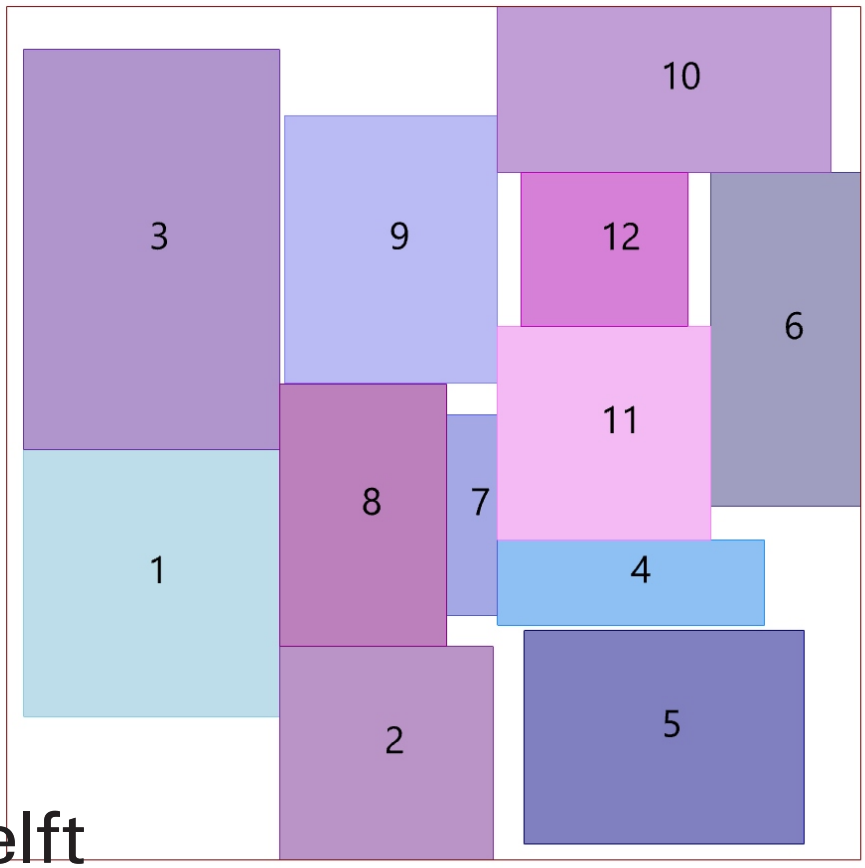
David and the vegetable factory

A Two-stage Optimization Model for Generating a Vegetable Factory's Facility Layout Using the Gradient Descent Approach

D. den Ouden

Master thesis

ISBN 000-00-0000-000-0



TU Delft

David and the vegetable factory

A Two-stage Optimization Model for Generating a Vegetable Factory's Facility Layout Using the Gradient Descent Approach

by

D. den Ouden

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Monday July 6, 2020 at 10:30 AM.

Student number:	4469712	
Project duration:	November 11, 2019 – July 6, 2020	
Thesis committee:	Dr. Ir. P. Nourian,	TU Delft, main mentor
	Dr. Ir. P. W. Heijnen,	TU Delft, second mentor
	G. Coumans,	TU Delft, board of examiners committee

TU Delft

Preface

After three years of bachelors and a year and 3 months of my masters, the time was finally there to choose a masters thesis, the final piece of ”official” education. I’m saying official because one never really does stop learning, be it in school when tackling another hard mathematics subject or when working and accepting a new challenge that is slightly out of your comfort zone: we never stop learning. That being said, the choice for a thesis seems like it should not matter too much, if one does not stop learning anyway, there should always be time to learn essential things later. However, the master thesis is something that reflects your skills, knowledge and passion, and is essentially a gateway into the ”real world”, a curriculum vitae, if you will.

My masters thesis should therefor reflect what I want to be doing for, at least, part of my life. This was no easy task however, I started my bachelors with the wish to become an architect, changed my mind 6 months in in favor of becoming a construction engineer and left with not much of a clue except the notion that I wanted something to do with architecture but more technical. Building technology sounded like just that to me: part architecture but more technical. However, at the end of my first year, I still had no clear idea of what to do with my thesis, other than that I had discovered that the design informatics chair appealed to me the most.

It was just before the summer vacation of 2019 that I emailed assistant professor Nourian with the question whether it was possible for me to join the course earthy even though the registrations had already reached the maximum number within hours. I would not have been writing this text today if assistant professor Nourian hadn’t gone out of this way and made sure that earthy would allow for 5 more students, including me. It was during earthy that I claimed my role in the group as ”programmer”, since the first few workshops we had had about the subject appealed to me a lot.

The way I liked programming in python was the final nudge I needed to convince myself to do a thesis in design informatics. However, it wasn’t until assistant professor Nourian convinced me that generating a factory layout had very little to do with parametric design and grasshopper coding that I decided to take a leap of faith. Without him and my second mentor assistant professor Heijnen I would not have been able to make this thesis into what it is today. For that, I want to express my sincerest gratitude.

Preface…

den Ouden
Delft, June 2020

Abstract

In this thesis a two-stage model is proposed, combining the idea of a two-stage model from Anjos and Vieira [1] with a gradient descent approach, much like proposed in Sikaroudi and Shahanaghi [42], in order to solve the facility layout problem for problems consisting of 8 and 12 departments . This method was chosen after analyzing the criteria for a vegetable processing factory and combining this with a literature research to previous methods applied to solve these ”facility layout problems”. The gradient descent approach uses the partial derivatives of a multi-variable objective, the flow-cost, in order to get a vector which is the direction of greatest descent. Computing this vector for all departments and then moving them creates an iterative improving loop. In addition to the gradient descent approach a swapping procedure and a shooting procedure is introduced in order to reduce the effect of random starting position. From the result of the first-stage model, relative location constraints are extracted to be used in the second-stage model, a linear constraint programming solver capable of running optimizations by google or-tools [36], to greatly reduce the solution space. The proposed method was tested on two toy-problems from Tam and Li [44] and compared to results existing in literature. Additionally, the effect of including a first-stage model was tested as well. When looking directly at the results, the proposed method is consistent and shows good results. Especially the inclusion of the first-stage model helps finding better solutions. Compared to other results found in the literature, the method shows slightly less good results based purely on the objective value. However, the final layouts found in this thesis are more compact.

GLossary of commonly used symbols and acronyms

A_i	Area requirement for department i (m2)
a_{ikl}	Total area assigned to department i (m2)
AUF	Area utilization factor
c_{ij}	Cost for moving materials from departments i to j (Euros/ Unit flow * m)
d_{ij}	Distance between departments i an j (m)
f_{ij}	Flow between departments i an j (units cargo/ unit time)
FLP	Facility layout problem
GH	Grasshopper
H_F	Height of the total facility (m)
h_i	Height of department i (m)
h_i^{max}	Max height for department i (m)
h_i^{min}	Min height for department i (m)
MPWG	Maximal planar weighted graph
P_i	Perimeter of department i (m)
r_i	Radius of department i (m)
SRF	Shape ratio factor
TBA	Total blanc area
W_F	Width of the total facility (m)
w_i	Width of department i (m)
w_{ij}	Closeness rating between departments i an j (-)
w_i^{max}	Max width for department i (m)
w_i^{min}	Min width for department i (m)
x_i	X coordinate of department i (-)
y_i	Y coordinate of department i (-)
α_i	Aspect ratio of department i (-)
λ	Decision variable (-)

Contents

1 Introduction **1**
1.1 Motivation . . . 1
1.2 Context . . . 1
1.3 The facility layout problem . . . 2
1.3.1 Equal vs Unequal . . . 3
1.3.2 Dynamic vs Static . . . 3
1.4 Problem statement . . . 3
1.5 Research objective . . . 4
1.5.1 research question . . . 4

2 Methodology **7**
2.1 Research methodology . . . 9
2.2 Assessment . . . 9
2.2.1 Verification . . . 9
2.2.2 Validation . . . 9

3 Literature study **11**
3.1 Introduction . . . 11
3.2 Layout formulations . . . 11
3.2.1 Quadratic layout formulation . . . 11
3.2.2 Mixed integer programming . . . 13
3.2.3 Graph theoretic . . . 14
3.3 Resolution approaches . . . 17
3.3.1 Exact methods . . . 17
3.3.2 Meta-heuristic methods . . . 17

4 Requirements and criteria **21**
4.1 Process . . . 21
4.2 Departments . . . 21
4.3 Flows . . . 22
4.4 Transportation . . . 23
4.5 Conclusion . . . 23

5 Methods evaluation **27**
5.1 Evaluation . . . 27
5.2 Conclusion . . . 31

6 Proposed methodology **33**
6.1 Problem formulation . . . 33
6.1.1 Shaping the departments . . . 33
6.1.2 Overlap . . . 34
6.1.3 Objective . . . 34
6.1.4 Relative location constraints . . . 35
6.2 Proposed methodology . . . 35
6.2.1 First-stage model . . . 35
6.2.2 Second-stage model . . . 36
6.2.3 Assessment . . . 36

7 Generation of the layout using python **37**
7.1 first-stage model . . . 37
7.1.1 Pseudo-code . . . 37
7.1.2 Input . . . 38
7.1.3 Generating the departments . . . 39
7.1.4 Adding vectors . . . 40
7.1.5 Moving the departments . . . 42
7.1.6 Computing the objective . . . 44
7.1.7 Assessment . . . 45
7.1.8 Improvement . . . 46
7.2 second-stage model . . . 49
7.2.1 Extracting relative location constraints . . . 49
7.2.2 Implementing the constraints . . . 50

8 Results **53**
8.1 First-stage model . . . 53
8.1.1 Computational time . . . 53
8.2 Second-stage model . . . 58

9 Conclusion **61**

Bibliography **67**

A Appendix A: Temperature differences **71**

B Appendix B: Previous solutions **75**
B.1 A solution using simulated annealing . . . 75
B.2 A simulation based optimization technique . . . 76
B.3 A two-stage optimization-based framework . . . 77
B.4 A solution by genetic algorithm and space filling curves . . . 79
B.5 A knowledge-based approach for converting a dual graph into a block layout . . . 80
B.6 A graph theoretic heuristic . . . 81
B.7 An improved tabu search heuristic . . . 82
B.8 Layout by collision detection and force exertion . . . 83
B.9 A slicing tree based heuristic . . . 85

C Appendix C: a sample of results for the vanilla first-stage model **87**

D Appendix D: a sample of results for the swapping first-stage model **93**

E Appendix E: a sample of results for the shooting first-stage model **99**

F Appendix F: a sample of results for the swapping and shooting first-stage model **105**

G Appendix G: the results for all time-frames for the 8 department second-stage model **111**

H Appendix H: the results for all time-frames for the 12 department second-stage model **115**

I Appendix I: first-stage model main python component **119**

J Appendix J: second-stage model main python component **127**

K Appendix K: A vegetable processing factory's requirements **133**

1 Introduction

In a world that is getting more complicated, so are the floor-plans of numerous buildings. Especially for industries, a layout of the floor-plan can really define how profitable a factory is. It is estimated that in total 20-50% of total operating expenses can be related to the material handling costs and the layout of a factory [45]. Furthermore, early changes in the design of a layout have a big impact on the final design and can be very profitable, especially considering that changes in a later stage are significantly harder, costlier and more complicated to perform. Chwif et al. [8] suggest that the optimal location of facilities is one of the most important issues that should be resolved early in the design stage. In the past, when layouts were less complicated, optimizing the location of facilities was not as much of an issue. However, after the second world war, operations research and with it the optimizing of processes arose. Optimizing various processes has been a big issue ever since, and so too the optimization of the layout of facilities emerged which became known as the Facility Layout Problem. With the development of powerful computers that are available to the wide public, a window of opportunities opened to be able to solve these ”FLP’s”.

1.1. Motivation

The motivation of this thesis comes partly from personal experience of designing factory layouts, where my work at the company Dapp Project-management in Food has taken me through the world of designing food factories. In this world, the use of computation to design is hardly used, traditional hand-drawings are the main source of designing and communicating plans. This comes with problems like long wait times to change certain details or entire plans. This let me to believe that a computational implementation of designing the layout could give a huge benefit to the industry. Besides potentially saving time and money, a faster and more adaptable ”computational tool”, if you will, would also ultimately lead to a more sustainable workflow, since time and money and thus emissions could be spent more efficient. A sustainable way of living is something that is absolutely necessary at the present date, in order to express its importance, a quote by the late Professor Stephen Hawking will be presented:

> ”Global warming is caused by all of us. We want cars, travel and a better way of living. The trouble is, by the time people realise what is happening, it may be too late. ... both effects (deforestation and melting of the ice caps) could make our climate like that of Venus: boiling hot and raining sulphuric acid, but with a temperature of 250 degrees Celsius. Human life would be unsustainable.” [18]p148-149

1.2. Context

The context of this thesis takes root in a number of fields, where three different fields can be marked as major. These three fields are: operations research, mathematics and the food industry. The common binder of these fields is the facility layout problem, which will be elaborated upon in great detail further on in this thesis. Other fields that fall under the three major fields named are all shown in the Venn diagram in the figure below. It is known that there are other fields which are closely related, including

but not limited to: game theory, operations management and supply chain management. However, they fall out of the scope. Hence, they will not be dealt with in this thesis.

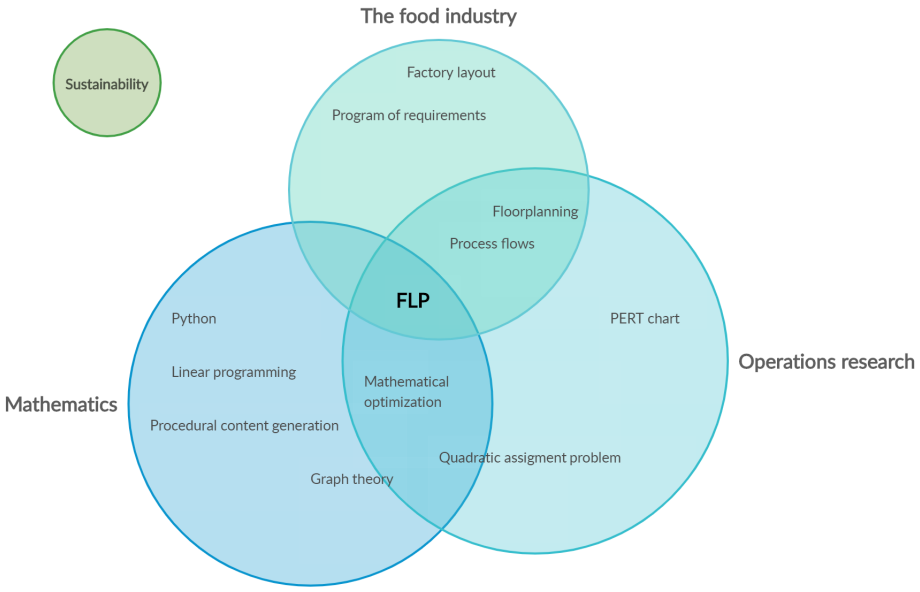


Figure 1.1: An Euler diagram showing the scope and context of this thesis.

1.3. The facility layout problem

Like said in the introduction, the facility layout problem is a problem that deals with allocating facilities to locations, but what exactly is the facility layout problem? The facility layout problem was first addressed in 1957 by Koopmans and Beckmann [26] who described the facility layout problem as a common industrial problem in which the objective is to configure facilities, in order to be able to minimize the cost of transporting materials between them. In order to get a better grip on what exactly is the facility layout problem, Heragu [19] has defined the parts that make up the problem, they define a facility layout as ”an arrangement of everything needed for production of goods or delivery of services”. Furthermore, a facility is ”an entity that facilitates the undergoing of a job, it could be a machine tool, a work center, a manufacturing cell or even a whole shop, department or warehouse”. Because of the variety of considerations found in all articles about facility layout problems, there is not one common and exact definition [11]. Drira et al. [11] gives as a reason that all layout problems are strongly dependent on specific features of the manufacturing systems in question.

A few examples of definitions of the facility layout problem in the literature are as follows. Meller et al. [31] describe the facility layout problem as the search for an arrangement of n non-overlapping planar orthogonal rectangular facilities within a given site that is also rectangular, in order to minimize the distance between them. From this definition it is important to notice that facilities should be non-overlapping. Noticeable is that all elements in the definition leave no room for curved or otherwise non-rectangular elements. Azadivar and Wang [4] seem to leave more room for elements of any size by stating that the facility layout problem is defined as the determination and allocation of given space amongst a number of facilities. This definition however, seems to be satisfied with the problem as long as all facilities are allocated in any order, no objective is specified. Another definition is that the FLP is the determination of the relative locations and allocation of an n number of unequal sized departments within a total given space in a way to minimize the total material handling cost and slack area cost [29]. In that definition, an objective is clearly stated, it also introduces the fact that the departments do not necessarily have to be of the same area and size.

Yet another take on how to look at the facility layout problem is given by Shayan and Chittilappilly [41], they define the facility layout problem as an optimization problem that tries to make layouts optimal by looking at various interactions that happen within the facilities and the material handling system.

According to this definition, a default layout must be present already so it can be improved upon. Interesting is that, in a paper by Merrell et al. [32], where layouts for architectural residential houses are being generated, the facility layout problem is called the spacial allocation problem, and is defined as the layout of architectural spaces on a plane. This goes to show that the facility layout problem is very closely related to architecture. One last recent definition given by Pourhassan and Raissi [37] states that the facility layout problem is the crucial task to allocate the machines/equipment in a manufacturing shop floor. This is effectively done when the layout ensures that there is a smooth flow of not only material, but also equipment and manpower while this is kept at a minimum.

Pourhassan and Raissi [37] labels the facility layout problem crucial because it not only denotes to 20-50% of the total operating cost but also to 20-70% of the total manufacturing costs, as stated in [45]. Also Drira et al. [11] states the importance of optimizing the facility layout problem, as they state that it can have an significant impact on manufacturing costs, work in progress, lead times and productivity. An efficient facility layout greatly improves the competitive edge a manufacturing company has by reducing the material handling costs and utilising the space more efficiently [48]. The facility layout problem can be categorised into either being equal or unequal and dynamic or static [11]

1.3.1. Equal vs Unequal

Facility layout problems can be classified into equal-sized and unequal sized facility layout problems which is defined by the sizes of the departments. With equal sized facility layout problems, all the departments are equal sized, meaning that one department can always swap positions with another department without having to be adjacent. This means that the two decisions of shaping and placing don't have to be done simultaneously but can be done sequentially [25]. On the other hand when dealing with unequal departments, this is no longer possible since swapping any two departments requires their area to be equal unless they are adjacent. This means that the shaping and placing of the departments have to be done simultaneously. In general the unequal facilities problems have a higher possibility to be applicable in a practical manner, however, in the past they have been addressed less in the literature in comparison to their equal sized counterpart [25].

1.3.2. Dynamic vs Static

The facility layout problem can be categorized in either an dynamic or a static problem, although static problems are way more common. Static approaches consider the material flows between different departments to be constant at all times and therefore the optimal layout is also designed with these material flow in mind. On the other hand, if the layout is designed based on different material flows during different time periods, the problem can be categorized as dynamic and the facility layout should also be designed to handle these different material flows optimally [37]. The material flow can become fluctuated due to variations in demand and changes in product mix. While in the literature the most dealt with problems are static, Page [35] states that, on average, 40% of a company's revenue come from new products. This means that companies must innovate and change layout frequently. Also, according to a study, 1/3rd of every company in the United States undergo a major reorganization concerning the production facilities every two years [16]. Dynamic layout problems therefor offer a solution, because they take different material flows into account. In the case of this thesis, the flows that are used between departments are representative for the food processing factory (because the flows are a simplified version of the real-life problem), and even-though that does not mean that the flows will remain exactly like this at any given time, it does make the problem static in the case of this thesis.

