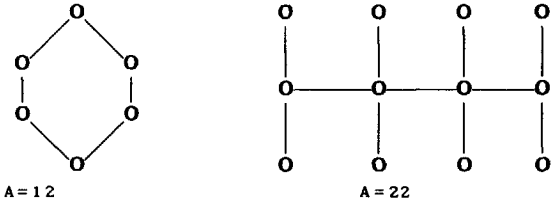


Figure A.5: Two graphs and their total and relative adjacencies

The trace of an adjacency matrix, which is the sum of diagonal elements, is a graph invariant denoting the number of first order loops in a graph. [Trinajstic-83a].

$$\text{Tr } A = \sum (a_{ii})$$

The trace of powers of an adjacency matrix is also a graph invariant that is equal to twice the number of edges in G.

$$\text{Tr } A^2 = \sum (a^2_{ii}) = \sum d(v_i) = 2m$$

Several graphs with different characteristics may map to a single TI. This problem, discussed in chapter 4, exists with most of the TIs including the total and relative adjacencies. To have an index with a higher discriminating power, an index with a higher level of complexity is needed.

The relative adjacency may be extended to multi-graphs as well. It is obvious that some multi-graphs, for example complete multi-graphs, will have RA larger than unity.

A.1.7 Branching degree

The branching-degree, also called branchedness or valancy, is among the well known graph invariants. Branching-degree is derivable both for vertices of a graph and the graph as a whole. The valancy of a vertex v_i, denoted by $d(v_i)$, is defined as the neighborhood of v_i. $d(v_i)$ is equal to the sum of the elements in i-th row (i.e., summation index j) of the adjacency matrix A(G), which is equivalent to the multiplication of row A_i by column A^T_i.

$$d(vi) = \sum_j a_{ij} = \sum_j a_{ij}\, a^T_{ji} = (a^2_{ii})$$

The branching degree of a graph, can be taken as the number of locations of degree one [Tzonis87], or the average of branching-degree of all vertices, depending on the application. The latter is sometimes called *the mean-local-degree* of a graph [Haggett69].

$$\text{Av. } d(G) = [\sum d(v_i)]/n, \text{ where n is the order of G.}$$

Vertices with identical graph invariants are said to be *topologically equivalent.* Similarly a graph whose vertices are all non-equivalent is called an *identity-graph* [Balaban87].

A.1.8 The Zagreb Group Indices

The following indices, devised for measuring the π-energy of chemical molecules [Trinajstic83b] [Bonchev83], are based on the branching degree, of vertices. The first index is calculated over all vertices of a graph G, while the second one is over all edges in G. Both of these indices monotonically increases with increase in the branchiness complexity of a graph.

$$M1(G) = \sum_i^n d^2(v_i), \quad \forall n \text{ nodes in G.}$$

$$M2 = \sum_{(i,j)}^m d(v_i) * d(v_j), \quad \forall m \text{ edges in G.}$$

A.1.9 The comparability index

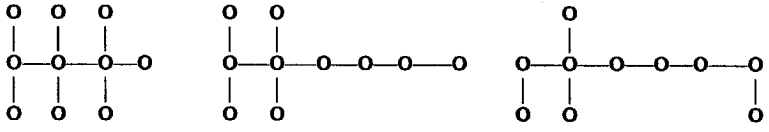
This index, introduced by Gutman and Randic [Trinajstic83b], uses the algebraic concept of comparability of functions to compare branching degrees of graphs. The modified version of this index is described by Trinajstic as follows:

Let $V=v_1, v_2, v_3, v_4, ..., v_n$, and $V'=v_1, v'_2, v'_3, ..., v'_n$ be two non-increasing sequences of the same length of natural numbers representing the degrees of vertices in two graphs. If V and V' fulfill the following conditions, then the sequence V is said to precede V, otherwise they cannot be compared.

(1) $\sum^{n} v_i = \sum^{n} v'_i$

(2) $\sum^{n} d(v_i) \geq \sum^{n} d(v'_i)$, $\forall\ v_i \in G$.

In the following example, G1 could be compared with G2 and G3, and it can be ordered to precede them (i.e., it is more branched). G2 and G3 cannot be compared, since the number of degree two vertices in G3 exceeds the number of vertices of the same size in G2.



G1: [4431111111] G2: [4322211111] G3: [4222211111]

Figure A.6: Comparability index for 3 different graphs

A.2 TIs based on the topological distances in a graph

The topological distance between two vertices v_i and v_j, denoted by $d(v_i\ v_j)$ or simply d_{ij}, is by definition the number of edges (or nodes) on the shortest path between v_i v_j. There are many topological indices based on topological distances in a graph, some of which are described below.

A.2.1 Shape index

The shape index for a graph G is obtainable by division of the total distance in G to its diameter. This index has an inverse relation with the compactness of a graph. The shape index is calculated in transportation networks by dividing the total mileage of a network by its diameter [Haggett69].

$$SI(G) = [\ \sum_i \sum_j d_{ij}\ ] / d(G)$$

A.2.2 Dispersion index

The dispersion index for a graph G is the summation of distances between all node pairs in G [Haggett69].

$$Disp(G) = \sum_i \sum_j d_{ij}$$

A.2.3 Average distance index

The average distance between all pairs of location can be calculated by dividing the summation of the length of all possible paths by the number of the paths. This index has an inverse relation with connectedness and compactness of a graph. The average distance can be expressed qualitatively as short, medium, or long, by comparing it with the total distance of a graph.

$$Av.D(G) = [\sum_{i} \sum_{j} d_{ij}]/\rho, \text{ where } \rho \text{ is the number of paths in G.}$$

A.2.4 Distance sum (accessibility) index

The distance sum index (S_i), also called the *vertex accessibility index* [Haggett69], for a node $v_i \in V(G)$ is the summation of distances from v_i to all other nodes in G. s_i for a vertex v_i is obtainable from a distance matrix D by adding all entries on its row.

$$S_i = \sum_{j}^{n} d_{ij}$$

Above accessibility measure relate to the paths originating from different locations. A similar accessibility measures can be obtained in terms of paths ending with different locations within a network.

The meaning of accessibility measures in terms of paths originating or coming to a location is that, for example a location with a high *accessibility-from-index* in a building is suitable for activities which demands rapid outward movements such as security, firing service, etc. while a location with high *accessibility-to-index* is suitable for inward static activities such as child care room, classroom, etc.

Penetration, a measure of relative accessibility, is depth (topological distance) of a location compared with the nearest deeper location [Tzonis87]. This measure is only suitable for systems having at least one point defined as the entry point.

$$\text{Penetration of } v_i = \frac{d(v_i, \text{ entry})}{d(v_i, \text{ entry}) + d(v_i, \text{ end})}$$

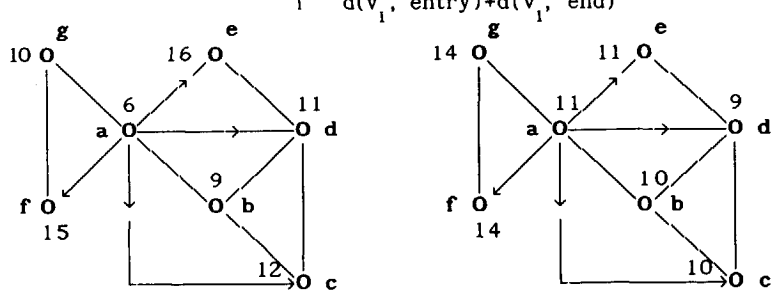


Figure A.9: Accessibility from measure and accessibility to measure
Example from: [Haggett69], page 46.

A.2.5 Balaban's distance connectivity index

This index, proposed by Balaban in 1982 [Bonchev83], is another topological index based on the distance matrix. In the following formula S_i and S_j are distance sums for vertices i and j to all other vertices in a graph G, and m is the number of edges in G.

$$J(G) = m \sum_{(i\ j)}^{m} (s_i * s_j)^{-1/2}, \quad \forall \text{ (I J) edges in G.}$$

A.2.6 Balaban average distance connectivity index

Another index proposed by Balaban [Balaban87] [Hanson87] is the average distance sum index. This index which is believed to have the most discriminating power [Trinajstic83b], is based on the distance sums s_i, the number of edges m, and the cyclomatic number f(G) of graphs.

$$J(G) = \frac{m}{f(G)+1} \sum_{(i\ j)}^{m} (s_i\ s_j)^{-1/2}, \ \forall \text{ edges (i j) in G.}$$

A.2.7 Centrality

Centrality, a vertex property of graphs, is a measure of closeness of a node to the center of the graph. One measure of the centrality is the *Konig number* [Kansky63]. Konig number of a particular vertex is calculated by counting the maximum number of edges in the shortest path by which that particular vertex is connected to any other vertex. This index, thus, monotonically increases with increase in distances of nodes from the center of a graph. Konig number is used in the analysis of the transport networks [Haggett69].

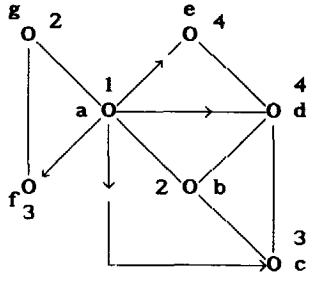


Figure A.7: A Partially symmetrical graph G, and its Konig number
[Haggett69] Page: 35.

A centrality measure developed for chemical application by Bavelas is based on the distance matrix D of a graph. The centrality index for a vertex v_i, denoted by $C(v_i)$, is obtained by dividing the total distance of the graph by the distance sum

s_i of v_i.

$$C(v_i) = (\sum S)/S_i$$

A global centrality index C(G), for a graph is defined as the sum of all point centralities.

$$C(G) = \sum_i C(v_i)$$

Vertex accessibility index is more powerful than the *Konig number* in a way that it can distinguish nodes with the same *Konig number*.

A.2.8 Eccentricity, radius, diameter, center and centroid of graphs

The eccentricity of a vertex $v \in V(G)$, denoted by e(v), is the distance farthest from v to any other vertex in G [Balaban87] [Buckley90].

$$e(v_i) = \text{Max } d_{ij}, \forall j$$

The *radius* r(G) is the minimum eccentricity of the nodes in G, whereas the *diameter* d(G) is the maximum eccentricity.

A vertex v_i is a *central* vertex if $e(v_i) = r(G)$.

The *center* of G, denoted by C(G), is the set of all central nodes (i.e., vertices with minimal eccentricity) in G [Bonchev83].

v is called a *peripheral node* if d(v)=e(v). The set of all peripheral nodes are called the *periphery* of G.

An *eccentric node* for $v \in G$ is any node at distance e(v) from v.

A *centroid* is a vertex with minimal weight or distance sums, where weight of a vertex v_i is the maximum number of lines in any branches of v_i [Balaban87] [Buckley90].

The following figure illustrate these indices. Notice that while the vertex degree decreases from the center of a graph toward its periphery, weights, eccentricities, and distance sums increase from the center towards the periphery of the graph. These information, obviously, have interesting implications for topological properties of buildings and architectural designs.

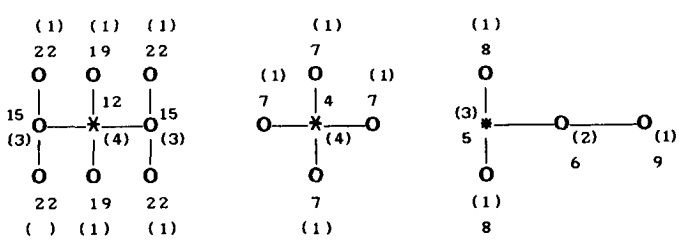


Figure A.10: The vertex degrees (in brackets), distance sums, and centroids (*) calculated for a number of graphs

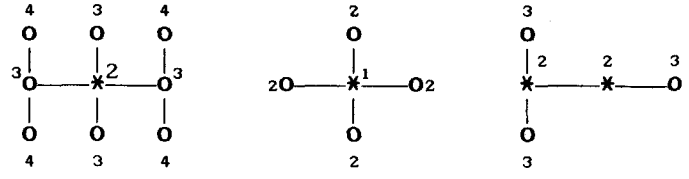


Figure A.11: Centers(*) of graphs based on centrality measures

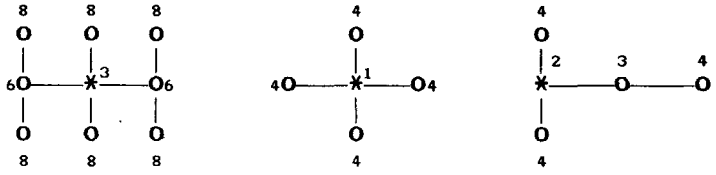


Figure A.12: Centroids (*) of graphs based on the vertices weights

A.2.9 Wiener's index, normalized Wiener's index, and Wieners polarity index
The index, introduced by H. Wiener in 1947 [Adler87], is a topological index characterizing the branchedness of a graph. The Wiener index is generally larger for graphs having a larger number of nodes, but it also provides a measure of branchiness of a graph. This index is equal to the sum of the distances between all pairs of nodes in a graph G, or half of all entries in the distance matrix of G [Balaban87] [Rouvray86].

$$W(G) = \frac{1}{2} \sum_{i} \sum_{j} d_{ij}$$

where, d_{ij} are the off-diagonal elements of the distance matrix D(G). This index is twice the distance sum index Si introduced earlier.