1.4. Problem statement

Even though there is great value to be added by computational design to improve the design performance, it is hardly ever used in the early design process [51]. When these computational design methods are in fact used, they tend to be only be used to verify the conceptual design, instead of being used to explore the different solutions which could be generated [13]. In the field of the food industry, computational design is not a known concept, instead, most of the work is done by hand. Introducing the power of a 21st century computer into the design process could lead to great benefits. However, this computational design should be presented in such a way that is comprehensible by its users, which are the people that work on designing the layouts of the factories. These people have minor experience

with computational design and thus a tool that copes with this lack of computational knowledge should be developed. The problem lies in how to develop such a tool.

As mentioned in section 1.3, the facility layout problem boils down to arranging facilities in such a way that the material handling cost are minimized. This problem, however, quickly becomes NP-complete when increasing the number of facilities. Johnson [22] has showed that when the number of facilities becomes larger than 15, the facility layout problem becomes a NP-complete problem. This means that with increasing number of facilities the computational time required to find the absolute best solution is increased by 2^n. Since the optimal solution for a larger number of facilities becomes virtually impossible to reach, (meta)heuristic methods have been developed in order to find a near optimal solution in a reasonable time span [47]. The problem of NP-completeness can thus be resolved by applying (meta)heuristic approaches in order to achieve near optimal solutions.

The formulation of the problem in mathematical terms is the generation of a layout for a food processing factory, while trying to optimize the material handling cost as seen in equation(1.1)[47] and keeping the problem compact (e.g. reduce the amount of empty space), where c_{ij} is the transportation cost for transporting between departments i and j, f_{ij} is the flow between those departments and d_{ij} is the distance between them. The problem will be tackled as an unequal-sized problem in order to maintain accuracy and as a static problem in order to simplify the problem.

$$Minimize F = \sum_{i=1}^{n} \sum_{j=1}^{n} C_{ij} f_{ij} d_{ij} \tag{1.1}$$

1.5. Research objective

Considering the problem statement the research objective then becomes to develop a tool for designers of factory layouts to use to generate the layout of a factory. To be more precise, this thesis will take the specific case of a vegetable processing factory. This tool should be user friendly, even to a user with little computational experience. To achieve this, the facility layout problem must be formulated and then solved using meta-heuristic methods since exact methods would impose a computational solving time that is just to vast.

1.5.1. research question

The posed problem statement and research objective lead to the following main research question:

> "How to computationally generate a layout of a vegetable processing factory given a program of requirements and flows between facilities as a matrix using a mathematical approach, minimizing the travel distance of goods needed for a product to be manufactured?"

This main research question consequently leads to the following sub-questions concerning how to answer the main research question but broken down in steps. They are as follows:

(1) "How can the facility layout problem be formulated?"

(2) "What computational approaches have been proposed to solve the FLP in the literature?"

(3) "What are the requirements of a vegetable food processing factory and how can these requirements be formulated mathematically?"

(4) "To what extent do the proposed methodologies in the literature comply with this specific case of the FLP?"

(5) "What are the main objectives and constraints to satisfy considering the generation of a layout for a vegetable processing factory with a given program of requirements and flows between facilities?"

Chapter 2 will go over the methodology used in this paper and discuss how to test and assess certain results. In chapter 3 a literature study will be conducted to provide a solid literate foundation, attempting to answer sub-questions (1) and (2). Chapter 4 answers sub-question (3) by providing a study to what criteria and requirements a method would have to fulfill in order to generate, for a food

processing factory, a satisfactory layout. This study is based on a real case-study. Chapter 5 assesses 9 previously applied methods from the literature based on the criteria presented in chapter 4 in order to find what methods work well and what methods work less well, thus answering sub-question (4). In chapter 6 a method is then finally proposed, based on the conclusions of chapter 5, answering sub-question (5), after which chapter 7 explains the implementation of this method in python using a Rhino/Grasshopper [3][2] environment. Chapter 8 presents the results of the method that was proposed and chapter 9 tries to answer the main research question and draws conclusions based on the results formulated in chapter 8.

2

Methodology

This Chapter will go over the methodology of this thesis. This research methodology is not to be mistaken with the proposed methodology that will be addressed in chapter 6. The methodology of writing this thesis finds its fundamentals in researching through design, which is vastly different then for example the researching of medical chemistry experiments where an experiment is conducted and certain data leads to certain conclusions. However, just like this thesis, they fall into the same category of research. The difference can best be explained by the diagram in figure 2.1, which shows the scale of different types of research. The scale has two directions, one from atomistic to holistic and one ranging from theoretical to empirical. The example given before would be defined to be both atomistic and empirical, as a chemical experiment would be considered to be atomistic, it is about something that is tactile and it is a well defined subject. Next to that it is also considered empirical, because it is about a clear experiment with clear results. On the exact other side of the scale would be philosophical research, which is neither atomistic or empirical, but holistic and theoretical because it deals with a whole of something and not something that is narrowed down and the experiment and expected results are not as easy to predict and black and white as with an empirical approach. The approach that is used in this thesis is based on a theoretical and atomistic standpoint: research through design. The subject that is dealt with is well defined but the results to be expected are hard to predict and thus different design must be evaluated in order to find what works and what does not.

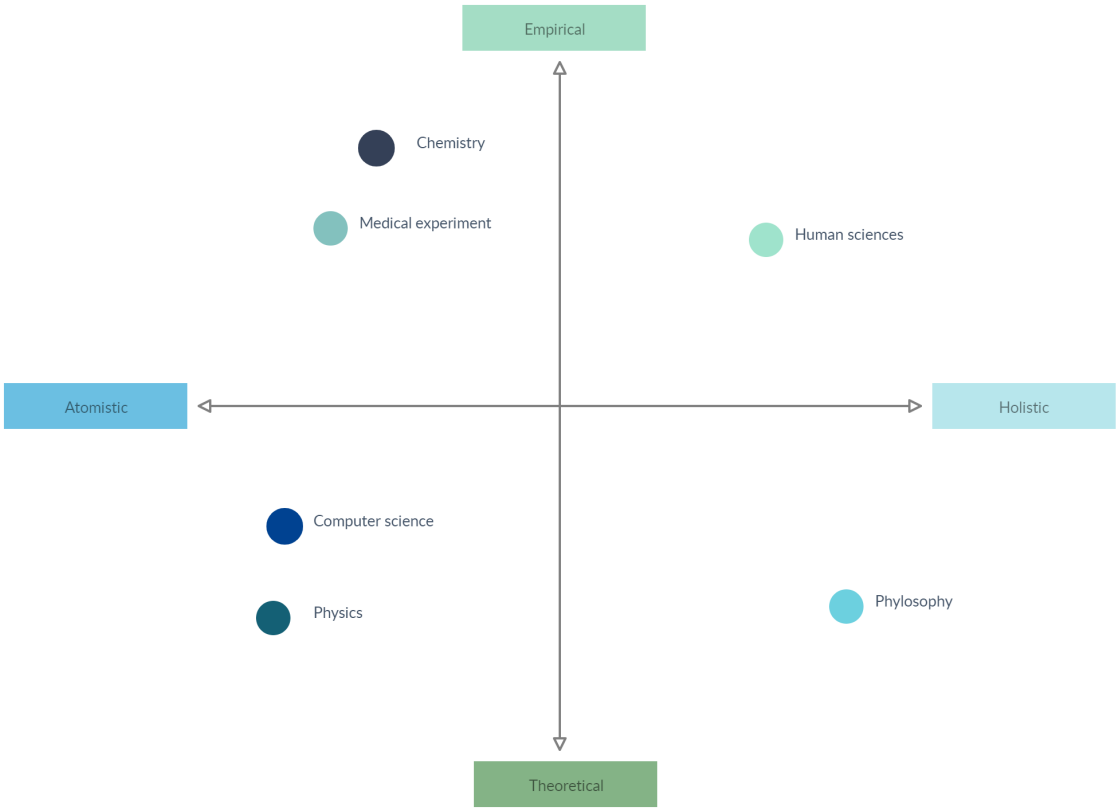


Figure 2.1: A diagram showing the different types of research.

2.1. Research methodology

The methodology consists of four parts. The first part is a literature study of the facility layout problem to get more acquainted. The goal of this stage is to find out how FLP's are formulated and how they are solved and what different approaches there are to going about this. This knowledge of previous research that has been done to solve the facility layout problem will form the basis of the next steps. The second part involves the study on an existing food processing factory and assessing its properties and needs. This way, criteria can be formulated which a potential method should be able to fulfill (e.g. The amount of departments). Thirdly, taking into consideration the knowledge gained in the first two steps, nine methods that are thoroughly researched and presented in appendix B will be analysed based on the criteria and requirements gained from the study in the second part. This way, all discussed previous methods will be tested and the results will summarize how well that method would work on this specific problem. Besides that the factors in which a method excels and doesn't excel become clearer through this analysis. Fourthly, the experimental design stage can begin by the proposal of a method on how to solve the facility layout problem in this particular case. The conclusions from part three are enough to identify which method work well in this particular case and a proposed method could be based on one or more of these analyzed methods. Although the methods found in the literature could give a basis for my own proposed method they should not be copied blindly. This is for several reasons. Firstly, even though the problems might look the same, they will probably be different in some aspects and that would favor tweaking the method. Secondly, the results of the methods in the research paper might be sufficient, but the authors most likely acknowledge some points of improvement, if possible, they should be taken into account. Thirdly, in order to push the boundaries of scientific knowledge, it is unfavorable to re-enact another's work. As the last step, the method is then carefully modelled in python in a Rhino/Grasshopper environment [3][2] where a feedback of tests loop improves the model until the results are satisfactory.

2.2. Assessment

The assessment of the proposed methodology is an important part of this thesis, for without a good assessment how could we be expecting good and reliable results? This assessment comes down to two important factors, which are verification and validation.

2.2.1. Verification

Verification is defined as: "The evaluation of whether or not a product, service, or system complies with regulation, requirement, specification, or imposed condition. It is often an internal process." - Committee. and Institute. [9]. Verification of a method is about the question: "How do we make the things right?" [5]. It is about whether, in this case, the algorithm that was written is working efficiently. The algorithm should be doing what it should be doing and in order to find out if it is, tests should be written and conducted to study its behaviour. An algorithm that gives good results could still be an algorithm that gives false results, hence the importance in the verification of the algorithm/method. This verification of an algorithm is known as test-driven-development. Test are conducted on situations where the answer the algorithm should be given is obvious. This could either be simple problems that can be verified by a hand calculation or it could be a larger problem but one with a lot of symmetries. Other tests that should be conducted are tests in extreme situations in order to find out if the algorithm still performs well under these situations.

2.2.2. Validation

Validation is defined as: "The assurance that a product, service, or system meets the needs of the customer and other identified stakeholders. It often involves acceptance and suitability with external customers." -Committee. and Institute. [9] .Validation of a method is about the question: "How do we make the right things?"[5]. At first glance, it is closely related to verification, but with validation, we are not concerned anymore with if the algorithm is working correctly, because it is assumed with verification that it is. Validation is about the effectiveness of the algorithm, whether it is giving results that are satisfactory. To test this, as specified above, toy problems can be used in order to find out how well the algorithm compares to already solved problems using different algorithms. Another method is to compare the results with a ready built factory.

3

Literature study

In this chapter the sub-research questions numbers (1) ”How can the facility layout problem be formulated?” and (2) ”What computational approaches have been proposed to solve the FLP in the literature?” are answered by looking into the literature and summarizing the most common way of formulating and solving the facility layout problem. The aim of this chapter is not to come to a conclusion of the best possible method, yet, but rather the goal is to gain an insight into what is possible and how is it done.

Numerous research has already been conducted regarding the facility layout problem since operation research made its appearance after the second world war. This research has attempted to solve the facility layout problem. However, no method has ever reached the global optimum for a problem that had beyond 15 facilities [22]. The global optimum of a problem is the solution out of all solutions that gives the absolute best result. Many solutions however fall into the category of local optimum, these are solutions that have the best values out of all neighbouring solutions, but once looked wider then their limited neighbouring space, better solutions are still available.

3.1. Introduction

The facility layout problem can essentially be taken apart into two different components, those are the formulation of the problem and the resolution of the formulated problem. Drira et al. [11] gives a comprehensive survey of the facility layout problem and talks about both the formulation and the resolution of it. Intertwined with the formulation of the problem comes the formulation of the objective of the problem and the constraints tied to it, which will also be discussed.

3.2. Layout formulations

According to Drira et al. [11] there are several ways to mathematically formulate the layout problem, there are several models which allow to express the complex relationships between all elements in a layout problems. However, all of these models are based upon the principles of discrete and continuous formulations. Drira et al. [11] suggests that graph theory be counted as a different principle. Other authors also report that graph theory is one of the main principles to formulate a layout problem [25][30] [42]. Some authors, like [37] [8], choose to only differentiate between discreet and continuous formulation as graph theory principles can be applied to both. In this thesis however, graph theory will be discussed under the different principles of layout formulations.

3.2.1. Quadratic layout formulation

Quadratic layout formulation, or put short ”QAP”, which stands for the quadratic assignment problem, is a kind of formulation to the facility layout problem that is discrete in nature. QAP is used by many researchers [8] and is proven to be NP-Complete [39]. This means that the global solution can be found, but the time required to do so increases rapidly as the number of components increases. With QAP formulations, the plant site is divided into different blocks of the same size and each facility will be assigned to one or more of these blocks (in the case of unequal areas) [11]. This formulation helps in

a way that the possible solutions are bound to the blocks that the plant site is divided into, a shape that can not be made by these blocks will never be available for evaluation. QAP also helps to standardise the input and the output, since all input and output must obey the rules of the smallest elements. In a specific case where more precision is required and craved, QAP may not be the best formulation due to its limiting nature. This can be helped by further decreasing the size of the smallest element but by doing so, the standardization advantage decreases and the number of possible solutions increases.

Chiang and Kouvelis [7] states that in general, the facility layout problem has been formulated as a QAP and describes it as a method which tries to find the optimal assignment of *n* facilities (which could be departments machines or workshops) to *n* sites, with the objective to minimize the total layout costs, which are based on the material handling costs. Chwif et al. [8], on the other hand, reports that the use of QAP in the literature has been extensively used on equal sized facility layout problems and that when QAP is applied to unequal areas facilities, often, irregular shapes are generated which are not very practical. This impracticality is a downside to working with QAP. Because of the relative simple nature of the QAP, it has often been used in dynamic problems [11]. Besides the generation of irregular shapes, QAP's have been reported to be insufficiently capable of modeling constraints such as the orientation of facilities, clearance between different facilities and the exact positions of facilities [11]. For this reason several authors have chosen to use continuous formulation instead [12] [10].

Objective and constraints

The objective in QAP is often to minimize the material handling cost, which is the product of the work-flow, travel distance and costs associated with the transportation. This is often formally stated as in equation (3.1).

$$MinCost(X) = \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{l=1}^{n} f_{ij}d_{kl}X_{ik}X_{jl} \tag{3.1}$$

Where X_{ik} and X_{jl} are the facilities *i* and *j* appointed at locations *k* and *l*, f_{ij} is the flowcost from facility i to j and d_{kl} the distance between locations k and l [11][7]. The objective represents the sum of all flow costs over pair of facilities (which are coupled to locations) [11]. The objective usually comes with constraints as in equation (3.3) and (3.4).

$$X_{ik} = \begin{cases} 1 & if\ machine\ i\ is\ allocated\ to\ location\ k \\ 0 & otherwise \end{cases} \tag{3.2}$$

$$\sum_{k=1}^{n} X_{ik} = 1, \forall i = 1, \dots, n \tag{3.3}$$

$$\sum_{i=1}^{n} X_{ik} = 1, \forall k = 1, \dots, n \tag{3.4}$$

These two constraints ensure that every facility is only allocated to one location and that every location only has one facility allocated to it, which is crucial for not having overlapping layouts [37]. Wang et al. [47] give two additional objectives to the facility layout problem, as they state that just the material handling cost does not cover the entirety of the solution. Additionally, an objective for the shape ratio of every department is added, as well as an objective for the utilization factor of the entire layout. The shape ratio factor SRF is based on the idea that the space of a single department is best utilised when the space comes closer to a square. The utilization factor for the entire layout AUF was added to add a punishing factor for space that is not used when a smallest possible rectangle is drawn around all facilities (as once again it was believed a square or rectangle utilises space optimally). These objectives are modelled as shown in equations (3.5) and (3.6).

$$SRF_{whole} = \left(\Pi_{i=1}^{n} \frac{P_i}{4\sqrt{A_i}}\right)^{1/n} \tag{3.5}$$

$$AUF_{whole} = \frac{\sum A_i}{\sum A_i + TBA} \tag{3.6}$$

Where SRF_{whole} is the shape ratio factor of the overall layout, n is the number of departments, P_i is the perimeter for the department i and the area required for department i is called A_i. AUF_{whole} is the area utilization factor considering the entire layout, TBA is the total blank area of the layout and $\sum A_i$ is the sum of all department areas [47]. These objectives are taken into account to not only focus on having the lowest material handling cost possible, but also on the practicality of having uniform and practical departments and an entire building. These practicalities ultimately save on the cost as well and could therefor also be taken into account. However, a side-note must be made that slack space is not always a bad thing, as other facilities such as cafeteria, bathrooms, offices amongst other should also be fitted into a building. Furthermore, this objective deems departments that are closer to a square as more efficient, while in reality, this does not always have to be the case. Wang et al. [47] proposed two additional constraints because of the extra objectives:

$$\sum_{i=1}^{W}\sum_{j=1}^{L} a_{ikl} \leqslant A_i \tag{3.7}$$

$$\sum_{i=1}^{W}\sum_{k=1}^{L}\sum_{l=1}^{N} a_{ikl} \leqslant H_F W_F \tag{3.8}$$

Where a_{ikl} is the total area assigned to one department, A_i is the total area required for department i, H_F is the maximum height of the plant site and therefor W_F is the maximum width [47]. The first constraint represents that there is no more area assigned to a department than required and the second constraint represents that the sum of all areas required for all the department are not greater than the total area of the plant site.

3.2.2. Mixed integer programming

In contrast to the discrete formulation of the problem, the facility layout problem is often formulated as continual. This continual formulation is often abbreviated to mixed integer programming ”MIP” [10]. Continual means that unlike the QAP, facilities can be placed anywhere and the main constraint is that they must not overlap, not even partially [10] [32] [12]. The facilities are no longer bound to small block-sized elements, but are defined by their centroid coordinates (often: x_i, y_i) and their half height H_i and half width W_i [11]. Another way to formulate the positions of the facilities are by the coordinates of one of their corners (most often the bottom left) and their full width w_i and height h_i [8].