Wiener index may be normalized by dividing it by the multiplication of the number of distances (H) and the number of edges (m) in a graph. This index, to be called normalized Wiener index, is [Hanson87]: $\tilde{W}(G) = W(G) / (H * m)$

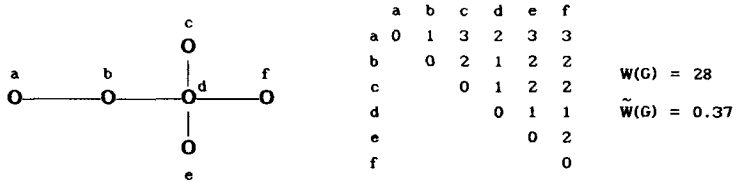


Figure A.13: A graph and its Wiener indices

Another index, based on the Wiener index, is Wiener's polarity index [Trinajstic-

83b], which is equal the number of pairs of vertices of distance 3. The polarity index can be generalized for paths of other lengths.

$$P(G) = 1/2 \sum_{i,j=1} \delta(d_{ij}, 3), \text{ where } \delta(a, b) = 1 \text{ if } a=b, \text{ and } 0 \text{ otherwise}$$

A.2.10 The Altenburg Polynomial index

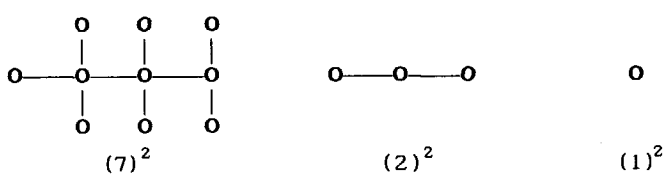
This index represented by the following polynomial [Kier76], is a characteristic measure of graphs in which n_i represents the number of pairs of nodes separated by d_i links.

$$y = \sum n_i d_i$$

This index for the graph of figure 4.10 is $y=5d_1+7d_2+3d_3=28$. Note that the result of this index is identical to wiener's index.

A.2.11 Balaban centric index

The centric index, named after Balaban [Bonchev83] [Rouvray83] [Trinajstic83b], emphasizes the branching degree of a node. This index, which is only available for trees, is calculated by squaring the number of vertices of degree one in a graph, pruning counted vertices, and continuing the previous step until no vertex is left. This index can be represented by the quadratic formula $B(G) = \sum a_i^2$, where a_i is the number of pruned vertices in step i. Balaban's index has a direct relation with the compactness and branchiness of a graph.



B(G)=49+4+1=54

Figure A.14: A graph and its Balaban's centric index

A.2.12 The Platt Index [Trinajstic83b]

This index introduced by Platt for predicting the physical properties of alkanes is based on the degrees of edges of a graph. The degree of an edge e, denoted by d(e), is the number of adjacent edges (i.e., immediate neighbors sum) to e. The Platt index is then defined as:

$$F(G) = \sum_{i}^{m} d(e_i) \quad \text{, where m is the number of edges in G.}$$

This index also have a direct relation with both the compactness and branchedness of a graph.

A.2.13 The Gordon-Scantlebury index

This index used in chemistry is defined as the total number of 2-walks in a graph, that is:

$$GS(G) = \sum d_{ij},\ \forall d_{ij} = 2$$

A.2.14 Topological index based on the vertices Weight [Balaban87]

Another TI, proposed by Balaban, is the weight index P based on the weights of the vertices on acyclic (tree) graphs. This index is similar to eccentricity formula.

$$P(G) = \sum^{m} (W_i * W_j)^{-1/2},\quad \forall \text{ m edges in a graph G.}$$

A.2.15 Some indices for comparisons of graphs

A Maximum Common Substructure MCS(G1, G2) for two graphs G1 and G2 is defined as measure of the maximum number of nodes and links (number of atoms and bounds-NAB- in the case of chemical molecules) in a sub-graph common to two graphs G1 and G2 such that no other sub-graph has a greater measure value [Johnson]. Figure 4.12 depicts MCS for two graphs G1 and G2.

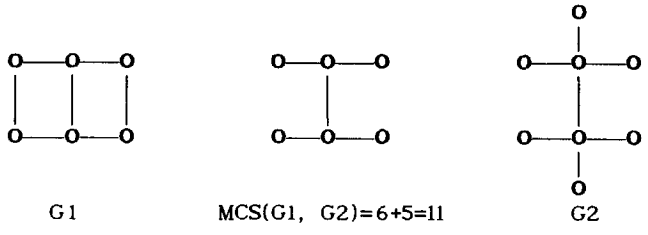


Figure A.15 : MCS for two graphs

Based on the MCS, Tsai [87] has proposed two indices for measuring the degree of similarity and dissimilarity between two graphs. The Maximum Similarity Index (MSI) for two graphs G1 and G2 is [Tsai87]:

$$MSI(G1, G2) = \frac{MCS(G1, G2)}{NAB(G1)} * \frac{MCS(G1, G2)}{NAB(G2)}$$

MDI, the (Molecular) Dissimilarity Index for two graphs is:

$$MDI(G1, G2) = 1.0 - MSI(G1, G2)$$

Another measure, also an indication of dissimilarity between two graphs is the Topological Distance index (TD) [Tsai87]. This index, not to be confused with the topological distances in a graph, is as follows:

$$TD(G1, G2) = NAB(G1) + NAB(G2) - 2*MCS(G1, G2)$$

Tsai [87] has applied MSI in quantitative analysis studies of structure-sweet

taste relationships of some chemical compounds, in the molecular similarity description of the compounds, and in molecular modeling.

Above indices are used [Johnson87] to arrive at a new index, called the *sub-graph metric* index, for comparison of two graphs:

$$d(G1, G2) = |G1| + |G2| - 2 * |MCS(G1, G2)|$$

where, $|G1|$ and $|G2|$ are the cardinalities of graphs G1 and G2, defined as the sum of the cardinality of their vertex and edge sets respectively. That is $|G1| = |V(G1)| + |E(G1)|$, and $|G2| = |V(G2)| + |E(G2)|$.

The sub-graph metric index can be extended to the graphs with labeled nodes and edges. Here the requirement is that the vertices and edges of a sub-graph agree in their type with the corresponding vertices and edges of sub-graph of another graph.

The indices introduced in this section are capable of matching the topological pattern of a building with other patterns (for example a set of patterns stored in a data base) for filtering and selection purposes.

A.2.16 Depth of a location

Depth of a location is its topological distance from an entry. In the case of a building with multiple entry points, depth for a location, depending on applications, could be considered as the distance of that location from the nearest, farthest, or even average distances of that location from the entry points. Depth is, in fact, a measure of relative accessibility of a location [Tzonis87].

A.2.17 Betweenness

The betweenness index for a node is the number of all pairs of nodes which paths between them passes through that node. The definition of this concept, depending on whether the effect of alternative paths (bypasses) is considered or not, may vary [Tzonis87].

A.2.18 Intervening locations

The intervening locations between two points v_i and v_j is the set of all locations on the paths between these two locations.

APPENDIX-B

EXAMPLES OF EXPERT RULES

This appendix presents examples of expert rules that relates the physical organization (topology), operation, and behavior of buildings with each other.

B.1 Rules relating design variables to norms

IF:
- Branching.

THEN:
- Privacy.
- Security.

IF:
- Serial arrangement of locations (Intervening locations).

THEN:
- Potential exposure.
- No privacy.
- Potential for intervening opportunities.

IF:
- Average Distance is short.

THEN:
- Low circulation cost.
- Low privacy cost.

IF:
- Average Distance is long.

THEN:
- High community.
- Potential for intervening opportunity.

B.2 Rules relating design variables to flow

- Location is a center location.

THEN:
- Distribution of flows.
- Collector of flows.

IF:
- Degree of branchiness = n.

THEN:
- Degree of control flow = n.

IF:
- Average Distance is short.

THEN:
- Divided flow among paths.
- Potential for regulating flow.

IF:
- Average Distance is long.

THEN:
- Flow is concentrated on one or few paths.
- Potential for high flow on a single path.

IF:
- No alternate paths.

THEN:
- All flow on the same path.
- All flow pass the same intervening locations.
- Create potential for flow monitoring at entry zone.

IF:
- Several alternate paths.

THEN:
- Divided flow.
- Increases indeterminacies of the flow.

IF:
- No intervening locations.

THEN:
- Flow between locations does not affect anything else.

IF:
- Low penetration for a location L.

THEN:
- All traffics (flow) from entry to location beyond passes through L.

IF:
- High penetration for a location.

THEN:
- Low flow.

IF:
- Betweenness.
- No bypass.
- One bracketing pair.

THEN:

- Flow determinate.
- Uniform flow.

IF:
- Betweenness.
- No bypass.
- Many bracketing pairs.

THEN:
- Flow is equal to the sum of flows between location-pairs.
- Possible variety of flow.

IF:
- Betweenness.
- Bypass.

THEN:
- Flow indeterminate.

IF:
- Balance near center.

THEN
- potential for high flow.

IF:
- Balance far from the center of gravity.

THEN:
- Potential for insulating from high flows.

B.3 Rules relating norms to design variables

IF:
- Many intervening locations.

THEN:
- Overlapping flows.
- Separation of pair.
- Affect flow rate.
- Potential for variety of experience.
- Potential for shared experience.
- Potential for high flow.

IF:
- Privacy.

THEN
- High degree of branching.
- Many dead-ends.
- Separate zones.

IF:
- Intervening opportunities.

THEN:
- Ave. distance long.

IF:
- Community for a location.

THEN:
- Intersecting flows.
- Overlapping flows.
- Simultaneous flows.
- No bypass.
- Maximum accessibility.
- Minimum distance from other points.
- No congestion around.
- Maximum flow.

IF
- Privacy.

THEN
- No through flow.
- Short path.
- Divide and separate flow.
- Location far from center of gravity.
- Many dead ends.
- Much branching.
- No dead end provide bypass.
- Low encounter.
- Minimum shared paths.
- Create hierarchies.

IF:
- Intervening opportunities.

THEN
- Share paths.
- Increase number of available locations.

IF:
- Intervening opportunities.

THEN
- Average distance long.

IF:
- Security.

THEN:
- Zoning.
- Channel all flow to pass a surveillance point.

IF:
- Safety.

THEN:
- Surveillance point.
- Easy out.
- Avoid paths where there is little flow.

B.4 Rules relating flow to norms

IF:
- No through flow.

THEN:

- Privacy.

IF:
- Through low.

THEN:
- Exposure.
- Potential for community.
- Potential for intervening opportunity.

B.5 Unclassified rules

IF:
- (Loop & n- locations & n is odd).

THEN:
- Exposure degree of each location = $2 \sum_{i=1}^{(n-1)/2} i$.

IF:
- (Loop & n- 1 locations AND n is even).

THEN:
- Exposure degree of each location = $1+2 \sum_{i=1}^{(n-2)/2} i$ + ((n - 2)/4).

IF:
- Degree of branchiness=n.

THEN:
- Degree of control flow=n.

IF:
- Stan type design with branchiness equal to n.

THEN:
- Security and
- n-degree of disturbance for center and privacy for branches.

IF:
- L1 connected to L2 and L2 dead-end.

THEN:
- L2 private with respect to L1.

IF:
- L1 connected to L2.

THEN:
- L1 and L2 have access.

IF:
- L2 inbetween L1 and L3.
- G1 moves from L1 to L3.

THEN:
- G2 has contact with G1.

IF:
- Prohibited-encounter(G_x, G_y, L_1).

THEN:

- Disfunction (G_x, L_1).

IF:
- Degree of branchiness equal to n.

THEN:
- Exposure degree equal to n.

IF:
- No of side-locations m and n.

THEN:
- Exposure degree = n(m+1) + m(n+1).

IF:
- Branching.

THEN:
- Zoning.
- Isolation from circulation.

IF:
- Location is a center location.

THEN:
- Information control.
- Surveillance.
- Social interaction.

IF:
- Branching.

THEN
- Zoning.
- Isolation from circulation.

IF:
- Branchiness of a location n.

THEN:
- Potential exposure degree=n.

IF:
- Branchiness is low.

THEN:
- Long paths.
- Few dead-end locations.
- No hierarchy of places.

IF:
- Branchiness is high.