The continuous formulation of the problem was adapted later than the discrete formulation, and is thus a more modern approach [40]. This new approach was made possible partially because the computational power grew. This computational power was needed for a continuous approach since the solution space was more broad, as it is hardly limited like the discreet formulation is. The idea of mixed integer programming is to some extend restrict the solution space by allowing some variables to be only integers instead of any number (e.g. 2 and 10 vs 2.1 and 9.7). Interestingly enough, the models for the mixed integer programming solution and the quadratic assignment solution are proven to be equivalent [23] [6]. This is because the integers restrictions divide the grid into blocks indirectly, so although the formulation of different departments may differ, if the blocks in the QAP are small enough, the same solution is possible.

Objective and constraints

The objective of the MIP approach or the continuous approach is often to minimize the material handling costs (the total transportation distance), much like it is the approach for the QAP. The MIP approach, however, adds the opportunity to add drop-off and drop-on points in the formulation, due to the way the departments are placed in the continuous plane, using coordinates. These drop-off and drop-on points are one of the ways a MIP approach can accurately model geometric constraints on a department [48]. The drop-off and -on points are added to the objective as seen in equation (3.9).

$$MinCost(X) = \sum_{i=1}^{n}\sum_{j=1}^{n} f_{ij}(|x_j^I - x_i^O| + |y_j^I - y_i^O|) \tag{3.9}$$

Where n is the total number of departments, f_{ij} is the flow from the drop-off point of department i to the drop-on point of department j, (x_i^O, y_i^O) are the coordinates of the drop-off point of department i and (x_j^O, y_j^O) are the coordinates for the drop-on points of department j [11]. Welgama and Gibson [49] added another objective to this objective function, which is to minimize the total overlap area, which is actually an important constraint in continuous approaches, to prevent the overlapping of areas. Welgama and Gibson [49] defined this constraint disguised as an objective as stated in equations (3.10) through (3.14).

$$A_{ij} \leqslant 0 \tag{3.10}$$

where

$$A_{ij} = \lambda_{ij}(\Delta X_{ij})(\Delta Y_{ij}) \tag{3.11}$$

$$\Delta X_{ij} = \lambda_{ij}\left(\frac{w_i + w_j}{2}\right) - |x_i - x_j| \tag{3.12}$$

$$\Delta Y_{ij} = \lambda_{ij}\left(\frac{h_i + h_j}{2}\right) - |y_i - y_j| \tag{3.13}$$

$$\lambda_{ik} = \begin{cases} -1 & for\ \Delta X_{ij} \leqslant 0\ and\ \Delta Y_{ij} \leqslant 0 \\ +1 & otherwise \end{cases} \tag{3.14}$$

In this case (h_i,w_i) are the height and width of department i and (x_i,y_i) are its coordinates [11].However, the overlapping of departments is not the only important constraint with the continuous formulation of the FLP. Other important constraints are presented in Anjos and Vieira [1], where constraints are presented to ensure that the departments are located inside the entire facility (3.15 and 3.16) and that the departments have the required area (3.17). Other constraints restrict the departments geometrically, with their aspect ratios (3.18) and maximum height (3.20) and width (3.19) being constrained [1]. In this case, W_F and H_F are the width and the height of the total facility and α_i is a max aspect ratio for department i.

$$x_i + \frac{1}{2}w_i \leqslant \frac{1}{2}W_F,\ \ and\ \ \frac{1}{2}w_i - x_i \leqslant W_F \tag{3.15}$$

$$y_i + \frac{1}{2}h_i \leqslant \frac{1}{2}h_F,\ \ and\ \ \frac{1}{2}h_i - y_i \leqslant h_F \tag{3.16}$$

$$w_i h_i = A_i \tag{3.17}$$

$$max\left\{\frac{w_i}{h_i}, \frac{h_i}{w_i}\right\} \leqslant \alpha_i \tag{3.18}$$

$$w_i^{min} \leqslant w_i \leqslant w_i^{max} \tag{3.19}$$

$$h_i^{min} \leqslant h_i \leqslant h_i^{max} \tag{3.20}$$

3.2.3. Graph theoretic

In graph theory, nodes and edges are used to build a graph, which is used in the FLP ultimately to minimize a certain objective, in most cases, the material handling costs. A graph consists of a set of vertices which are connected using edges, represented as $G = (V, E)$, where V is the set of vertices and E is the set of edges [17]. In general, the edges of a graph represent the adjacencies between two departments and the vertex represent the departments. There can be many different kinds of graphs but the main interest concerning FLP's lies in the range of graphs which are connected and undirected, meaning, every vertex is connected to all other vertexes by a network of edges and these edges are unoriented [17]. Areas that are bounded by the edges of a graph are considered faces. In the FLP, the outside is also considered as a face, be it an infinite one [17].

With a graph theoretic formulation, the desirability to have two departments next to each other is known, this is known as the closeness rating w_{ij} [28].Most of the time, this desirability is provided in the shape of a REL-chart, which is a matrix that describes closeness [48]. This REL-chart is designed to facilitate the consideration of factors that are of primary concern in layout designing, in this case the

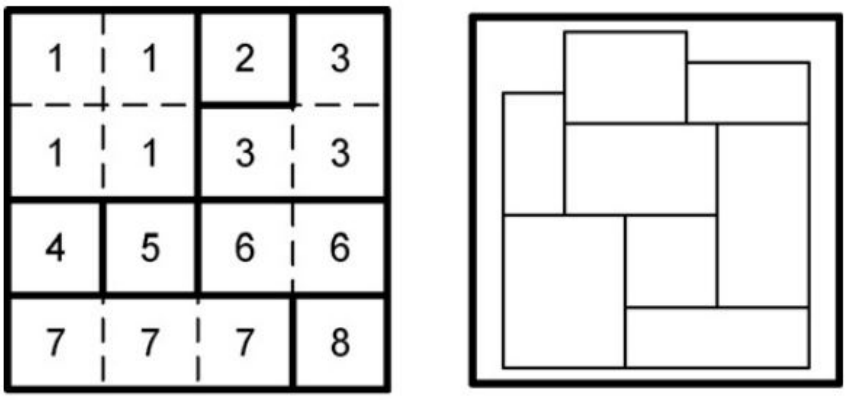


Figure 3.1: Discreet (left) and continuous (right) layout representations. Source: Drira et al. [11]

closeness between pairs of departments [25]. This closeness is often presented by the letters A, E, I, O, U and X which translate to absolutely necessary, especially important, important, ordinary, unimportant and undesirable respectively [25].

In the literature, the use of graph theory to solve the FLP is widely regarded to be split into three steps. These three steps are the constructing of the maximal planar weighted graph (MPWG), constructing the dual of this graph, and finally convert this dual into a block layout [17]. The MPWG is a special type of graph, which is maximal planar and the edges combined have the maximum weight such a graph could possess, in this MPWG, the edges represent the adjacencies and the vertex represent the departments. A graph is planar if can be drawn on one flat plane and every edge does not intersect with other edges [25]. Maximal planarity is achieved when it contains the maximum number of edges that could be inserted without losing planarity, this leads to the following interesting properties of MPWG's as shown in equations (3.21 & 3.22) [17].

$$e = 3v - 6 \tag{3.21}$$

$$f = 2v - 4 \tag{3.22}$$

Where e is the maximum number of edges in a MPG, f is the maximum number of faces, including the "infinite" face which is the outside face, and v is the number of vertices in the MPWG. The planarity of the graph is necessary because a non-planar graph converted to a layout can not be satisfied, edges that cross each other represent a relationship that conflicts. The MPWG is usually made from a REL relationship chart and attempts are made to maximise the edges on the MPWG [25]. While constructing the MPWG, many methods that were developed show signs of the umbrella effect, where one facility has adjacencies with all others, leading to unfavorable layouts [25].

In the second step, this MPWG is converted into its dual, meaning, the faces that were previously created by the edges of the MPWG are now the vertices, and then each vertex duo whose faces have a common edge is connected, creating the edges in the dual of the graph as seen in figure 3.2. [17]. This dual is a close representation of the final layout as seen in figure 3.3, adding some kinks in the edges can already make the layout graph (the dual of the MPWG) look like a block plan.

This translation from the graph layout to the block layout is the third step. This is the step that has been giving the most trouble, considering the complexity of it. Welgama et al. [48] have tried to reach this block layout by using a placement procedure based on IF and THEN rules. Kim and Kim [25] have developed a way to construct the block layout using a construction mechanism and improving the first layout through an iteration based improvement. The most notable is that this improvement happens directly to the graph instead of the block layout, constructing the layout again but with a slightly altered graph.

Solutions to the FLP based purely on graph theory are often formulated as seen in equations (3.23) through (3.26) [28], this model finds the MPWG, where $G = (V, E)$ is a weighted graph with V being the set of vertices and E the set of edges, w_{ij} is the closeness rating (often from the REL chart) indicating the desirability to locate department i next to department j, N is the set of pairs of departments that must

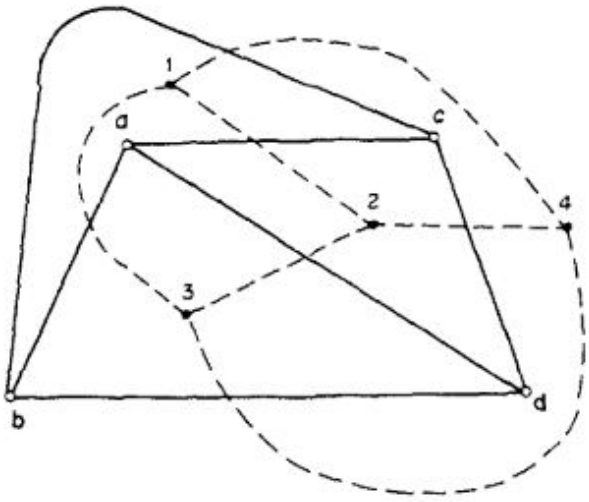


Figure 3.2: A graph and it's dual showed by the dotted lines. Source: Hassan and Hogg [17].

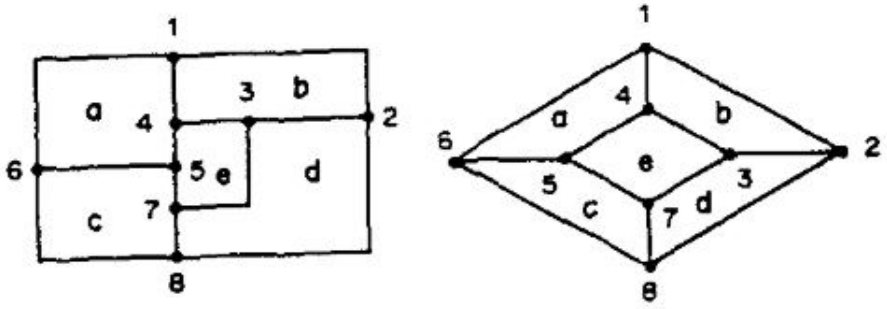


Figure 3.3: Layout representation as a graph block plan (left) and a layout graph (right). Source: Hassan and Hogg [17].

be adjacent for a feasible solution and F is the set of pairs of departments that must not be adjacent for a feasible solution Kusiak and Heragu [28]. Also x_{ij} is a variable that is equal to either 0 or 1, as seen in equation 3.27.

$$Max \quad \sum_{i \epsilon E}^{n} \sum_{j \epsilon E}^{n} w_{ij} x_{ij} \tag{3.23}$$

$$s.t. \quad x_{ij} = 1, \quad i, k \epsilon N \tag{3.24}$$

$$x_{ij} = 0, \quad i, k \epsilon F \tag{3.25}$$

$$G = (V, E) \ \ is\ planar \tag{3.26}$$

$$x_{ij} = \begin{cases} 1 & if\ department\ i\ is\ adjacent\ to\ facility\ j \\ 0 & otherwise \end{cases} \tag{3.27}$$

A graph theoretic approach is an advantageous approach because it establishes an upper bound on the solution, since a maximum number of adjacencies in a planar graph is equal to equation (3.21), maximizing the weight on these relations then gives the highest bound [17]. However, when the distances (which are not apparent in the MPWG) are added during the construction of the block layout, the objective vastly changes from one in equation (3.23) to one which is more like equation (3.1). Graph theory is also great because it lets you search the solution space by changing the graph directly instead of only altering the block layout [25].

It should be noted, however, that a graph theoretic approach has major drawbacks as well. The most inconvenience caused by a graph theoretic approach is the fact that it is generally very hard to

find a good way to perform the last step, constructing the block layout based on the dual of the graph, because it requires human intervention [48]. Other limitations are given in Hassan and Hogg [17], where it is stated that the proper length of boundaries between departments are not being taken into account when the block plan is constructed. Furthermore, although an adjacency may be present in the graph, in the block layout, implementing this adjacency may proof to be a challenge, often resulting in irregular and inconvenient department shapes. Finally, a graph theoretic approach works with qualitative and quantitative data, which it has issues with combining, leading to problems [17].

Despite the shortcomings of pure graph theoretic based approaches, graph theory is still very much present in nearly any method that tries to solve the FLP. For example, in most continuous and discrete approaches, graph theory is being used in the form of REL-charts or other matrices that define the flow between departments or the cost of moving between departments. Other concepts where graph theory takes root are for example found in a slicing tree method, where a tree, consisting of vertices and edges make a graph-based structure [50].

3.3. Resolution approaches

In all the years since the facility layout has been around, several resolution approaches have been proposed. Their goal is either to find the global (or local) optimum or to find a good solution that is adhering to certain constraints that were enforced. In general, there are two types of resolution approaches, the exact methods, and the (meta)heuristics [11].

3.3.1. Exact methods

Exact approaches aim to explore the entire solution space, and can therefor always find the global optimum, because all solutions have been evaluated. An example of such an approach is presented by Kouvelis and Kim [27], who make use of a branch and bound algorithm. Another example is by Kim and Kim [24], who also implement drop-off and pick-up points for the departments. However, like the numerous other articles that dealt with exact methods, these methods are often incapable of solving cases with a large number of departments (as mentioned earlier, the maximum was 15 departments), making the exact methods infeasible for larger problems [11].

3.3.2. Meta-heuristic methods

Since exact methods are not suitable for larger problems given that there is simply not enough time and computational power to explore the entire solution space, for larger problems, all approaches make use of some sort of meta-heuristic method [11]. Meta-heuristics are in essence algorithms that evaluate the way a problem should be solved based on previous tries and results, making smart choices to avoid a large number of the solution space and come to good (but almost never optimal) solutions. Meta-heuristic methods therefor sacrifice in the objective of the solution in order to gain enormous amounts of time and computational power, something that is very necessary when a problem reaches a size of 15 departments or bigger. Meta-heuristics try to find a good solution in a reasonable amount of time.

Amongst the different meta-heuristic methods, the most popular ones are the simulated annealing approach, the tabu search approach and the genetic algorithm approach. The first two are considered to be global search methods while the latter is classified as an evolutionary method [11]. All three of these methods are however improvement algorithms, based on an initial solution and altering it slightly to iterate towards a more optimal solution. This is the complete opposite of construction algorithms, which, as the name might imply, construct a solution only once, carefully evaluating all steps before taking one. In practise, these two type of algorithms are often combined, as construction algorithms can provide a healthy starting solution for an improvement algorithm [11].

Simulated annealing

Simulated annealing is a meta-heuristic method that is based on the principle of Monte Carlo Simulation which allows solving difficult problems through optimization [8]. Simulated annealing is an iterative process, in where, much like with the name for the physical process, the temperature is lowered (or raised in the physical process) until the solution (or metal) is satisfactory. It tries to find a near optimal solution of the solution set by evaluating neighbours of the current solution, which are often acquired by altering the current solution in some way and accepting a solution to continue with. After every iteration, the temperature is lowered until the stop criteria is true [8]. The temperature is important in

countering being stuck in a local optimum, because the higher the temperature, the more likely it is for the program to accept uphill moves [8]. This way, at the start of the algorithm the solution space is thoroughly searched (divergence) and at later stages, the method aims for finding an optimal objective value (convergence).

Tabu search

Tabu search algorithms are based on the word ”taboo”, which is a social construct prohibiting a certain something. In Tabu search, certain solutions are also prohibited. The tabu search algorithm was first used and also developed by Glover [14], and is has multiple uses, also outside of the FLP [7]. Tabu search starts with a starting solution, it can be either random or it can be given using a construction algorithm. After this initial solution, the algorithm finds the best solution out of a set of candidate solutions (which are, again, often acquired by altering the current solution in some way), however, it will refrain from choosing certain solutions if they are labeled as tabu (or forbidden), where this forbidden status is specified by certain rules which vary from project to project[7]. The solution that has the highest objective and is not labeled tabu is selected and becomes the new current solution, after which the process repeats. A forbidden status on a certain solution is not permanent and can be lifted if certain requirements are met. The idea of given this forbidden status to certain solutions is to get out of local optimum and diverge over the solution space.

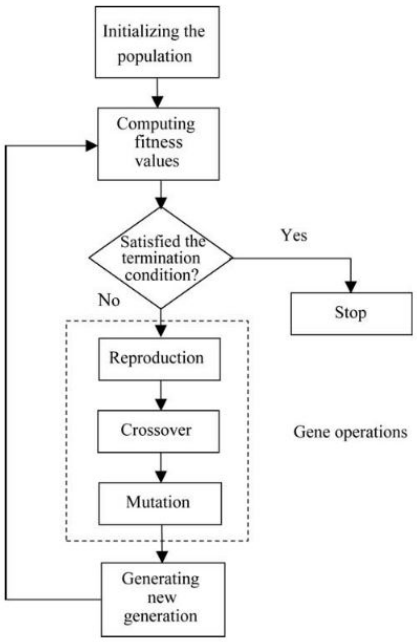


Figure 3.4: The procedures of a genetic algorithm. Source: Wang et al. [47]

Genetic algorithms

Genetic algorithms take inspiration in nature when trying to find an optimal solution. This inspiration takes the shape of the ”survival of the fittest” principle, pioneered by Charles Darwin. In nature the survival of the fittest allows for the survival of the best of any species, sometimes through the lucky random mutations they were born with. These mutated species will then produce offspring with the same benefits, who might have mutations themselves. This is the principle of evolution. In nature this process takes years and years, however, genetic algorithms used for optimization fast-forward this principle to their benefit. A random starting population is created, consisting of multiple solutions, a next generation will then be generated by reproducing the best solutions (copying them), mutating the best solutions (making random alterations in them) and crossing over the best solutions with one another (taking the first part of solution 1 and the second part of solution 2 for example) [47]. Then, the fitness of the new solutions are evaluated and the cycle starts again, until termination condition is satisfied. This process is described in figure 3.4.

Important with genetic algorithms, is to represent the layout in such a way that this encryption can be used as input for the genetic algorithm [11]. Examples of this are the representation of the layout as a slicing tree, as used in Scholz et al. [40]. Another example is the use of a space-filling-curve, as seen in Wang et al. [47] in section B.4. This space-filling-curve is represented by a string of numbers, which is perfect for using in genetic algorithms, as it can easily be reproduced, mutated and breeded.

4

Requirements and criteria

After learning about the general way the FLP can be formulated and solved, a study must be conducted to find out what would be the best course of action in this specific case of the FLP. This chapter attempts to answer sub-question (3) ”What are the requirements of a vegetable food processing factory and how can these requirements be formulated mathematically?” in order to come to a list of requirements and criteria that are needed to evaluate how well previously proposed methods for solving the FLP would comply in this particular case of the facility layout problem. Appendix K dives deep into the requirement for a vegetable food processing factory, while in this chapter, a brief summary is given as well as the conclusions.