THEN:
- Creates hierarchy of places (by penetration).
- Provide dead ends (location of 0 flow).
- Opportunities for convergence and divergence.
- Potential for separation.
- Potential for isolation.
- potential for controlling.
- Potential for mixing.

IF:
- Several alternate paths.

THEN:
- Provides bypass.
- Increases accessibilities.

IF:
- No intervening locations.

THEN:
- Flow between locations does not affect anything else.

APPENDIX-C

AUTOMATIC DISCOVERY: EXAMPLES

This appendix presents examples of small scale designs (connectivity patterns) with respect to social norms. These examples reflects generation of connectivity patterns of buildings with respect to the social norms based on graph enumeration techniques. Here I have shown the paper and pencils enumeration of graphs for discovery of designs of up to 6 or 7 locations. The approach may be automated, and results of the discovery can then be cataloged and stored in a data base to be used for generation of larger size designs.

In the examples that follows, a graph is tagged with G, designs (connectivity patterns) are tagged with an S, As represent the activities attached to each location, and CO, PR, CC, IO indicate the community, privacy, circulation-cost and intervening opportunity norms, respectively.

For n=1, we have only a null graph with a single node. This case is valueless in designs.

(G1) O

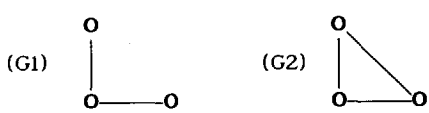
For n=2, we have a tree type graph with 2 nodes:

(G1) O——O

A single possibility for this case, obviously reduces the burden of choices regardless of the points of view on the design. A design choice for this case is as follows, where A1 are the activities for a design.

(S1) A1 O——O A2

For n=3 there are only 2 planar graphs (a tree and a cyclic graph).

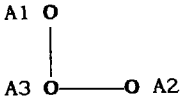


But once we try to label them, the number of possibilities increases, and the complexity of designs shows itself. The following paragraphs presents samples of different formulation of design problems with 3 locations, and their possible solutions that are intuitively found.

Problem:
- We have 3 activities A1, A2, and A3.
- The activity pair (A1 A2) has a flow potential of some degree.
- No intervening opportunity activity is defined.

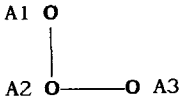
What is (are) the design(s) with respect to each social norms Co, Pr, CC, and a combination of them.

(S1)

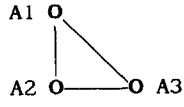


Sub-optimal solution for the community norm.

(S2)



(S3)



Sub_optimal solutions for

circulation_cost / privecy norm.

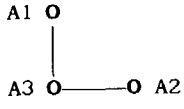
For this problem a globally optimal solution comprising all social norms does not exist. In this case one has a free choice of taking a sub-optimal solution at random. However, such a globally optimal design can be recognized if weights are assigned to the norms, in which case the solution associated with the higher weighted norm is the optimal design. For example, S1 is preferred over S2 if the community norm has a higher priority over other norms, otherwise S2 is preferable.

Problem:
- We have 3 activities A1, A2, and A3.
- Only the activity pair (A1 A2) has a flow potential of some degree.
- A3 is an intervening opportunity activity for groups responsible for activities A1 and A2.

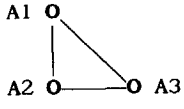
What are the optimal designs for different ranking order of social norms?

As in the previous case several design possibilities exist for above problem, best of which are as follows:

(S1)



(S2)

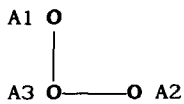


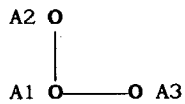
Ranking order	SOLUTION
CO > PR > IO	S1
CO > IO > PR	S1
IO > CO > PR	S1
IO > PR > CO	S2
PR > CO > IO	S2
PR > IO > CO	S2

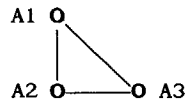
Problem:
- We have 3 activities A1, A2, and A3.
- The activity pair (A1 A2) has a flow potential of some degree.
- A1 is an intervening opportunity activity for groups responsible for activities

A2 and A3.
What are the optimal designs for different ranking order of social norms?

Best solutions and their ranking order:

(S1)


(S2)


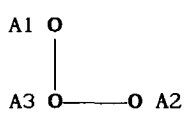
(S3)


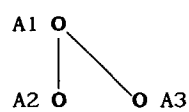
Ranking order	SOLUTION
CO > PR > IO	S1
CO > IO > PR	S1
IO > CO > PR	S3
IO > PR > CO	S2
PR > CO > IO	S3
PR > IO > CO	S2

Problem:
- We have 3 activities A1, A2, and A3.
- The circulation degree between activity pairs in decreasing order are as follow: C(A1 A2) > C(A1 A3)
- No intervening opportunity activity is defined.

What are the optimal designs for different ranking order of social norms?

Best solutions and their ranking order:

(S1)


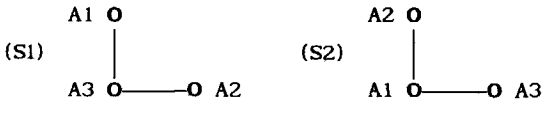
(S2)


Ranking order	SOLUTION
CO > PR > IO	S1
CO > IO > PR	S1
IO > CO > PR	S1
IO > PR > CO	S2
PR > CO > IO	S2
PR > IO > CO	S2

Problem:
- We have 3 activities A1, A2, and A3.
- The circulation degree between activity pairs in decreasing order are as follow: C(A1 A2) > C(A1 A3)
- A1 is an intervening opportunity activity for groups responsible for A2 and A3.

What are the optimal designs for different ranking order of social norms?

Best solutions and their ranking order:

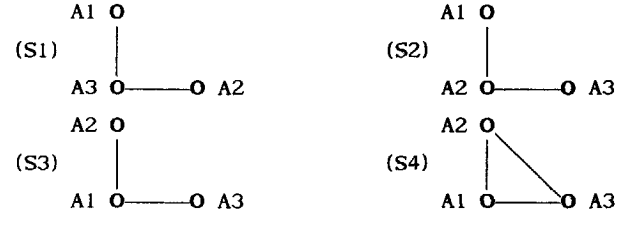


Ranking order	SOLUTION
CO > PR > IO	S1
CO > IO > PR	S1
IO > CO > PR	S2
IO > PR > CO	S2
PR > CO > IO	S2
PR > IO > CO	S2

Problem:
- We have 3 activities A1, A2, and A3.
- The circulation degree between activity pairs in decreasing order are as follow: C(A1 A2) > C(A1 A3)
- A2 is an intervening opportunity activity for groups responsible for A3 and A4.

What are the optimal designs for different ranking order of social norms?

Best solutions and their ranking order:

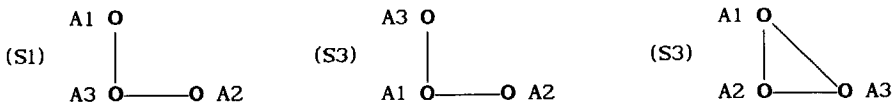


Ranking order	SOLUTION
CO > PR > IO	S1
CO > IO > PR	S1
IO > CO > PR	S2
IO > PR > CO	S2
PR > CO > IO	S3
PR > IO > CO	S4

Problem:
- We have 3 activities A1, A2, and A3.
- The circulation degrees between location pairs, in increasing order, are assumed to be as follow: C(A1 A2) > C(A2 A3) > etc.
- A1 is an IO activity for the groups responsible for activities A3 and A2.

What are the optimal designs for different ranking order of social norms?

Best solutions and their ranking order:

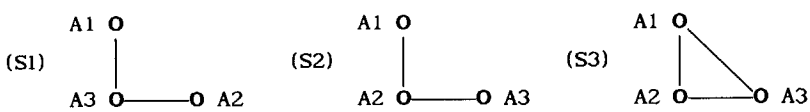


Ranking order	SOLUTION
CO > PR > IO	S1
CO > IO > PR	S1
IO > CO > PR	S2
IO > PR > CO	S3
PR > CO > IO	S2
PR > IO > CO	S3

Problem:
- We have 3 activities A1, A2, and A3.
- The circulation degrees between location pairs, in decreasing order, are assumed to be as follow: C(A1 A2) > C(A2 A3) > etc.
- A2 is an IO activity for A1 and A3.

What are the optimal designs for different ranking order of social norms?

Best solutions and their ranking order:

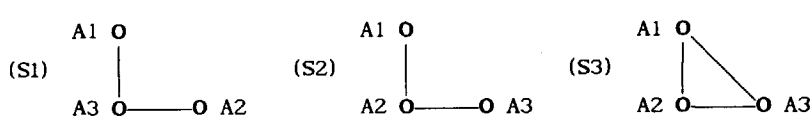


Ranking order	SOLUTION
CO > PR > IO	S1
CO > IO > PR	S2
IO > CO > PR	S2
IO > PR > CO	S3
PR > CO > IO	S3
PR > IO > CO	S2

Problem:
- We have 3 activities A1, A2, and A3.
- The circulation degrees between location pairs are the same.
- C(A1 A2) = C(A2 A3) = C(A1 A3).
- A2 is an IO activity for A1 and A3.

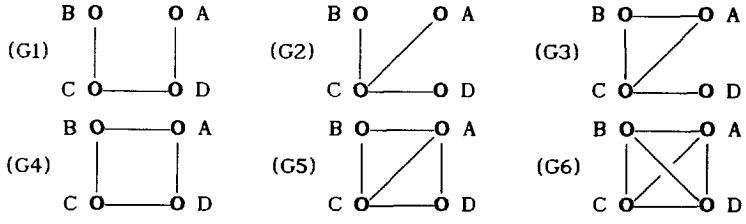
What are the optimal designs for different ranking order of social norms?

Best solutions and their ranking order:



Ranking order	SOLUTION
CO > PR > IO	S1, S2
CO > IO > PR	S1, S2
IO > CO > PR	S2
IO > PR > CO	S3
PR > CO > IO	S3
PR > IO > CO	S3

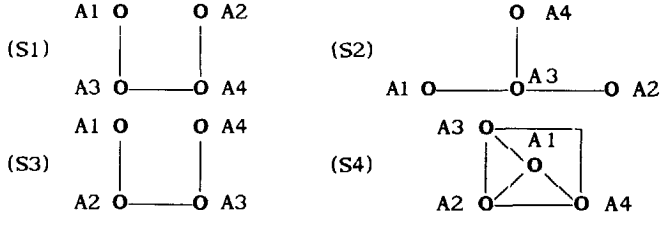
For n=4, we there are 6 planar graphs (2 trees and 4 general graphs)



Problem:
- We have 4 activities A1, A2, A3, and A4.
- The circulation degrees between location pairs, in decreasing order, are assumed to be as follow:
- C(A1 A2) > C(A2 A3) > C(A3 A4).
- A3 is an I.O. activity for A1 and A2.

What are the optimal designs for different ranking priorities of the social norms?

Best solutions and their ranking order:



Ranking order	SOLUTION
CO > PR > IO	S1
CO > IO > PR	S1
IO > CO > PR	S1
IO > PR > CO	S2
PR > CO > IO	S3
PR > IO > CO	S4

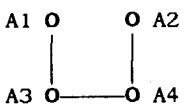
Problem:
- We have 4 activities A1, A2, A3, and A4.
- The circulation value between location pairs, in decreasing order, are assumed

to be in the following manner:
- (A1 A2) > (A2 A3) > (A3 A4).
- A4 is an I.O. activity for A1 and A2.

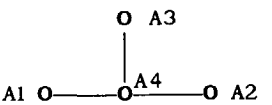
What are the optimal designs for different ranking priorities of the social norms?

Best solutions and their ranking order:

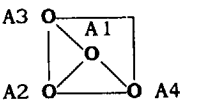
(S1)



(S2)



(S4)



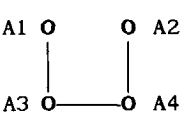
Ranking order	SOLUTION
CO > PR > IO	S1
CO > IO > PR	S1
IO > CO > PR	S1
IO > PR > CO	S2
PR > CO > IO	S3
PR > IO > CO	S3

Problem:
- We have 4 activities A1, A2, A3, and A4.
- The circulation degrees between location pairs, in decreasing order, are assumed to be as follow: (A1 A2) > (A2 A3) > (A3 A4).
- A2 is an IO activity for A1 and A3.

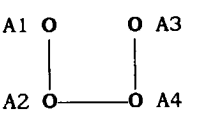
What are the optimal designs for different ranking priorities of the social norms?