4.1. Process

The vegetable food processing factory in question is a future vision of a factory by Hessing Supervers [20]. This factory processes different kinds of vegetables, but also mushrooms and products that include vegetables but where other ingredients are added as well, like meal salads. A trend may be that during certain times of the month or year, a certain product may be ordered more than another. This creates a situation where in one time period, a process may be utilized more than in another time period. This concludes that, a method that could take the dynamic facility layout problem into account as well being able to assess future expand-ability would be favorable over any method that is not able to do this.

In terms of the current layout, first thing that catches the eye is the fact that the factory is multi leveled. This implies that multi-layering a factory might actually be an interesting benefit and might be wise to consider when generating a layout. The upper level is almost empty, except for the mixing and buffering functions and a platform crossing from one side of the building, which is used to supervise the various processes that take place on the ground level. It can be concluded that a multilevel layout may be a good solution, but might not be since it does leave a lot of empty space, which is undesirable. Therefore, a method that would be able to assess these multilevel layouts is favored.

4.2. Departments

Considering the departments, two types of departments can be identified, namely constraint departments and free-shaped departments. Constraint departments have as a trait that they are constrained to a certain shape or shape ratio. This is because the department cannot be changed into any other shape without losing its functionality. In this study case, such departments are present and are typically those departments that house a machine or multiple machines. Since these machines are not fluid but come in one shape and one shape only it is not possible to reshape them, just to rearrange machines amongst each other. Free departments are departments that can be furnished however their shapes may be, with the exception of excessively high aspect ratios. These departments might have machines in them that are so small that every arrangement is possible or it might have a function that is not dependent much on the shape. To conclude, any method should be able to deal with constraint free and constraint departments and should be able to distinguish between them instead of just labeling the free departments as constraint departments.

The case study factory contains in total 53 departments, of which 42 are situated on ground floor level, 11 on the first floor level and 2 departments partially spawn both. While 53 seems like a very big number, it is reasonably small considering that a lot of different vegetables and products are being processed. From this information we can conclude that the chosen method will in fact have to be able to handle at least 50 facilities. Surely an exact method will then not work since almost infinite computational time would be required. Furthermore, all departments have an area requirement ranging from 75 square metres to 6000 square metres and a temperature requirement ranging from -20 degrees centigrade to 20 degrees centigrade. Since the temperature varies from minus 20 to positive 20 degrees centigrade, it can be assumed that it is likely that departments with different temperatures are going to mix, causing extra costs needed to insulate the walls to accommodate for the difference (or the cost of energy that is being used to cool the departments if they are not insulated). This concludes that the objective of the problem should not only be to minimize the material handling costs, but for more accuracy, ideally also to minimize the cost of placing departments with different temperatures next to each other.

Next, each department needs to take into account buffer spaces, as they are needed for transportation, cleaning and safety purposes. To implement this into a method would gain accuracy, but at the cost of a lot of complexity. The alternative is to partially neglect buffer spaces. In the case of the storage areas, it could be assumed that buffer spaces can be efficiently arranged manually in a later stage. In the case of the one machinery departments, it is likely that the neighboring departments will also start with buffer space / empty space and not with a wall or any other kind of solid matter. Therefore, in the cases of the one machine departments, taking into account only half of the buffer spaces on every side should be satisfying.

Lastly, some department have certain requirements for the location that have to be taken into account. These requirements for example are that certain departments need to be next to the outer perimeter like the offices but most importantly: the inbound and outbound goods. Some of these requirements can already be put in the flows between facilities when the outside is also considered a facility. However, in some cases this does not work appropriately, since the offices have no flow to the outside for example. Therefor, it might be a good addition to implement a visibility objective of some sort.

4.3. Flows

There are actually multiple type of flows in the factory. In total there are 8 different kinds of flows, which are:

- People
- Raw materials
- Packing materials
- Semi-finished goods
- Finished products
- Outbound crates
- Inbound crates
- Garbage

While some flows may be considered to be practically the same (like the raw materials, semi-finished goods and finished goods) and all flows should essentially be minimized as they have a certain cost per distance, however, the cost and the different needs that are bound to each flow are different, and therefore, should ideally be treated as different. The numbers of the different kind of flows between the departments are hard to come by, the exact movement between the different departments are not recorded, not for the materials and let alone for the movement of the people. In order to reduce the complexity, it is possible to combine all flows into one type of flow, granted that the cost for these flows do not differ too vastly.

4.4. Transportation

As for the transportation between the departments, it is either done vertically or horizontally. The distance for horizontal movement can be measured in Euclidean distance or Manhattan distance and the vertical distance in case of a multi-layer problem can be discarded since the height of a floor is constant. For extra accuracy, the distance can be measured from pick-up and drop-off points of the machines, instead of their centroids, however, this does add a lot of complexity.

4.5. Conclusion

The research concludes 14 different criteria that are important for a method to solve, which are listed in table 4.1 and come forth from the previous research. Because 14 different criteria is a scale that is out of scope for this thesis, several will not be able to be implemented in the final method. Because each of these criteria has shown importance in the research, they should be ranked based on their priority. Furthermore, the knowledge gained in section 3 should already give insight on how easy a criteria has been to implement in previous solutions.

Both these rankings are given either one digit(1), representing low priority and low ease of implementation (e.g. the lack of implementation in the literature), two digits (10) representing a medium priority and a medium ease of implementation (e.g. it has been implemented before, but with the needed difficulty), three digits (100) representing high priority and high ease of implementation (e.g. several researches have successfully implemented this criteria) or four digits (1000) representing the highest priority and very high ease of implementation (e.g. several researches have successfully implemented this criteria with little extra effort).

The product of the two scales will determine the final priority of the criteria, which in table 4.1 is given the name P*I, which stands for priority times implementation. These final scores determine whether the criteria will be implemented in this thesis. The criteria that score 6 or 7 digits, are criteria that aim to be implemented in this thesis. This means that from the start, efforts will be made to take these criteria into account and a chosen method must be developed in such a way that these criteria are implemented. Additionally, the criteria that have 5 digits will be taken into account, but only if the time allows it. This means that at an early stage, little to no effort will be made to implement these criteria, and as time progresses, considerations will be made to determine whether or not these criteria will be implemented.

The criteria with 6 or 7 digits that will be implemented from the start are:

- Empty space punishment
- Geometric constraints
- Aspect ratio constraints
- Scalability
- Temperature differences
- Unequal areas
- Location constraints

The criteria with 5 digits that will be implemented only if the time allows it are:

- Multi-level
- Multiple flows
- Drop-off and Pick-up points

This leaves the following criteria with 4 of fewer digits to not be regarded in this thesis:

- Dynamic

- Expandability
- Vision
- Buffer spaces

Dynamic has been evaluated to be of medium priority, because in most cases, a dynamic layout is a nice add-on instead of a must have criteria. Furthermore, an alternative to providing a dynamic layout solver, is to take the averages of the different flows during the different time period and treating the layout as static nonetheless. The initial implementation is evaluated to be medium, since only one research has done so successfully and this provided the needed difficulties.

Expand-ability has been evaluated to be of the lowest priority, mainly because expand-ability can indirectly be implemented by making use of other criteria like the location constraints and the number of departments, by simply adding extra departments and possibly constraining their location will simulate a future layout that is already expanded. Also, the implementation has been evaluated to be low, since no researches have directly tried to implement this.

Multi-level has been evaluated to be of medium priority, for the reason that multilevel is not a necessity for a layout to work, but merely an amazing extra. Furthermore, the implementation was evaluated to be of very high ease of implementation, since multiple researches have successfully implemented this criteria without lot of difficulty.

Vision has been evaluated to be of medium priority, because in a case where vision is not taken into account, an alternative could be to provide a physical flow between the departments, that is high enough to ensure vision. Of course this is not as accurate as actually implementing this criteria, but for the time being it could be enough. Also, vision was evaluated to be of low ease of implementation, since no research has implemented this.

Empty space punishment has been evaluated to be of high priority, since empty space would add to the total costs, which, if not implemented, would defy the purpose of trying to lower the costs of material handling. Implementing the punishment of empty space is very easy, multiple researches have successfully implemented this criteria without facing a lot of difficulty.

Geometric constraints has been evaluated to be of the highest priority. This is really what differentiates this method from older methods, which had a hard time implementing these constraints. Geometric constraints is vital to these solutions as the departments are separated to be close to 50% free-shaped and 50% constraint. The implementation of this criteria is evaluated to be 100, since some researches have successfully implemented this.

Aspect ratio constraint has been evaluated to be of the highest priority. This criteria is an addition to the geometric constraints, and should therefor be treated to be of the same priority. Aspect ratio constraints is what helps determine how ”free” the free-shaped departments are and how the exact dimensions of the constraint departments can be determined. Implementing this has been proven to go without difficulties by many researches, and has therefor be awarded 1000 in the implementation category.

Scalability has been evaluated to be of the highest priority. This is because the number of departments that are present in a factory can vary a lot, but can be as high as over 50 departments. Obviously, the method is not going to do much good if it only works up to 10 departments. Previous research has proven that implementing over 50 departments does not come with much difficulties, if the method is fast enough and simplified enough. Especially in the last 10 years, computational power has overcome the difficulties that existed on this matter in the past.

Temperature differences has been evaluated to be of high priority, since this is something that is very typical to most food processing factories, furthermore, including temperature differences in the objective adds more accuracy to the final layout being the optimal one. Research has not worked with temperature differences yet. However, implementing this is a matter of simple mathematical equations when determining the objective score of a certain layout, therefor, 1000 has been awarded.

Unequal areas has been evaluated to be of the highest importance. Equal sized facility layout problems make the problem simpler, but have not yielded very satisfying and accurate layouts, certainly not when the area of a department can range between 100 square meters and 3000 square meters. In the literature, this problem of unequal areas has been tackled often instead of an equal sized setup, implementing this proved to be of little difficulty.

Criteria name	Priority	Implementation	P*I
Dynamic	10	100	1000
Expandability	1	1	1
Multi-level	10	1000	10000
Vision	10	1	10
Empty space punishment	100	1000	100000
Geometric constraints	1000	100	100000
Aspect ratio constraint	1000	1000	1000000
Scalability	1000	1000	1000000
Temperature differences	100	1000	100000
Unequal areas	1000	1000	1000000
Buffer spaces	10	1	10
Location constraints	1000	100	100000
Multiple flows	100	100	10000
Drop-off and Pick-up points	100	100	10000

Table 4.1: All criteria that were concluded from the vegetable food processing factory research. each criteria is evaluated on priority and easy of implementation, the product of the two determines whether the criteria will be implemented directly (6 or 7 digits), if time allows it (5 digits) or not (4 or less digits).

Buffer spaces has been evaluated to be of medium priority, since these spaces can easily be incorporated in the area of the departments, taking away a lot of difficulty at the cost of very little accuracy. Additionally, for larger free-shaped departments like storage rooms, a separate algorithm can be run to find a satisfactory layout for the storage racks within that room. There has not been a single literature study that took these buffer spaces into account.

Location constraints has been evaluated to be of the highest priority. This is because certain departments are not functioning when they are not at a certain location, like inbound goods rooms. In the literature, these locations constraints are often applied, sometimes with more difficulty and sometimes with less.

Multiple flows has been evaluated to be of high priority. Since for an accurate layout, all flows must be taken into account. It is possible to simplify these flows into one flow if needed. In the literature these multiple flows are really always simplified to one flow, but since adding multiple flows would only add to the objective and not bring other difficulties, it has been awarded a score of 100 in implementation.

Drop-off and Pick-up points has been evaluated to be of high priority, since adding this criteria would add a lot to the accuracy of the final generated layout. However, since it adds a lot of complication, it does not have the highest priority. The literature has showed one interesting example where these points were successfully implemented without much difficulties.

5

Methods evaluation

Following the literature research conducted in chapter3 and the requirement study conducted in chapter 4, the methods can now be evaluated based on their performance meeting the requirements, basically answering sub-question (4): ”To what extent do the proposed methodologies in the literature they comply with this specific case of the FLP?” . These requirements are the requirements after they have been assessed on first glance implementation (how well it can be solved) and priority. These requirements are split into two categories, where one category illustrates the requirements a method must perform well in (hard requirements) and another category makes this optional but preferable (soft requirements) . Nine methods from the literature, which are vastly elaborated in appendix B, are being evaluated on 10 different requirements, of which 7 are hard requirements. These evaluation leads to a score ranging from 0 to 5, where all 6 digits imply something different:

0 Is not possible using this method in any way.

1 Already implemented but not working very well.

2 Could be implemented, but has significant impact/ very complicated.

3 Could be implemented.

4 Could be implemented easily.

5 Already implemented and working.

These scores will then be multiplied by either a factor 1 for soft criteria or a factor 2 for hard criteria. All scores are then added to create a total score. Note that this total score would range from 0 to 85 (the product of the multipliers added and the maximum score).

5.1. Evaluation

The results in tables 4.1 through 4.9 are listed below, every table concludes the score for one of the nine methods as presented in appendix B. In the caption the source and title of the paper is listed, as well as whether the method is based on a QAP or MIP formulation. Table 5.10 shows the comparison between the nine methods.

Criteria Name	Multiplier	Score out of 5	Total
Multi-level	1	3	3
Empty space punishment	2	1	2
Geometric constraints	2	5	10
Aspect ratio constraint	2	5	10
Scalability	2	2	4
Temperature differences	2	3	6
Unequal areas	2	5	10
Location constraints	2	4	8
Multiple flows	1	4	4
Drop-on and -off points	1	5	5
		Total	62

Table 5.1: The evaluation for the MIP based method ”A solution to the facility layout problem using simulated annealing” by Chwif et al. [8].

Criteria Name	Multiplier	Score out of 5	Total
Multi-level	1	3	3
Empty space punishment	2	0	0
Geometric constraints	2	2	4
Aspect ratio constraint	2	0	0
Scalability	2	2	4
Temperature differences	2	2	4
Unequal areas	2	3	6
Location constraints	2	4	8
Multiple flows	1	5	5
Drop-on and -off points	1	3	3
		Total	37

Table 5.2: The evaluation for the QAP based method ”An integrated simulation-based optimization technique for multi-objective dynamic facility layout problem” by Pourhassan and Raissi [37].

Criteria Name	Multiplier	Score out of 5	Total
Multi-level	1	2	2
Empty space punishment	2	4	8
Geometric constraints	2	4	8
Aspect ratio constraint	2	5	10
Scalability	2	5	10
Temperature differences	2	3	6
Unequal areas	2	5	10
Location constraints	2	4	8
Multiple flows	1	3	3
Drop-on and -off points	1	2	2
		Total	67

Table 5.3: The evaluation for the MIP based method ”An improved two-stage optimization-based framework for unequal-areas facility layout” by Anjos and Vieira [1].

Criteria Name	Multiplier	Score out of 5	Total
Multi-level	1	0	0
Empty space punishment	2	5	10
Geometric constraints	2	2	4
Aspect ratio constraint	2	3	6
Scalability	2	3	6
Temperature differences	2	4	8
Unequal areas	2	5	10
Location constraints	2	4	8
Multiple flows	1	5	5
Drop-on and -off points	1	0	0
		Total	57

Table 5.4: The evaluation for the QAP based method ”A solution to the facility layout problem by genetic algorithm” by Wang et al. [47].

Criteria Name	Multiplier	Score out of 5	Total
Multi-level	1	2	2
Empty space punishment	2	5	10
Geometric constraints	2	4	8
Aspect ratio constraint	2	4	8
Scalability	2	2	4
Temperature differences	2	3	6
Unequal areas	2	5	10
Location constraints	2	5	10
Multiple flows	1	3	3
Drop-on and -off points	1	0	0
		Total	61

Table 5.5: The evaluation for the MIP based method ”Facilities layout: A knowledge-based approach for converting a dual graph into a block layout” by Welgama et al. [48].

Criteria Name	Multiplier	Score out of 5	Total
Multi-level	1	3	3
Empty space punishment	2	5	10
Geometric constraints	2	2	4
Aspect ratio constraint	2	2	4
Scalability	2	3	6
Temperature differences	2	4	8
Unequal areas	2	4	8
Location constraints	2	3	6
Multiple flows	1	4	4
Drop-on and -off points	1	0	0
		Total	53

Table 5.6: The evaluation for the QAP based method ”Graph theoretic heuristics for unequal-sized facility layout problem” by Kim and Kim [25].

Criteria Name	Multiplier	Score out of 5	Total
Multi-level	1	4	4
Empty space punishment	2	5	10
Geometric constraints	2	0	0
Aspect ratio constraint	2	0	0
Scalability	2	5	10
Temperature differences	2	4	8
Unequal areas	2	0	0
Location constraints	2	4	8
Multiple flows	1	3	3
Drop-on and -off points	1	0	0
		Total	43

Table 5.7: The evaluation for the QAP based method ”An improved tabu search heuristic for solving facility layout design problems” by Chiang and Kouvelis [7].

Criteria Name	Multiplier	Score out of 5	Total
Multi-level	1	4	4
Empty space punishment	2	4	8
Geometric constraints	2	2	4
Aspect ratio constraint	2	2	4
Scalability	2	5	10
Temperature differences	2	3	6
Unequal areas	2	5	10
Location constraints	2	4	8
Multiple flows	1	3	3
Drop-on and -off points	1	4	4
		Total	61

Table 5.8: The evaluation for the QAP & MIP based method ”Facility layout by collision detection and force exertion heuristics” by Sikaroudi and Shahanaghi [42].

Criteria Name	Multiplier	Score out of 5	Total
Multi-level	1	3	3
Empty space punishment	2	5	10
Geometric constraints	2	5	10
Aspect ratio constraint	2	5	10
Scalability	2	5	10
Temperature differences	2	3	6
Unequal areas	2	5	10
Location constraints	2	2	4
Multiple flows	1	4	4
Drop-on and -off points	1	0	0
		Total	67

Table 5.9: The evaluation for the MIP based method ”STaTS: A Slicing Tree and Tabu Search based heuristic for the unequal area facility layout problem” by Scholz et al. [40].

	ML	ESP	GC	ARC	S	TD	UE	LC	MF	DoDo	Total
Chwif et al. [8]	3	2	10	10	4	6	10	8	4	5	62
Pourhassan and Raissi [37]	3	0	4	0	4	4	6	8	5	3	37
Anjos and Vieira [1]	2	8	8	10	10	6	10	8	3	2	67
Wang et al. [47]	0	10	4	6	6	8	10	8	5	0	57
Welgama et al. [48]	2	10	8	8	4	6	10	10	3	0	61
Kim and Kim [25]	3	10	4	4	6	8	8	6	4	0	53
Chiang and Kouvelis [7]	4	10	0	0	10	8	0	8	3	0	43
Sikaroudi and Shahanaghi [42]	4	8	4	4	10	6	10	8	3	4	61
Scholz et al. [40]	3	10	10	10	10	6	10	4	4	0	67

Table 5.10: The comparison of the 9 methods side by side.

5.2. Conclusion

As the most important conclusion, the continuous approach (MIP) of formulating the problem was chosen over the discrete (QAP) approach of formulating the problem. Seeing as both methods have been applied in the past, both having success in different areas and drawback in others, it might be a hard choice to eliminate one approach before even starting. A quadratic assignment approach to the problem leaves the solution space smaller and is also more comprehensive, whereas the continuous or mixed integer programming approach has a much wider solution range and may be less comprehensive.