Best solutions and their ranking order:

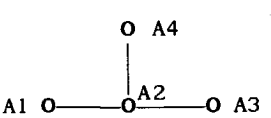
(S1)



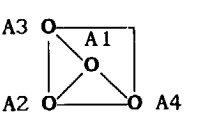
(S2)



(S3)



(S4)

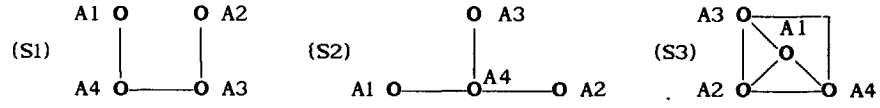


Ranking order	SOLUTION
CO > PR > IO	S1
CO > IO > PR	S1
IO > CO > PR	S2
IO > PR > CO	S3
PR > CO > IO	S3
PR > IO > CO	S4

Problem:
- We have 4 activities A1, A2, A3, and A4.
- The circulation degrees between location pairs, in decreasing order, are assumed to be as follow: (A1 A2) > (A2 A3) > (A3 A4).
- A4 is an I.O. activity for A1 and A3.

What are the optimal designs for different ranking priorities of the social norms?

Best solutions and their ranking order:



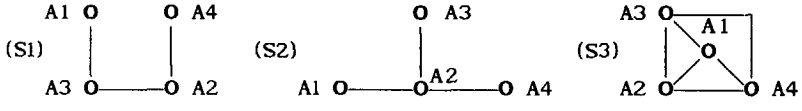
Ranking order	SOLUTION
CO > PR > IO	S1
CO > IO > PR	S1
IO > CO > PR	S1
IO > PR > CO	S2
PR > CO > IO	S3
PR > IO > CO	S3

Problem:

- We have 4 activities A1, A2, A3, and A4.
- The circulation degrees between location pairs, in decreasing order, are assumed to be as follow: (A1 A2) > (A2 A3) > (A3 A4).
- A2 is an IO activity for A1 and A4.

What are the optimal designs for different ranking priorities of the social norms?

Best solutions and their ranking order:



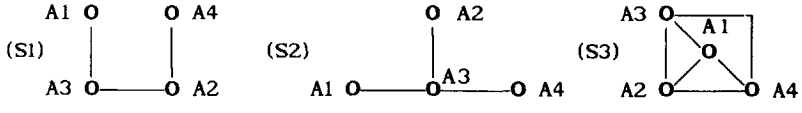
Ranking order	SOLUTION
CO > PR > IO	S1
CO > IO > PR	S1
IO > CO > PR	S1
IO > PR > CO	S2
PR > CO > IO	S3
PR > IO > CO	S3

Problem:

- We have 4 activities A1, A2, A3, and A4.
- The circulation degrees between location pairs, in decreasing order, are assumed to be as follow: (A1 A2) > (A2 A3) > (A3 A4).
- A3 is an I.O. activity for A1 and A4.

What are the optimal designs for different ranking priorities of the social norms?

Best solutions and their ranking order:



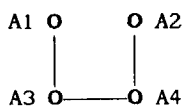
Ranking order	SOLUTION
CO > PR > IO	S2
CO > IO > PR	S1
IO > CO > PR	S1
IO > PR > CO	S2
PR > CO > IO	S3
PR > IO > CO	S3

Problem:
- We have 4 activities A1, A2, A3, and A4.
- The circulation degrees between location pairs, in decreasing order, are assumed to be as follow: (A1 A2) > (A2 A3) > (A3 A4).
- A3 is an IO activity for A1 and A4, and A4 is an IO activity for A3 and A2.

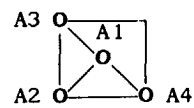
What are the optimal designs for different ranking priorities of the social norms?

Best solutions and their ranking order:

(S1)



(S3)



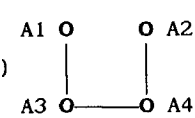
Ranking order	SOLUTION
CO > PR > IO	S1
CO > IO > PR	S1
IO > CO > PR	S2
IO > PR > CO	S2
PR > CO > IO	S1
PR > IO > CO	S2

Problem:
- We have 4 activities A1, A2, A3, and A4.
- The circulation degrees between location pairs, in decreasing order, are assumed to be as follow: (A1 A2) > (A2 A3) > (A3 A4).
- A4 is an I.O. activity for A1 and A3, and A3 is an IO activity for A4 and A2.

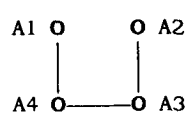
What are the optimal designs for different ranking priorities of the social norms?

Best solutions and their ranking order:

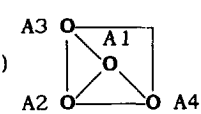
(S1)



(S2)



(S3)

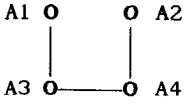


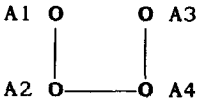
Ranking order	SOLUTION
CO > PR > IO	S1
CO > IO > PR	S1
IO > CO > PR	S3
IO > PR > CO	S3
PR > CO > IO	S2
PR > IO > CO	S3

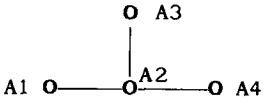
Problem:
- We have 4 activities A1, A2, A3, and A4.
- The circulation degrees between location pairs, in decreasing order, are assumed to be as follow: (A1 A2) > (A2 A3) > (A3 A4).
- A2 is an IO activity for A1 and A4, and A4 is an IO activity for A2 and A3.

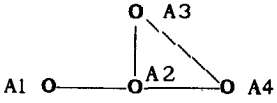
What are the optimal designs for different ranking priorities of the social norms?

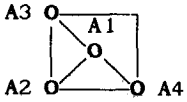
Best solutions and their ranking order:

(S1)


(S2)


(S2)


(S2)


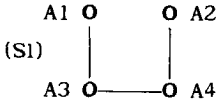
(S3)


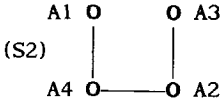
Ranking order	SOLUTION
CO > PR > IO	S1
CO > IO > PR	S2
IO > CO > PR	S1
IO > PR > CO	S2
PR > CO > IO	S3
PR > IO > CO	S4

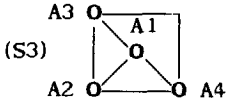
Problem:
- We have 4 activities A1, A2, A3, and A4.
- The circulation degrees between location pairs, in decreasing order, are assumed to be as follow: (A1 A2) > (A2 A3) > (A3 A4).
- A4 is an I.O. activity for A1 and A2, and A2 is an I.O. activity for A4 and A3.

What are the optimal designs for different ranking priorities of the social norms?