Through the thorough evaluation of the methods provided by literature, however, it becomes clear that the continuous approach has advantages over the discrete approach, in this particular case of the FLP. The voxelated approach of the quadratic assignment problem leaves the problem that the definite shape of one department (unless it consists solely of one voxel) is hard to influence. For example, an aspect ratio would ultimately be very hard to implement using the quadratic assignment problem. This is not just based on logic, but also the literature has found this. Scholz et al. [40] describes that the main drawback on discreet approaches and graph theoretic approaches is the geometric constraints that can not be considered sufficiently.

Scholz et al. [40] also states that the voxelization makes it hard if not impossible to assure connectivity and the given shapes of facilities. Graph theoretic approaches have problems of the same nature. With graph theoretic approaches, the dimensions of the facilities cannot be considered during the optimization. Thus, they can only be incorporated when the optimization is already done, reducing the chances at satisfactory results [40].

As the aspect ratio and the to model geometric constraints are considered of high importance, as pointed out by the research on the criteria for generating a food factory, the question is whether these shortcomings are solved by using a continuous approach. The answer is that these continuous approaches are actually very fit to model geometric constraints on departments, with one major drawback: only rectangular shapes are possible. However, as Welgama et al. [48] points out, it is clear that in a environment focused on manufacturing, facility shapes will mainly be rectangular. To use a discrete or graph theoretic environment and voxelating the departments in order to find an optimal solution can not be justified if the end result can not actually facilitate all the machines that need to be fitted into a department [48].

This concludes that in this case, the continuous approach offers an advantage which is so large that not using it over the discrete approach after the knowledge gained in this section would be an unwise decision. Therefore, the continuous approach is chosen over the discrete approach because it is very capable of modeling geometric constraints.

Another conclusion is that different methods have different purposes and thus not all methods fit the requirements at hand. Some methods had trouble meeting the free shapes & constraint shape requirements, which were mainly discreet oriented approaches, while other methods had trouble meeting the 50 departments requirement or the location constraints requirement. As it turns out, none of the presented methods would suffice when implemented directly to generate a vegetable processing factory's layout. Since for every method, at least one of the criteria is not met sufficiently.

However, some of these methods were outstanding in some fields while only lacking minorly in others. This made several of these methods a good foundation when designing a method for ourselves. And even when a method showed little promise, some of the fields were covered adequately enough

to add to the available knowledge-base for solving these fields. Thus, it can be concluded that for a working method, one of the higher ranking methods found in the literature should be used as a foundation, while the other methods are helpful in overcoming the shortcomings of that very foundation. There are in fact two methods that are promising, which are the methods by Anjos and Vieira [1] and Scholz et al. [40]. The method by Anjos and Vieira [1] was chosen over the method by Scholz et al. [40] because it can be combined with the method by Sikaroudi and Shahanaghi [42], since it performs very well in some aspects and combining it with the method by Anjos and Vieira [1] would make a more interesting attribution to the vast academic knowledge. The next chapter explains the proposed methodology.

6

Proposed methodology

This chapter explains the chosen methodology, starting with its problem formulation, objectives and constraints and with it answers the sub-question (5) ”What are the main objectives and constraints to satisfy considering the generation of a layout for a vegetable processing factory with a given program of requirements and flows between facilities?”. The proposed methodology will solely be explained in this chapter, and only be elaborated in the next chapter. The chosen method is based off of the analysis done in the previous two sections.

6.1. Problem formulation

The problem at hand is the problem described in section 1.4, the facility layout problem. More specifically, the problem is fine-tuned for the generation of a layout for a food processing factory. This means that the material flow factor costs is the main problem, but it will be referred to as flow-cost since other flows might be present. The problem is broadly the problem stated in the main research question:

> ”How to computationally generate a layout of a vegetable processing factory given a program of requirements and flows between facilities as a matrix using a mathematical approach, minimizing the travel distance of goods needed for a product to be manufactured?”

This minimizing of the distance between goods extends to materials, crates, garbage and employees. However, for simplification in this thesis, the toy problems will essentially have all these flows combined into one number.

6.1.1. Shaping the departments

The method is primarily based on the assumption that it should be feasible for producing layouts for factories, in particular, vegetable food factories. Since in these type of factories the departments are split heavily between departments that can take multiple dimensions (e.g. a storage room with a flexible organisation of storage racks) and constraint departments that can really only take on one set of width and height (e.g. a room with one big machine). Hence all departments are defined by their area (A_i) and max aspect ratio a_i [1]:

$$max\left\{\frac{w_i}{h_i}, \frac{h_i}{w_i}\right\} \leqslant a_i \quad for\ all\ i\ in\ \{i, .., n\} \tag{6.1}$$

$$w_i h_i = A_i \quad for\ all\ i\ in\ \{i, .., n\} \tag{6.2}$$

The aspect ratio constraints will be discarded in the first-stage model (graph decent) as its purpose is to find relative relations between departments, hence, all departments in the first-stage model will be modeled as circles with radius r_i, an area requirement $A_i = \pi * r_i^2$ and a set of coordinates x_i & y_i. In the second-stage model, all departments will be modelled as squares by assigning a width w_i and height h_i and a set of coordinates x_i & y_i.

6.1.2. Overlap

In order to restrict the departments from overlapping with the total facility two constraints are introduced in both the first-stage and the second-stage model:

$$x_i + \frac{1}{2}w_i \leqslant \frac{1}{2}W_F, \ and \ \frac{1}{2}w_i - x_i \leqslant W_F \quad for\ all\ i\ in\ \{i,..,n\} \tag{6.3}$$

$$y_i + \frac{1}{2}h_i \leqslant \frac{1}{2}H_F, \ and \ \frac{1}{2}h_i - y_i \leqslant H_F \quad for\ all\ i\ in\ \{i,..,n\} \tag{6.4}$$

Where W_F and H_F are equal to the total facility's width and height, respectively. In case of the first-stage model: w_i = h_i = r_i. The way that the overlap between facilities is constrained differs between the two models. In the first-stage model (gradient descent), overlap is allowed, however, it does add to the objective by adding the one dimensional overlap $r_i + r_j - d_{ij}$ to the distance in the case there is overlap. In this case, d_{ij} is the euclidean distance between the centroids of the departments.

$$d_{overlap} = \lambda_d(r_i + r_j - d_{ij}) \tag{6.5}$$

$$\lambda_d = \begin{cases} 0 & for\ r_i + r_j - d_{ij} \leqslant 0 \\ 1 & otherwise \end{cases} \tag{6.6}$$

In the case of the second-stage model, where the actual layout is computed, overlap is not allowed, and thus constraints are added to prevent the departments from overlapping. Since the solver does not take non-linear constraints, a boolean variable is added which is implemented in such a way that the absolute value between coordinates is found, see equations (6.7) through (6.9).

$$\lambda_{ij} = \begin{cases} -1 & for\ \ x_i \leqslant x_j\ and\ y_i \leqslant y_j \\ +1 & otherwise \end{cases} \tag{6.7}$$

$$2\lambda_{ij}(x_i - x_j) \geqslant w_i + w_j \quad for\ all\ i,j\ in\ \{i,..,n\} \tag{6.8}$$

$$2\lambda_{ij}(y_i - y_j) \geqslant h_i + h_j \quad for\ all\ i,j\ in\ \{i,..,n\} \tag{6.9}$$

6.1.3. Objective

The objective of this method is to minimize the flow-cost of the layout. For a two department layout, this would be the flow between the departments, (f_{ij}), multiplied by the distance between the departments, (d_{ij}), multiplied by the cost per flow per distance of moving materials between the two departments, (c_{ij}). Naturally, the flow-cost for a layout consisting of more than two departments would then be the sum of the flow-costs for all sets of departments. This then leads to the following objective for the flow-cost, which is similar to the objective for minimizing the material handling costs as found in Wang et al. [47].

$$MinFlowCost = \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}f_{ij}d_{ij} \tag{6.10}$$

The flow f_{ij} between the departments is provided using a matrix of i rows by j columns, providing the flow between departments i and j at row i and column j. The cost c_{ij} is taken to be one for simplification. The distance d_{ij} between the departments is different for both models. In the first-stage model, the distance is measured as the euclidean distance as seen in equation (6.11), additionally, the distance for overlapping, as seen in equation (6.5) is assimilated in the distance equation . The second-stage model however, will make use of the rectilinear distance, like seen in equation (6.12), as the chosen solver is only capable of performing linear operations.

$$d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} + d_overlap \tag{6.11}$$

$$d_{ij} = |x_i - x_j| + |y_i - y_j| \tag{6.12}$$

6.1.4. Relative location constraints

The whole idea of splitting the method into two models is to gain an early insight in what arrangement of departments would work near optimal without requiring too much computational power. This early insight does not have to be accurate in aspect ratio or other geometric constraints, nor in overlapping. The loss off accuracy in that case gains the benefit that the model becomes much faster. This way, a near optimal arrangements can be found in the first-stage model and this arrangements data can be passed onto the second-stage model, where all previous requirements must be met, but due to the extra constraints found in the first-stage model the second-stage model should be able to converge faster.

The second-stage models therefor takes all constraints that were found in the first-stage model. These will be constraints between all departments i and j that will separate them either horizontally or vertically. To determine this separation the differences between x and y coordinates of both departments is evaluated. Should δx be 1.5 times bigger than δy, then a separation constraint will be issued to separate department i to the left of j if i was to the left of j in the most optimal solution from the first-stage model, or to the right of j if i was to the right of j in the most optimal solution from the first-stage model. In the same way, should δy be 1.5 times bigger than δx, then a separation constraint will be issued to separate department i to the top of j if i was to the top of j in the most optimal solution from the first-stage model, or to the bottom of j if i was to the bottom of j in the most optimal solution from the first-stage model. If neither of the two differences is significantly bigger than the other (a factor 1.5), no constraint will be issued. The idea to split the method into two models to first find these relative location constraints and how to find these constraints was originally proposed in Anjos and Vieira [1]. The main difference is that in that paper, the relative location constraints will be issued even if one δ is only slightly bigger than the other.

6.2. Proposed methodology

The method proposed in this model is separated into two stages, a first-stage model and a second-stage model. The first-stage model is a simplified version that does not take into account aspect ratio and other geometric constraints and takes the overlap as a soft constraint. This is in order to quickly find a good solution and then transfer the relative location information to the second-stage model in the shape of constraints. This way, the hypothesis is that the second-stage model, which does take all the hard constraints as described in the previous section into account, will be able to converge into a good solution faster.

6.2.1. First-stage model

The first stage model makes use of the gradient descent approach to iteratively move towards a better objective function (6.10). This idea was already proposed in Sikaroudi and Shahanaghi [42], where the term force exertion heuristics was used in order to get to a layout. The main difference with this approach is that here the gradient descent approach does not need to find a feasible solution, only an idea about the relative positions of the departments in a good solution. This gradient descent approach builds upon the fact that the gradient of any function can be used to find the direction of steepest ascent (or descent when taking the negative of this direction). This approach works for any objective and for multiple variables by taking all the partial derivatives and combining these in a vector, which is the direction of steepest ascent. This principle is explained in equation (6.13) for an objective f consisting of two variables: x and y.

$$\nabla f = \begin{bmatrix} \frac{df}{dx} \\ \frac{df}{dy} \end{bmatrix} \tag{6.13}$$

$$f(x_i, y_i) = \sum_{j=1}^{n} f_{ij}(d_{ij} + \lambda_d(r_i + r_j - d_{ij})) \tag{6.14}$$

$$\nabla f(x_i, y_i) = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \; for\ \lambda_d = 1 \tag{6.15}$$

$$\nabla f(x_i, y_i) = \begin{bmatrix} \frac{df}{dx} \\ \frac{df}{dy} \end{bmatrix} \; for\; \lambda_d = 0 \tag{6.16}$$

$$\frac{df}{dx_i} = \sum_{j=1}^{n} \frac{(x_i - x_j) f_{ij}}{\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}} \tag{6.17}$$

$$\frac{df}{dy_i} = \sum_{j=1}^{n} \frac{(y_i - y_j) f_{ij}}{\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}} \tag{6.18}$$

The principle of the gradient descent is used in this first stage model by iteratively moving every department in the direction of greatest descent by taking the gradient of the objective. In this objective, the variables are the x_i and y_i coordinates of department i whilst the coordinates of every other department j are considered to be a constant. This gradient is then calculated for every department, after which the departments are moved resulting then in a different gradient thus creating an iterative process in which all departments move towards the lowest objective. This iterative process is explained in algorithm 1.

Interestingly, the partial derivatives between any two departments that are not overlapping ($\lambda_d = 0$) are equal to the difference in x-coordinates or y-coordinates divided by the distance between the two departments and multiplied by the flow, as seen in equations (6.17) and (6.18). In essence this would mean that the gradient would be a normalised vector that origins from one department and points to the other. When $\lambda_d = 1$, however, all the variables disappear from the objective, leaving the gradient to be [0,0], meaning that any direction and no direction will improve the objective considering just these two departments.

An important addition is the inclusion of the overlap repulsion vectors, these vectors are introduced to remove overlap whenever there is overlap. The vector acts in the opposite direction of the department j that department i has overlap with. A last vector prevents the departments from overlapping with the total facility, although in this first-stage model the total facility is merely present to prevent the departments from moving out of bounds.

6.2.2. Second-stage model

In the second-stage model, the relative location constraints from the first-stage model will be implemented in addition to all hard constraints as mentioned in the previous section. The second-stage model takes all the constraints and objective and models that in Google's OR-tools [36], using its CP-Sat solver, a linear constraint programming solver capable of running optimizations by using only integers. These limitations meant that some constraints found in the literature were rewritten, but other constraints involved problems, like the total area constraint, the next chapter covers dealing with these problems.

6.2.3. Assessment

To assess if the method is working, it will be tested on two toy problems that are relatively small, 8 and 12 departments. This way, mistakes in the workings of the method can be found faster and certain aspects can be tested easier, not to mention the computational time it would save. To test if the method is doing the right things, it will be tested in obvious situations where the outcome is easily recognisable even without the computer. To ensure that the results can be assessed by a human, the processes will be as much as possible visualised. To test if the method is doing the things right, existing problems can be analysed and the results can be compared to the existing results. These problems can be real existing factories where the objective was calculated by hand, or toy problems existing in literature.

7

Generation of the layout using python

This chapter covers the process of designing the algorithms for both stages, and how the two stages link together. It gives insight in the pseudo-codes, as well as premature results. The actual final results will be discussed in the next chapter, after which a conclusion can be drawn.

Both stages will be modeled in an Rhino/Grasshopper [3][2] environment, with Cpython [38] components for any parts of the code that need external modules and GHpython components to visualize as much as possible. Here and there regular grasshopper nodes will be used for small data visualisations.

7.1. first-stage model

In the first stage model, the input is being processed so the departments can take shape and are randomly placed on the plane, adhering to their area requirements by setting the radius of the departments. Then, using the gradient descent approach, all vectors of steepest descent are calculated for all departments. In this stage the repulsive vectors for possible overlapping between departments and the total facility are also computed after which all departments are moved a set distance in the combined direction of all these vectors. This process repeats for 800 iterations, after which the iteration with the best results will give the final layout of this first-stage model. Due to the random starting positions, every test will end up differently. Multiple test are conducted to try to come be even better solutions. This final best solution will be used as a basis for the second-stage model, where the relative locations will be used as constraints.

7.1.1. Pseudo-code

Algorithm 1 contains the pseudo-code for the first stage model. It covers the process of finding the optimal layout for one instance of random placement of the departments. Note that the optimal solution is not found at the last iteration, but rather it is stored when it is achieved.

Algorithm 1: Pseudo-code for the first-stage model

```
Import necessary modules
Import data from excel using pandas
for i in range(nr_departments) do
    Radius = sqrt(Area / π)
    x = random_integer (min,max)
    y = random_integer (min,max)
end
for k in range(max_iterations) do
    for j in range(nr_departments) do
        for j in range(nr_departments) do
            if i!=j then
                Compute partial derivative df/dx_i
                Compute partial derivative df/dy_i
                Compute flow attraction vectors
                Compute repulsive vectors for overlapping
                Compute repulsive vectors for overlap with outside facility
                Add weights to all vectors
                Compute the master vector for each department
            end
        end
    end
    if Department = free to move then
        Move the departments in the direction of the master-vector a set distance
    end
    Compute the Objective
    for i in range(nr_departments) do
        for j in range(nr_departments) do
            if i!=j then
                Objective_ij = distance_ij * flow_ij
                Objective = Objective + Objective_ij
            end
        end
    end
    Append Objective to Objectives
    if Objective == min(Objectives) then
        Save x and y positions
        Save iteration number
        Save objective and overlap amount
    end
end
```

7.1.2. Input

The input required for the first-stage model is all contained in one excel file, where different data frames are separated by different sheets. This information is being read using the module pandas. The three different data frames that are created contain different information, the first data frame contains information about the size, geometrical constraints and the temperature of the departments, as seen in figure 7.1, the second sheet contains the information for the flows between all departments as a matrix, as seen in figure 7.2. Please note that if a project has multiple types of flows, multiple matrices will be present and loaded into the project, simply by creating extra data frames titled ”flows2” etc. The third data frame contains information for departments that are restricted from moving.

	Required Area	Free-shaped(0)/ Constraint(1)	AR restriction/ width	temperature requirement
1	2400	0	1.25	20
2	1600	0	1.33	20
3	3600	0	1.85	20
4	800	0	3.33	20
5	2100	0	1.18	20
6	1750	0	2	20
7	360	0	3.33	20
8	1540	0	1.67	20

Figure 7.1: The input excel sheet that is used for the 8 department problem.

Department	1	2	3	4	5	6	7	8
1	x	5	2	4	1	0	0	6
2	5	x	3	0	2	2	2	0
3	2	3	x	0	0	0	0	5
4	4	0	0	x	5	2	2	10
5	1	2	0	5	x	10	0	0
6	0	2	0	2	10	x	5	1
7	0	2	0	2	0	5	x	10
8	6	0	5	10	0	1	10	x

Figure 7.2: The flow matrix that is used for the 8 department problem

7.1.3. Generating the departments

Once the information is loaded, the departments can take shape. by attributing them with a radius, depending on their area. Due to the lowered importance of the exact department shapes (and the fact that they will remain static through the entire first-stage model), all departments, be it free-shaped or constraint, will be drawn as a just a circle. Hence, the departments are attributed a radius, depending on their area. Furthermore , the radius added to the data frame so they can be called in a later stage.

After awarding the departments their dimensions, they will be attributed with a random starting position that is within the plane. This is done by generating a random number twice for every department, that is within the range of the dimensions of the plane. These coordinates are also added to the data frame, so they can easily be called in a later stage. All information for every department is then transferred from this Cpython component into a GHpython component where the results can be visualised, as seen in figure 7.3 and figure 7.4 . Furthermore, the algorithm is provided in algorithm 2.

Algorithm 2: Pseudo-code for shaping the departments

```
make list of required areas by reading from dataframe;
radii = [];
for i in range(nr_departments) do
    radius = sqrt(Area / π)
    append radius to radii
end
x_coordinates = [];
y_coordinates = [];
for i in range(nr_departments) do
    append random integer (0, facility width) to x_coordinates;
    append random integer (0, facility height) to y_coordinates;
end
```

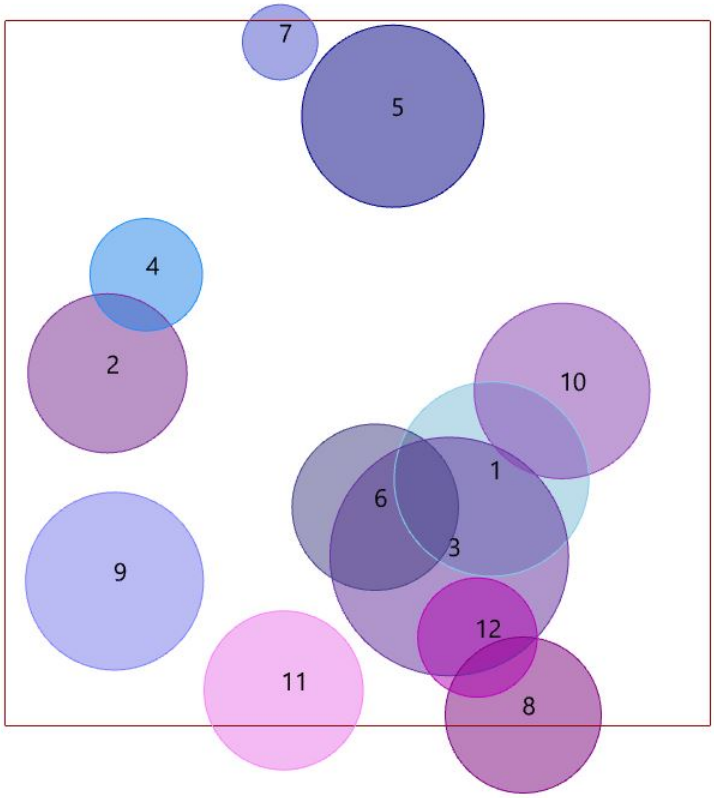


Figure 7.3: A problem with 12 departments that have been given dimensions and are randomly placed on the plane.

7.1.4. Adding vectors

With the departments now placed on the plane at random, the different vectors that will set them in motion can be computed. In total there are three different vectors that act upon a department, where the most important vector is the flow-cost vector that is computed using the gradient descent approach. This vector is, as explained in chapter 6, an attractive vector that is computed by taking the gradient of the objective. In this gradient, the variables are the x and y coordinates of the department in question, while all the other departments are considered to be constant, as seen in equation 6.14. The pseudo-code for this algorithm can be found in algorithm 3.

The second vector is the repelling vector for overlap, which is a repelling vector that works in the opposite direction of any possible overlap between departments. For every department, the amount of overlap is computed that exists between all other departments (which can also be zero). Then vectors are produced from the center of the department in question to the departments that it has overlap with multiplied by the amount of overlap. Then they are recomputed into one single vector by taking the average of the vectors. The algorithm for computing the overlap vectors can be found in algorithm 4.

The last vector is an attracting vector that is used to keep the departments inside the total facility. This is not absolutely necessary, however, without this vector, the departments sometimes move out of the focus of the capturing algorithm (resulting in images where half the departments are missing). This vector is computed by looking at each department and checking for overlap with the outside facility, if the facility overlaps, a vector is added attracting the department to the center of the total facility. The way the overlap is calculated is similar to the overlap vector.

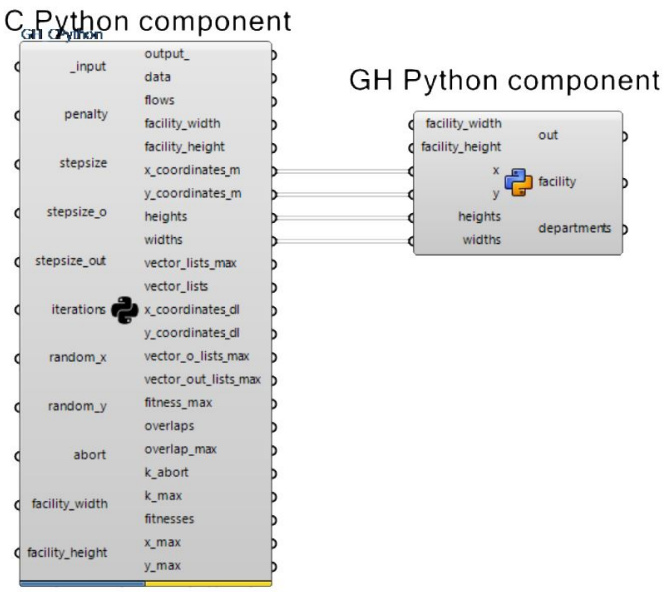


Figure 7.4: Two python component working together, the C python component handles the data and passes it to the GH python component for visualisation.

Algorithm 3: Pseudo-code for computing the flow attraction vectors

```
flow-vectors = [];
for i in range(nr_departments) do
    dfdx = 0 ;
    dfdy = 0 ;
    for j in range(nr_departments) do
        if i != j then
            compute d_ij
            if radi[i]+radi[j]-d_ij > 0 then
                pass ;
            end
            else
                Compute dfdx_addition ;
                Compute dfdy_addition ;
                dfdx = dfdx + dfdx_addition ;
                dfdy = dfdy + dfdy_addition ;
            end
        end
    end
    flow-vector = [dfdx,dfdy] ;
    append flow-vector to flow-vectors ;
end
```

Algorithm 4: Pseudo-code for computing the overlap repelling vectors

```
overlap_vectors = [];
for i in range(nr_departments) do
    overlap_vector = [0,0];
    for j in range(nr_departments) do
        vector = [ Δx_ij,Δy_ij] ;
        neutralise vector to eliminate distance influence ;
        Compute amount of overlap ;
        vector_add = vector *Overlap ;
        overlap_vector = overlap_vector + vector_add ;
    end
    append overlap_vector to overlap_vectors;
end
```

Once all three different vectors are computed for all departments, they are combined into one single master-vector. Note that each department has at least one vector (the flow-cost vector), but could have up to three. This master-vector is, again, generated using all vectors by giving them multiplications, these multiplications are variable and thus can be changed at any moment, however, the final multiplications are chosen as follows: 1.2 for both overlap vectors and 1 for the flow attracting vector. The overlap vector multiplications are chosen to be bigger than the flow-cost vector by a small amount to further discourage overlapping between the departments.

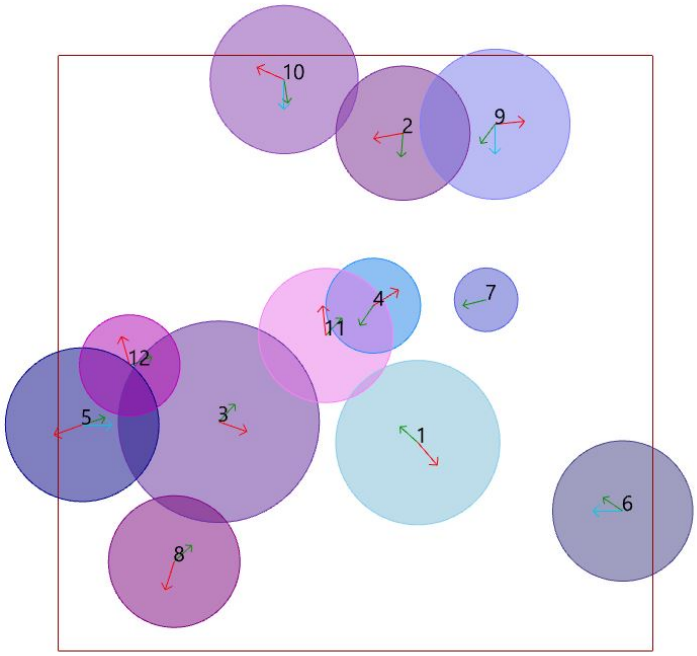


Figure 7.5: The visualisation of the computed vectors for every department. Green is the flow vector, red is the overlap between departments vector and blue is the overlap with the outside facility vector.

7.1.5. Moving the departments

With the computation of the master-vector for every department complete, the theoretical direction a department should move in order to get a better objective value has been realised. The next step would be to move the department in said direction. The question remains how much each department would have to move, since after each movement, the vectors would slightly change. This means that moving the departments too much would result in less accuracy, since a larger movement would alter the vectors too much and intermediate changes are disregarded. However, moving the departments too little to gain accuracy would in turn result in a very slow progress and heavy computation times.

As a bet that remains on the safe side, the chosen amount of movement was set to 1, as long as that number is in the order of 1/150th of width of the total facility size, meaning that in theory, a department could cross form one side to the other in 150 generations. With a running time of 800 generations, this would allow for enough time for the departments to find an equilibrium while still maintaining enough accuracy.

The moving of the departments is nothing more than updating their x and y coordinates in the data frame created at the start. The amount of movement in the direction of the master-vector is always 1, henceforth, the change of x and y coordinates will be equal to the derivative of that vector. The pseudo-code can be found in algorithm 7. A function is added that imports moving restrictions on certain departments provided by the third excel sheet. This function reads the departments that are bound to one location specifically and sets their starting position. During the movement of the departments these location restricted departments are excluded from moving. This locking of certain departments is an optional function that helps with location restrictions of certain rooms, like an office for example that needs to be on the outside perimeter of a factory.

Algorithm 5: Pseudo-code for moving the departments each iteration

```
for i in range(nr_departments) do
    if i not location constraint then
        recompute vectors to have the same amplification;
        add new amplification to vectors;
        master_vector = sum off all vectors;
        x_i = x_i + x'master_vector;
        y_i = y_i + y'master_vector;
    end
end
```

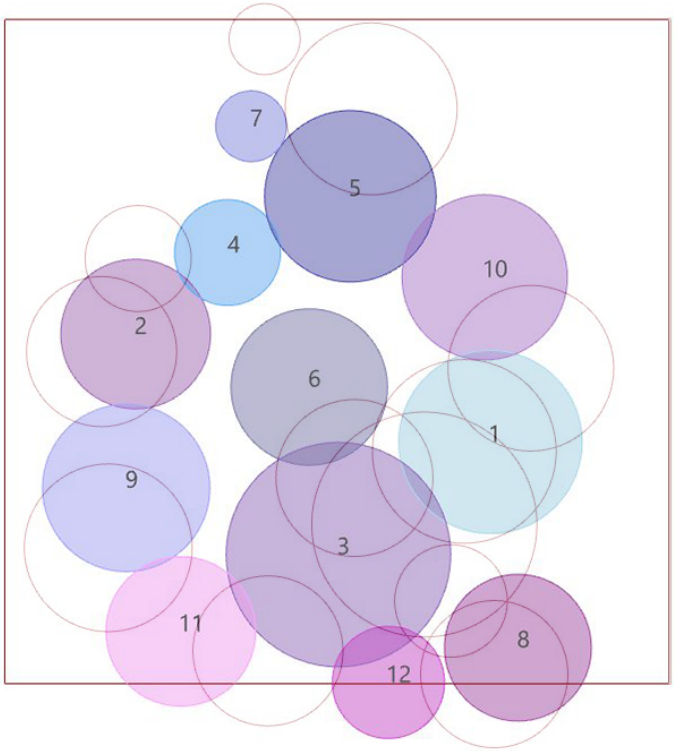


Figure 7.6: The movement for the department over 20 iterations visualised.

7.1.6. Computing the objective

After moving the departments in the direction of the master vector, the whole process starts again with computing the new vectors for each department. Using this iterative process, the departments should theoretically be moving towards an optimal objective. However, the monitoring of this theoretical improvement can only be tested and quantified if this objective is computed after every movement, so that the hypothesis can be tested.

The objective in this first model is formulated like in equation (6.10), where the flow-cost makes up a large portion. However, the overlap is also integrated into the objective by increasing the distance whenever overlap is present as explained in chapter 6 and seen in equations (6.11), (6.5) and (6.6)

In order to monitor just how well this implementation of the overlap is working, in addition to the objective, the flow-cost (the objective excluding the overlap) and amount of overlap (in m2) will be monitored as well. In order to do this, all information is stored in a list inside the program. Once the program terminates, a few grasshopper nodes handle the list to show in a graph the progression (or regression) of the objective, flow-cost and overlap. Figure 7.7 shows an example of a graph depicting the change of the objective, flow-cost and overlap through the 800 iterations. As visible, the objective decreases slightly with higher overlaps, but not as much as the flow-cost decreases. Also, the information inside the lists is used to create a Pareto diagram, showing flow-cost on the y-axis and overlap on the x-axis. This allows a human observer to quickly get a feeling of the best solutions. The best solution found by the computer is highlighted as well, which can be seen in figure 7.8. These graphs give the observer insight in how well the program is performing.

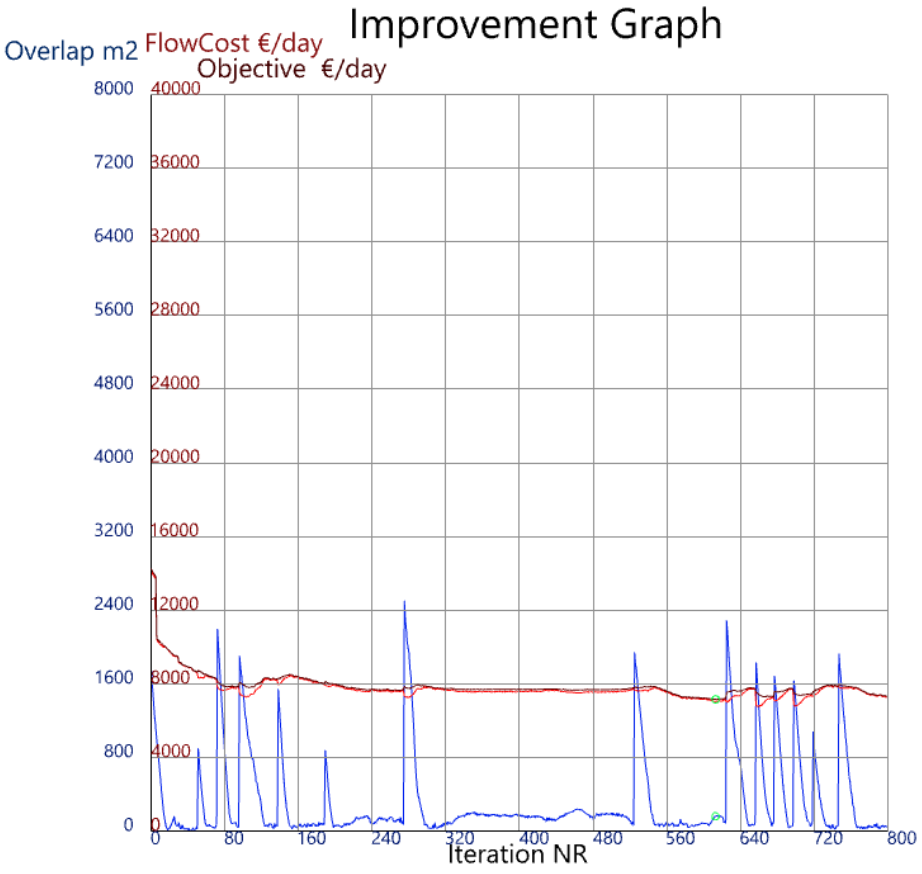


Figure 7.7: An example of a graph monitoring the improvement of the objective, flow-cost and overlap through the iterations. When the overlap increases, the objective decreases, but not nearly as much as the flow-cost.

After every iteration, the objective is calculated. If the objective is the minimum of all found objectives thus far, it will be stored, along with all information on the current state of the departments, such as: x and y coordinates, the amount of overlap and the iteration number. If the objective is improved in future iterations, this information is overwritten. At the termination of the program, the information for the best

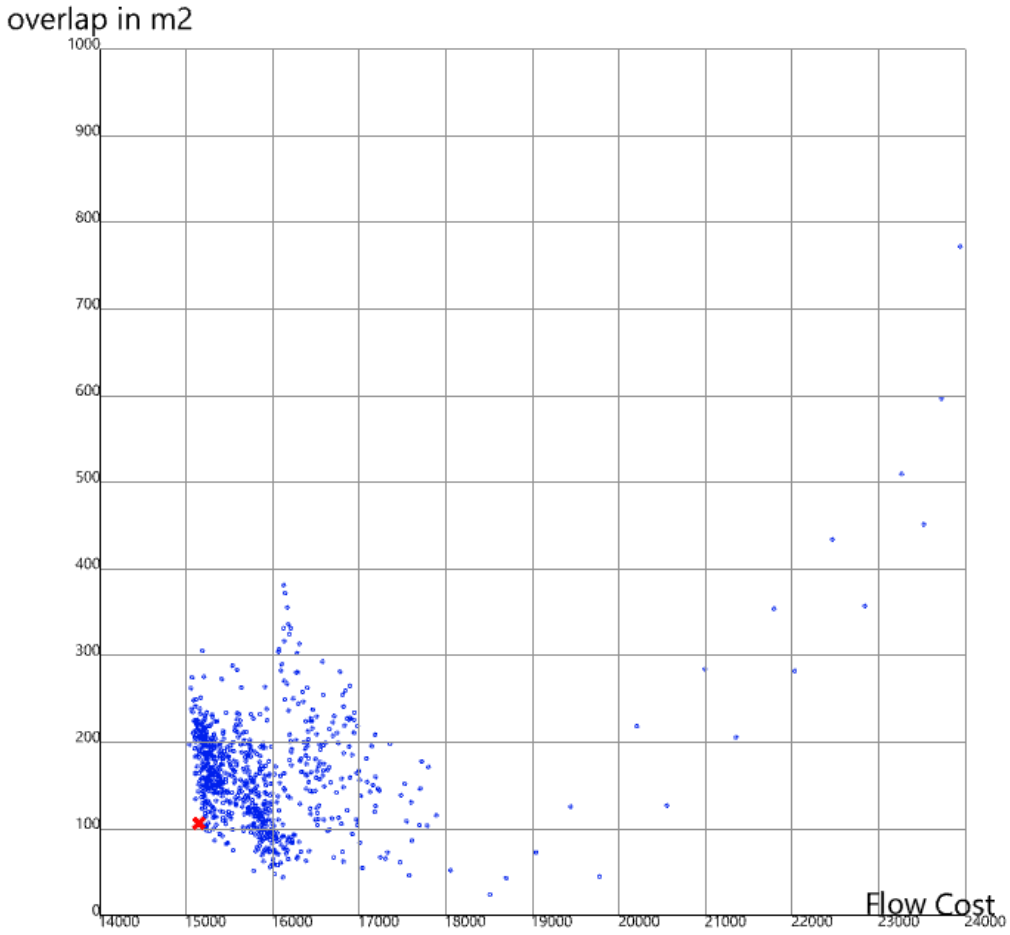


Figure 7.8: An example of a Pareto diagram of flow cost and overlap. All solutions are represented by a dot, the solution chosen by the algorithm is highlighted in red.

found objective will be available for the other main python node to reproduce the departments with, as seen in figure 7.4. This way, once the program terminates, the best solution/iteration will be presented.

7.1.7. Assessment

This subsection is about the way the method was assessed on whether it is doing the things right and on whether it is doing the right things as discussed in section 2.2.

The first part, the evaluation of whether the program is doing the right things, is conducted in small tests to verify, with known methods that are know to work, the results presented by the program. An example of this is the verification of the amount of overlap for the presented best solution matches the actual overlap for the departments generated by the GH python component. This verification is done by using grasshopper nodes to determine the amount of overlap (which is a method known to be working), as seen in figure 7.9.