Best solutions and their ranking order:

(S1)


(S2)


(S3)


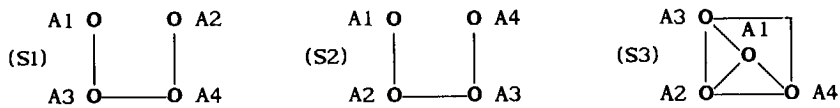
Ranking order	SOLUTION
CO > PR > IO	S1
CO > IO > PR	S2
IO > CO > PR	S1
IO > PR > CO	S2
PR > CO > IO	S3
PR > IO > CO	S4

Problem:
- We have 4 activities A1, A2, A3, and A4.
- The circulation degrees between location pairs, in decreasing order, are assumed

to be as follow: (A1 A2) > (A2 A3) > (A3 A4).
- A2 is an I.O. activity for A1 and A3, and A3 is an I.O. activity for A2 and A4.

What are the optimal designs for different ranking priorities of the social norms?

Best solutions and their ranking order:

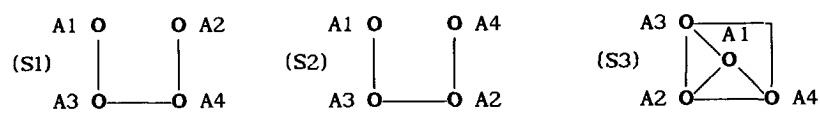


Ranking order	SOLUTION
CO > PR > IO	S1
CO > IO > PR	S2
IO > CO > PR	S1
IO > PR > CO	S2
PR > CO > IO	S3
PR > IO > CO	S4

Problem:
- We have 4 activities A1, A2, A3, and A4.
- The circulation degrees between location pairs, in decreasing order, are assumed to be as follow: (A1 A2) > (A2 A3) > (A3 A4).
- A3 is an IO activity for A1 and A2, and A2 is an IO activity for A3 and A4.

What are the optimal designs for different ranking priorities of the social norms?

Best solutions and their ranking order:



Ranking order	SOLUTION
CO > PR > IO	S1
CO > IO > PR	S2
IO > CO > PR	S1
IO > PR > CO	S2
PR > CO > IO	S3
PR > IO > CO	S4

Problem of larger size is left for the discoverer system. Designs problems with a size larger than 4, are in any case beyond the handling capacity of individuals. The only way to get around the complexity of their enumeration, testing, and selections of appropriate designs are by means of an automatic system.

SAMENVATTING

Dit proefschrift behandelt architectureel ontwerpen (van gebouwen) in drie etappen: analytisch, topologisch en meetkundig. In de analytische fase wordt de opdracht geformuleerd en het programma van eisen opgesteld. De volgende twee fasen betreffen het eigenlijke ontwerp. Beide zijn, rekentechnisch gezien, onhandelbaar qua complexiteit. Mensen echter, kunnen gebouwen wel ontwerpen. Dit proefschrift automatiseert kennis die architecten gebruiken om vloerontwerpen van gebouwen (in de topologische ontwerpfase) te maken. Dit gaat vooraf aan de maatvoering (meetkundige fase). De kennis in de topologische fase is primair kennis die de functie van een gebouw uitdrukt in ruimtelijke verbanden (bijvoorbeeld: plaats secretaressekamer altijd met toegang naar werkkamer chef of gang). Deze kennis wordt in dit proefschrift verdeelt in vier "sociale" kennisregels: privacy (van ruimtes), de gemeenschappelijkheid (van ruimtes), de ontmoetingskans (van mensen in ruimtes) en de loopafstand (tussen ruimtes). In dit proefschrift wordt het ontwerpen (op topologisch nivo) van een gebouw geautomatiseerd m.b.v. de genoemde kennisregels, ook wordt de beoordeling van bestaande ontwerpen m.b.v. dezelfde kennisregels uitgetest.

Omdat ontwerpen een combinatorisch complex karakter heeft, is het voor grote aantallen ruimtes rekentechnisch onhaalbaar. Het belangrijkste deel van dit proefschrift is de TOPologische GENErator. Dit is software die kennisregels kan hanteren om (incrementeel) vloerindeling te ontwerpen en te beoordelen. De rekentechnische problemen zijn gekraakt m.b.v. een heuristische zoekmethode "hill-climbing", door de ruimte van mogelijke ontwerpen.

Omdat de expertise in de kennisregels (logisch) ook nog afhankelijk en zelfs strijdig is, neemt de complexiteit van de zoekruimte juist door het

gebruik van deze ontwerpkennis toe. De mate van interactie tussen ruimtes volgens die kennisregels is behandeld m.b.v. de zgn. "Q-analysis" om tijdens het ontwerp- redeneerproces zoveel mogelijk clusters (van ruimtes) te kunnen behandelen, ter reductie van de zoekruimte.

Als alternatief voor de expliciete kennisaanpak is een Neuraal Netwerk (NN) volgens Hopfield's ideëen geprogrammeerd. Deze impliciete kennisaanpak is met succes toegepast op dezelfde ontwerpproblemen zoals behandeld door het op expliciete kennisregels gebaseerde TOPGENE. Verbazingwekkend is dat de NN aanpak soms zelfs beter scoort dan de kennisaanpak.

CURRICULUM VITAE

Abolfazl Zandi-nia

Born in Kashan, Iran 1_07_1952

Education:

"Bachelor of Science" degree in computer science 1975
University of District of Columbia,
Washington, D.C., U.S.A.

"Master of Science" degree in computer science 1977
George Washington University
Washington, D.C., U.S.A.

"Engineer" degree in computer science 1979
George Washington University
Washington, D.C., U.S.A.

Experiences:

Lecturing in computer science 1980-82
A.T. University, Theran, Iran.

Head of planning and information processing, 1982-85
Electronic equipment procurement and distribution
center (E.E.P.D.C.).
Ministry of Commerce, Tehran, Iran.

Managing director 1984-85
DP Iran (Ex. IBM branch, Iran)
Tehran, Iran

Member of the board of directors. 1985-86
Iran Argham co. (Ex. NCR branch, Iran)
Tehran, Iran

Member of High Council of Informatics 1985-86
Ministry of Budgeting and Planning
Tehran, Iran

Research work:

TOPGENE: An Artificial Intelligence Approach 1987-1992
to a design Process
Department of Informatics & Mathematics,
Delft University of Technology,
Delft, The Netherlands.

Paper in print:

Zandi-nia, A., and H. Koppelaar,
Solving a class of architectural design problems by a Neural Network,
International Journal of Intelligent Systems,
John Wiley & Sons, Inc. Publishers,