Yet another example is the trick for verifying the flow vector to be correct, where one of the departments in a toy problem of 10 departments (5), has zero flow to other departments except to department 9. This means that the vector for flow attraction for department 5 should always cross the center of department 9, as is seen in figure 7.10. All these examples, together with more tests, have proven that the method is, in fact, doing the right things.

The second part, the evaluation of whether the program is doing the things right, is tested by taking toy-problems from the literature and comparing the results that are found by this method with the results of other methods in the literature. Sometimes, this comparison requires that the objective is rewritten to match the objective of the paper in question. There is two ways to go about this, reconstructing the objective to match the other, or optimise the toy problem using your own objective and then recomputing

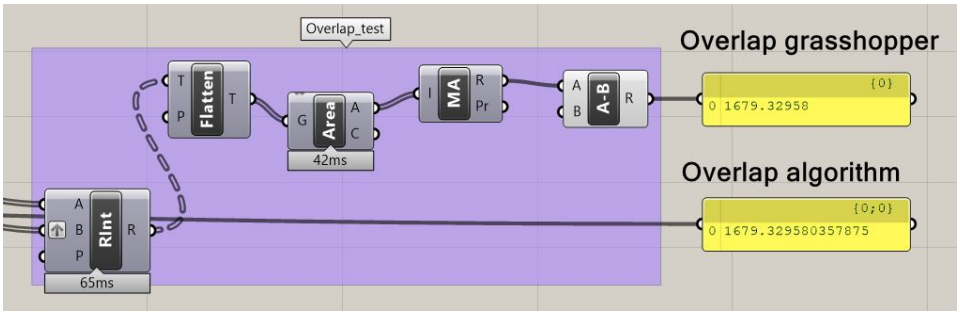


Figure 7.9: A test to verify the amount of overlap calculated by the algorithm is in line with the amount of overlap given by grasshopper nodes.

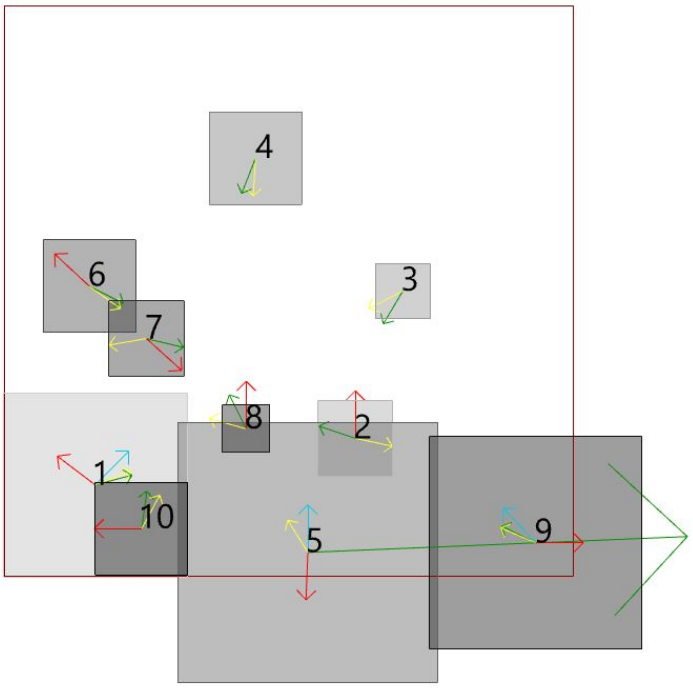


Figure 7.10: A test to verify that department 5's flow vector (which only has flow to department 9) crosses the center of department 9.

the objective one time for the best result found.

Since the results of the first-stage model are still very abstract it is hard to compare the results of the first-stage model with other results in literature because the difference in constraints would be large. In order to still test whether the first-stage model is working well, its influence on the second stage model should be tested. Example given: the second-stage model can be tested twice, once with the constraints from the first-stage model and once without.

7.1.8. Improvement

Shooting

In addition to the gradient descent approach, two procedures are proposed to solve certain problems that a gradient descent approach encounters. The first problem is that the the departments are seeking equilibrium, but sometimes this is a local minimum equilibrium. The downside of the equilibrium is that there is no natural way for the gradient descent approach to escape this local equilibrium once trapped. In order to counter this, the shooting procedure is introduced. This procedure initiates when there has been no progress in the objective for n iterations, after which the next iterations multiplies the flow attraction vector by a factor α. During this shooting iteration, all overlap vectors are ignored.

This procedure introduces an imbalance in the equilibrium so that the local minimum can be es-

caped. Once the shooting procedure has initiated, a second one can not be initiated for n iterations. Furthermore, during this cool-down period, the objective is stored but the solution does not qualify as valid until the cool down is reduced to 0 again. The procedure for shooting is summarized in algorithm 6.

Swapping
The second problem that occurs is the influence of the random starting positions. While the shooting procedure will already partly eliminate this problem, a swap procedure is also proposed. When triggered, the swap procedure evaluates all possible department swaps and ranks them based on their influence on the objective. It is important to notice that the overlap is not included in the objective when evaluating a swap, since this would decourage swapping. When all possible swaps have been evaluated, the most profitable one (if any) is performed. The trigger for swapping is a rapid increase in the objective, where the current iteration must have improved by at least β % in comparison to objective of the previous iteration. Just like with shooting a cool-down of n iterations is issued where neither a swapping of shooting procedure maybe initiated. The procedure is summarized in algorithm 7 and visualized in figure 7.11.

Algorithm 6: Pseudo-code for the ”shooting” procedure

```
if cool-down != 0 then
    cool-down = cool-down -1
end
if iteration > n AND cool-down == 0 then
    if objective_current >= objective_iteration-n then
        shoot = True
    end
end
if shoot == True then
    Compute only the flow-attraction vectors
    Move all departments α * the set distance
    cool-down = n
    shooting = False
end
else
    Move departments normally
end
```

Algorithm 7: Pseudo-code for the ”swapping” procedure

```
if cool-down == 0 then
    if objective_current/objective_previous ≤ β then
        swapping = True
    end
end
if swapping == True then
    flag = False for i in range(nr_departments) do
        for j in range(nr_departments) do
            if i !j then
                copy the data frame
                swap coordinates of department i and j
                calculate the objective excluding overlap
                if objective < min(objectives) then
                    save i
                    save j
                    flag = True
                end
            end
        end
    end
    if flag == True then
        swap departments i and j in actual data frame
        cool-down = n
    end
    swapping = False
end
```

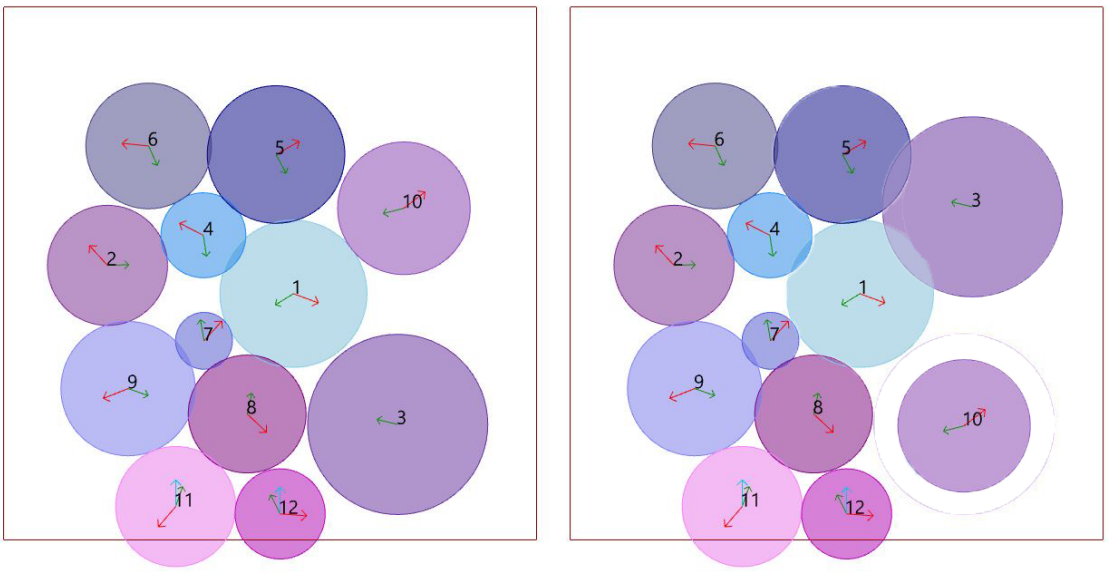


Figure 7.11: The improvement method of ”swapping” visualized. In this case departments 3 and 10 get swapped from position at iteration 140.

7.2. second-stage model

The second-stage model takes the best solution found by the first-stage model and uses its information to implement constraints in a third-party solver to tremendously narrow down the solution space.

The second-stage model will take the shape of a third party solver CP-SAT from OR-tools [36], where the objective and the constraints as described in chapter 6 will be modelled. The solver uses the width, heights and the coordinates of each departments as variables, which is quite a lot of variables. However due to implementing the relational constraints found in the previous model, the hypothesis is that this will help the solver converge to a good solution fast. In order to prove this, two models can be made, where one does not implement the relational constraints found in the first-stage model, and the other one does.

7.2.1. Extracting relative location constraints

After the first-stage model finds a solution that is satisfying, that particular solution can be analysed in order to get the relative location constraints, as described in section 6.1.4, that should be implemented in this second-stage model. Theoretically, implementing these constraints in a third-party solver, should enable the solver to more quickly find a more optimal solution. This is because the solution space the solver has to search gets drastically lowered when $\frac{(n-1)}{2}n$ [46] unique constraints are added to the model. This way the solver is already steered in, according to the first model, the right direction.

Producing these relational constraints at the final solution of the first-stage model is rather easy, the only thing needed are all the departments x and y coordinates. From here, a data frame can be produced that represents a relational constraint matrix, filled with 0's, 1's, 2's and 3's. Each number represents a constraint for the department in the row of the matrix in relation to the department in that column of the matrix, where a 0 represents the row department should be positioned to the left of the column department, 1 represents that it should be located above, 2 represents that it should be located to the right and a 3 represents that it should be located below the other department.

These numbers are filled in based on the amount of horizontal and vertical distance between each set of department (and thus for each slot in the matrix), should the vertical distance be a factor 1.5 higher than the horizontal distance, the number that is filled in will either be a 1 or a 3, based on the relative position of the departments. If the horizontal distance is a factor 1.5 higher on the other hand, a 0 or 2 will be attributed to the matrix. In the case that both values lie within a factor 1.5 of each other, no constraint is added. This possible outcome of no constraint is implemented because if neither distance is significantly larger, the chance on implementing a constraint that is only limiting increases. This procedure's pseudo-code can be found in algorithm 8.

Algorithm 8: Pseudo-code for transcribing the relational constraints into a data frame, which can then be written to an excel file.

```
initiate data_frame with size i by j ;
for i in x_coordinates do
    for j in y_coordinates do
        if i!=j then
            calculate δx ;
            calculate δy ;
            if δx>δy*1.5 then
                if x_i< x_j then
                    set data_frame at row i and column j to 0 ;
                end
                else
                    set data_frame at row i and column j to 2 ;
                end
            end
            if δy>δx*1.5 then
                if y_i< y_j then
                    set data_frame at row i and column j to 3 ;
                end
                else
                    set data_frame at row i and column j to 1 ;
                end
            end
            else
                set data_frame at row i and column j to NaN ;
            end
        end
        else
            set data_frame at row i and column j to NaN ;
        end
    end
end
```

7.2.2. Implementing the constraints

Google's OR-tools [36] offers a number of solvers. All of these solvers have its limitations but also advantages, and the solver that fits best with the problem should be chosen. Sadly, none of the solvers offered an environment that could easily work with floats and non-linear constraints.

The chosen solver is the CP-SAT solver [36], a constraint programming solver that also offers the possibility of implementing an objective. This solver was chosen over the GLOP linear programming solver [36] because the linear programming solver offered no tools to somehow implement non-linear constraints, while the CP-SAT solver did.

The variables are modelled first. These are the widths and heights of all departments and their x and y coordinates. Immediately there is one big constraint that has to do with the widths and heights, being that the multiplication of these variables should be equal to the area requirement, as seen in equation (6.2). However, a multiplication of two (or more) variables is non-linear so this constraint can not be implemented regularly. The CP-SAT solver offers a way that still allows this multiplication of different variables through the .AddMultiplicationEquality method. Together with the fact that only integers are allowed, and the constraint can only be set to equality (no bigger then or less than), the number of options that are left to shape a department are low, since the area requirement only has so many ways to be a product of two integers. A possible way to work around this is the up-scaling of the variables and the area requirements, every factor 10 will add an extra decimal to the options of forming the area requirement. A nice example of this up-scaling is shown in table 7.1.

The no overlap constraints are added smoothly, as the inclusion of the boolean value λ_{ij}, as seen in equations (6.7), (6.8) and (6.9), removes the non-linearity of the absolute value between the coordi-

Area Requirement	Nr of possible multiplications
12	3
1200	15
120000	35
12000000	63
56	4
5600	18
560000	40
56000000	70

Table 7.1: The amount of integer multiplications to make the area requirement, up-scaling by factors of 100 offers more possibilities, but this comes at a cost of computational power required as the variables will be scaled as well (by factors 10).

nates. This principle works because the no overlap constraint is of the form $-|f(x)| \leqslant -z$, but this can be rewritten to $-f(x) \leqslant -z$ when $f(x)$ = positive and $f(x) \leqslant -z$ when $f(x)$ is negative [15].

The aspect ratio constraints, as seen in equation (6.1) , are implemented right at the implementation of the variables for the width and height, where minimum and maximum bounds can be specified. These bounds are calculated in a separate python component and transferred to the main python component.This separate python component imports the area and aspect ratio constraints for every department, as well as the boolean for whether the department is free shaped or constraint, and it then translates these in a minimum width (which also is the minimum height). The pseudo-code can be found in algorithm 9. If the department is of the constraint type, an additional constraint is added that sets the height or width of the department to one specific value.

Algorithm 9: Pseudo-code for finding the minimum width or height for each department, depending on their area and aspect ratio limitations.

```
minimum_dimensions = [] ;
for i in range(nr_departments) do
    loop = True ;
    if department == free-shaped then
        j = 1 ;
        while loop == True do
            if area_i/j < j * max_aspect_ratio then
                append j to minimum_dimensions ;
                loop = False ;
            end
            else
                j = j + 1
            end
        end
    end
    else
        append max_aspect_ratio to minimum_dimensions
    end
end
```

8

Results

In this chapter, the results for both models will be discussed based on the objectives, but also on computational time. The impact of the swapping and shooting procedure, as well as the them combined can be set out, comparing them and discussing the impact. Furthermore, the consistency of the results will also be discussed. The conclusions of the results can be found in chapter 9. The proposed framework was tested using a python 3.8 environment set up in Rhinoceros [3] using the plugin grasshopper [2] for the first-stage model and the CP-Sat solver from google OR-tools [36] modelled in python for the second-stage model. The tests were done using an Intel(R) Core(TM) i7-9700K CPU @3.6GHz with 32 Gb of memory. As test problems, two cases were taken from [44] and [8] which were originally composed in [34] consisting of 8 and 12 departments. Both the results for the first- and second-stage model are presented.

8.1. First-stage model

The results for the first model are published in table 8.1. The table holds the results that were taken for a 50 sample test on the four different methods, vanilla, swapping, shooting and swapping and shooting combined. Each method had its own sample of 50 runs, taking a maximum of 800 iterations.

Each column provided the results for one of these methods, going over the objective as well as the flow-cost, the amount of overlap and the number of iterations it took to find the best result. Each subsection is split into the average, minimum, maximum and standard deviation. Table 8.1 shows the results for the 8 department test, only the 8 department problem was used to test the different methods efficiency. Additionally, the relative improvement of the proposed improvement procedures compared to the vanilla method can be found in table 8.2.

The computational time needed for the test problems were on average 20.1 seconds for the 8 department problem and on average 29.8 seconds for the 12 department problem for 800 iterations. Figures 8.1 through 8.4 show the best layout for the first-stage model for respectively the 8 and 12 department problems, as well as their graphs. The graphs show the improvement of the objective, flow-cost, and overlap through all 800 iterations. The green circle encircles the best iteration (corresponding with the layout). Extra layouts and graphs for the vanilla method, the swapping, shooting, and both methods combined can additionally be found in appendixes C through F.

8.1.1. Computational time

Table 8.3 shows the time required to run the toy problem with 10 departments at different max iterations, where there is a linear relationship between the two columns. Because most operations have two for loops over the amount of departments, it is estimated that when the number of departments is increased, the computational time required to run one solution will increase exponentially by a factor to the power two. Table 8.4 shows the estimated computation times for other size problems at 800 maximum iterations.

	Vanilla	Swapping	Shooting	Both
Objective				
Average	905.8	766.4	864.7	767.6
Minimum	760.6	745.1	747.9	742.5
Maximum	1225.7	829.6	1207.9	977.9
Standard deviation	96.7	16.9	99.2	33.8
Overlap				
Average	4.9	1.47	3.14	2.04
Minimum	0.46	0.62	0.7	0.61
Maximum	146.41	3.34	40.7	5.73
Standard deviation	20.46	0.77	5.63	1.16
Flow-cost				
Average	894.1	757.3	854.6	755.8
Minimum	568	727.7	741	723.1
Maximum	1220.5	816.8	1200.8	971.4
Standard deviation	104.5	17.5	99.9	35
Iterations				
Average	475.8	521.9	613.1	518.8
Minimum	0	141	8	118
Maximum	799	799	796	790
Standard deviation	238.3	197.1	181.2	169.1

Table 8.1: The results for the 8 department problem for the first-stage model.

	Swapping	Shooting	Both
Objective			
Average	-15%	-5%	-15%
Minimum	-2%	-2%	-2%
Maximum	-32%	-1%	-20%
Standard deviation	-83%	3%	-65%
Overlap			
Average	-70%	-36%	-58%
Minimum	35%	52%	33%
Maximum	-98%	-72%	-96%
Standard deviation	-96%	-72%	-94%
Flow-cost			
Average	-15%	-4%	-15%
Minimum	28%	30%	27%
Maximum	-33%	-2%	-20%
Standard deviation	-83%	-4%	-67%
Iterations			
Average	10%	29%	9%
Minimum	-	-	-
Maximum	0%	0%	-1%
Standard deviation	-17%	-24%	-29%

Table 8.2: The improvement in percentages for the proposed improvement methods compared to the vanilla method for the 8 department problem, negative is a decrease (improvement).

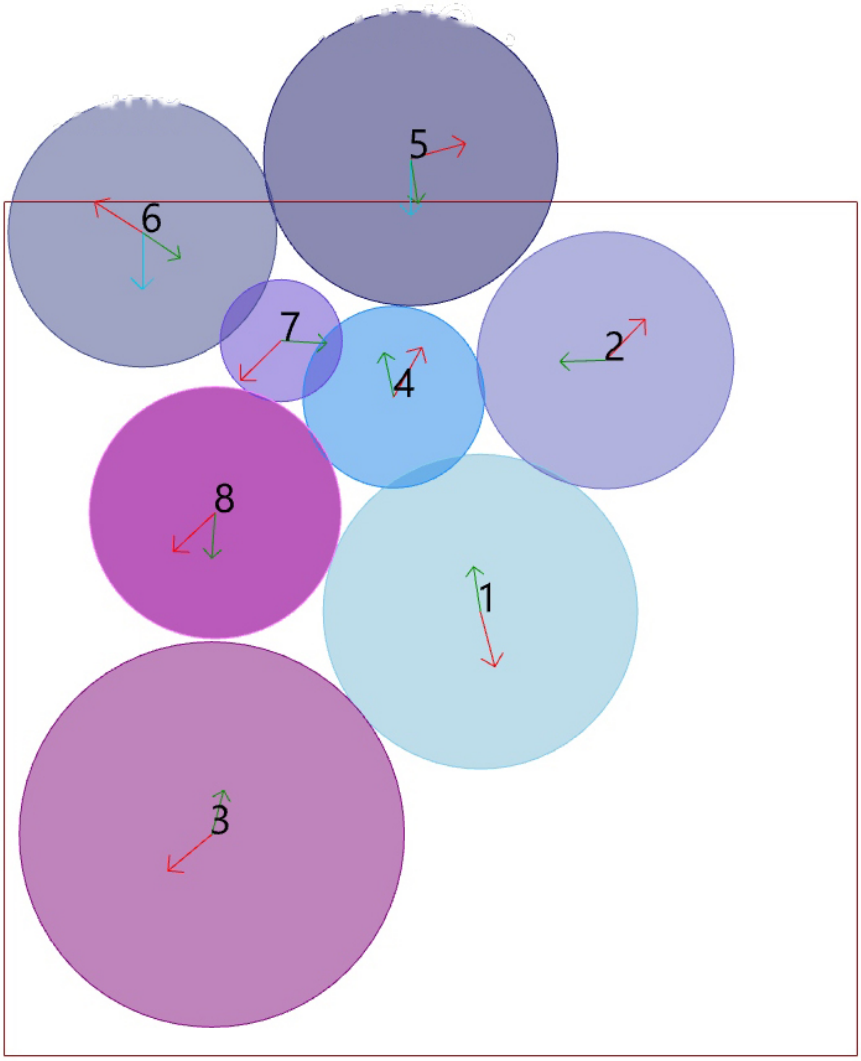


Figure 8.1: The result for the first-stage model considering the layout for the 8 department problem.

Number of iterations	Computational time
200	~8 s
400	~15 s
800	~28 s
1600	~55 s
3200	~108 s

Table 8.3: The computational time needed to cover a varying number of iterations for a problem with 10 departments, notice that there is a linear relationship.

Number of departments	Estimated computational time
5	~7 s
10	~28 s
20	~104 s
50	~700 s
100	~2800 s

Table 8.4: The computational time needed to cover 800 iteration with a varying number of problem sizes , notice that there is a quadratic relationship relationship.

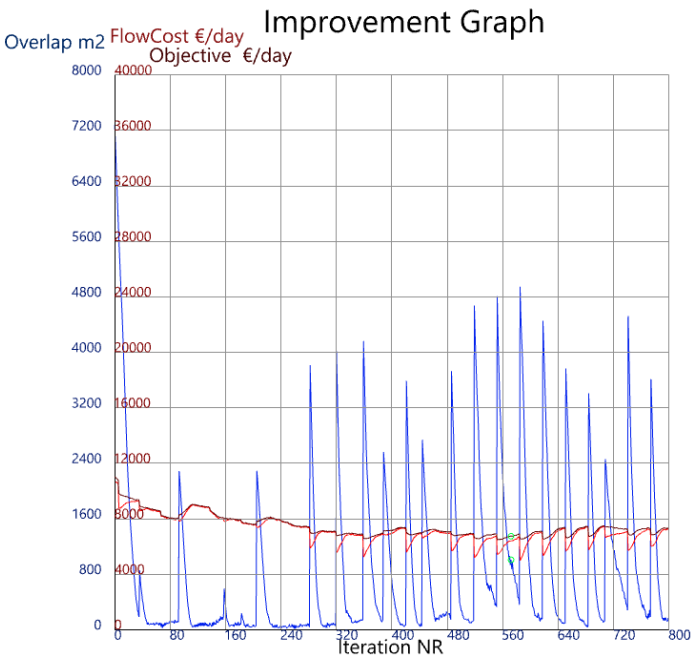


Figure 8.2: The change of objective, flow-cost and overlap through the iterations concerning the best solution for the 8 department problem.

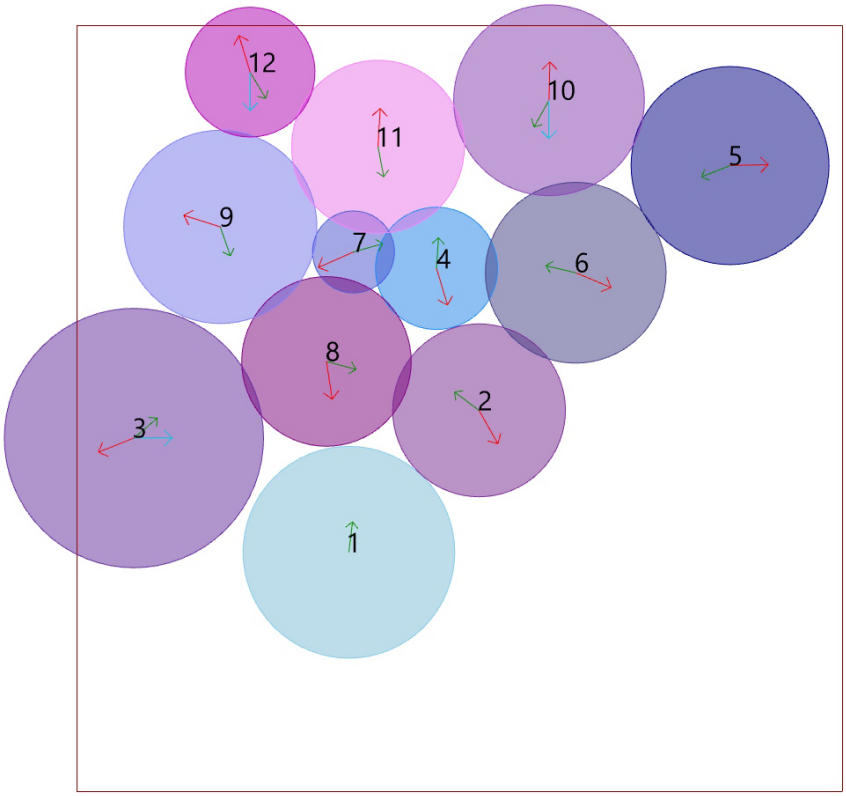


Figure 8.3: The result for the first-stage model considering the layout for the 12 department problem.

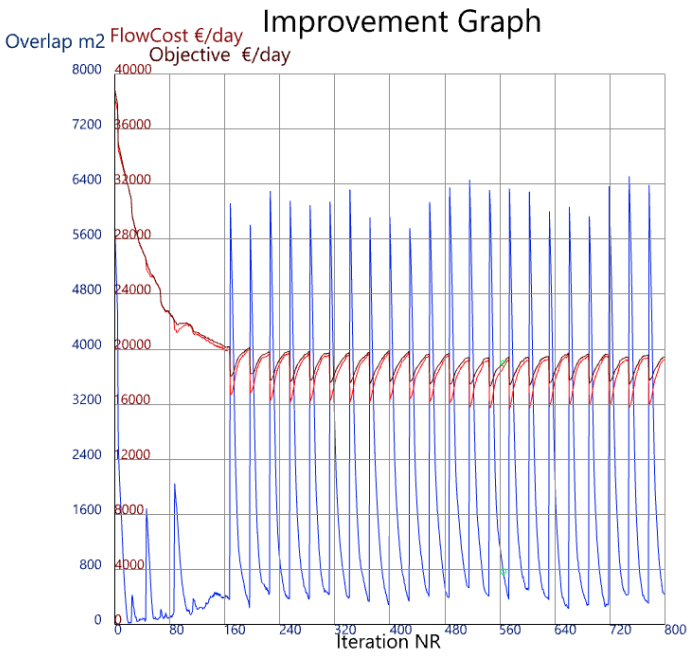


Figure 8.4: The change of objective, flow-cost and overlap through the iterations concerning the best solution for the 12 department problem.

8.2. Second-stage model

In the second-stage model, the best solution found during the first-stage model for the 8 and 12 department problem, as seen in figures 8.1 and 8.3, are used to find the relational constraints. These constraints, together with all constraints established in section 2, will be modelled in the CP-Sat solver and solved for three time frames: 25 seconds, 100 second and 300 seconds. Additionally, the same models are solved but this time the relational constraints from the first model are excluded in order to test the effectiveness of the first model. In order to gain more accuracy, a scale factor of 100 is used to upscale the problem, as explained in section 3. The total facility size is taken to be 13x13 for the 8 department problem (or 1300x1300 when the scale factor is applied) and 16x16 for the 12 department problem (or 1600x1600 when the scale factor is applied).

The results for the 8 department problem, taking into account 3 different time frames and the in- or exclusion of the first-stage model's constraints are published in table 8.5, while the same results for the 12 department problem are published in table 8.6. Additionally, the best results are published in table 8.7, which, in addition to providing information on the objective as stated in this paper, compares the objective with up to two references who have solved the exact same problem. In order to be able to compare the objective, the final layout was found using the objective in this thesis, but an additional python component would calculate the objective for that solution as formulated in Tam and Li [44], thus both objectives comparable.

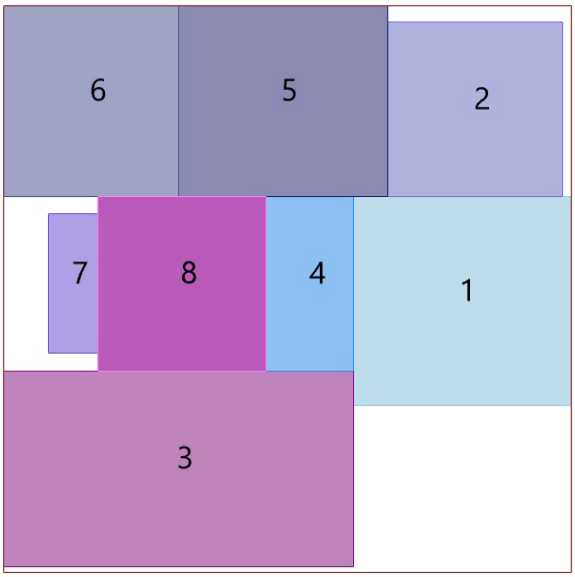


Figure 8.5: The result for the second-stage model considering the layout for the 8 department problem.

	Relational constraints	No relational constraints	Difference%
25s	477.2	528.1	9.6
100s	427.6	520.3	17.8
300s	410.3	515	20.3

Table 8.5: The results for the 8 department problem including the first-stage model's constraints and excluding them.

	Relational constraints	No relational constraints	Difference%
25s	1338.9	1678.4	20.2
100s	1302.1	1607.1	19.0
300s	1302.1	1444.0	9.8

Table 8.6: The results for the 12 department problem including the first-stage model's constraints and excluding them.

Departments	OV	Recomputed OV	Tam OV	Chwif OV
8	410.3	967.3	839	-
12	1302.1	3931.4	3162	3684

Table 8.7: The two problems compared to methods from Chwif et al. [8] and Tam and Li [44]

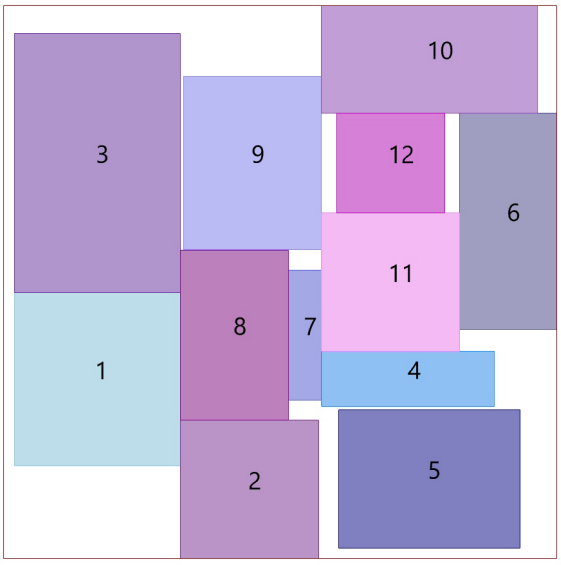


Figure 8.6: The result for the second-stage model considering the layout for the 12 department problem.

9

Conclusion

This chapter presents the conclusions of the results that were found in the previous chapter and also answer the main research question:

> ”How to computationally generate a layout of a vegetable processing factory given a program of requirements and flows between facilities as a matrix using a mathematical approach, minimizing the travel distance of goods needed for a product to be manufactured?”

In this thesis, knowledge was gained through a literature study to evaluate previous solutions other authors might have had for the FLP. This, together with a study on the requirements and criteria for the specific FLP for a vegetable food factory led to a number of criteria that the previously found methods could be assessed on. Based on these conclusions, a two-stage model was chosen as a method, where the first-stage model is a gradient descent approach model and the second-stage model is a constraint programming solver. The two models work together in the way that the first model finds relative location constraints for the second model.

The first-stage model has four versions: one initial version and three attempts to improve upon this initial version. When looking at the results found in section 8.1, a number of conclusions can be drawn. It can be noted that the results for the vanilla method are the least promising, with the highest average objective, overlap and high standard deviations, indicating a heavy influence of the random starting positions. This influence is slightly reduced for the shooting method, as well as the average objective. However, the real improvements can be found with the swapping method and the method that includes both swapping and shooting, greatly reducing the average objective and the standard deviation. While swapping has the best average objective, the method with both procedures has the absolute single best solution, which is due to the fact that it has a slightly higher standard deviation, increasing the chances to get a very low objective (and also a very high objective) while still, on average, producing very solid solutions. Therefor, it can be argued that the method that includes both procedures is preferred when there is enough time to run multiple tests while the swapping method would be preferred when only a few samples can be run.

Computationally, all methods finish within a reasonable time of below 30 seconds for a problem size of 12 departments iterating over 800 iterations. This computational time gets higher as the amount of iterations increases, however, testing has shown that the best results are rarely found beyond the 800th iteration. Even if a better solution is found, it is a very small improvement percentage wise. A bigger problem is increasing the number of departments, where the computational time required gets exponentially higher. Finding 100 different solutions for a problem size of 50 departments would take 70000 seconds, or almost 19.5 hours, an absolute maximum. This concludes that the chosen method is not feasible for solutions with more departments than 50, and hardly feasible at 50 departments, until improvements are made to make the model more streamlined and therefor faster.

When comparing the inclusion of the first-stage model’s constraints versus the exclusion of those constraints in tables 8.5 and 8.6 , it can be said that the inclusion of the constraints does help significantly to lower the objective as the improvement is always at least 9.6% in all time frames, as well for the 8 department problem as for the 12 department problem. When comparing the best results of the

method with the results found in the literature however, it can be seen that the approach in this paper is still behind the best approaches found in the literature. As Chwif et al. [8] already noted, the method used in Tam and Li [44] is an exact method, so the results there are an absolute optimum, but that method becomes unfeasible after 15 methods, while the method in this paper does not. In addition to that, the empty space ratio (ESR) was better in this case, 20% versus 37% and 23%, proving that this method creates a more compact layout.

An interesting solution can be drawn when looking at the layouts for the first- and second-stage model side by side for both toy problems, as presented here in figures 9.1 and 9.2. For the 8 department problem, it is clearly visible how the first-stage model is translated into the second-stage model, where all departments stay more or less in the same spot. For the 12 department problem, however, this translation is seemingly less apparent. Still, the results for the 12 department problem are promising, even though the positions from the first-stage model seem not to be copied exactly. This is a result of the 1.5 factor when deciding to add a relative location constraint between two department, where if the vertical difference and horizontal difference don't show a significant difference, no relative location constraint is added, as explained in section 6.1.4.

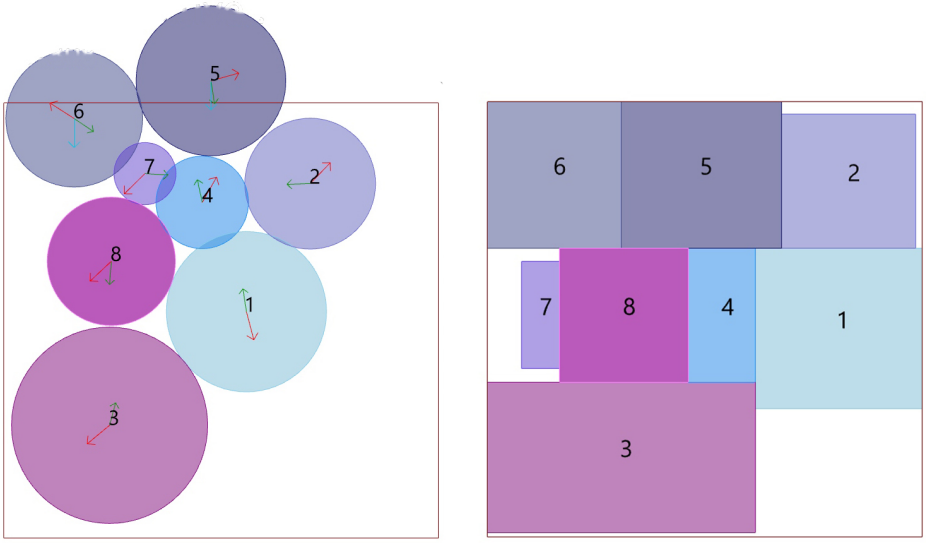


Figure 9.1: The comparison between the results of the first- (left) and second-stage model (right) for the 8 department problem.

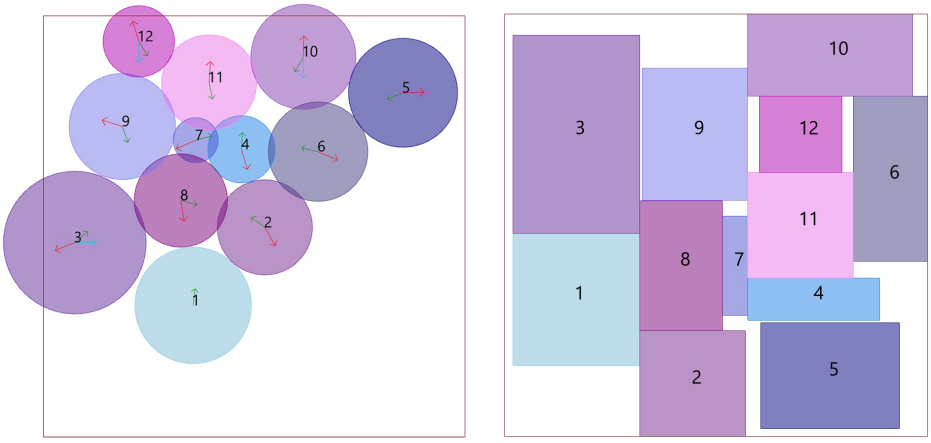


Figure 9.2: The comparison between the results of the first- (left) and second-stage model (right) for the 12 department problem.

When looking directly at the results, the proposed method is consistent and shows good results. Especially the inclusion of the first-stage model helps finding better solutions. Compared to other results found in the literature, the method is performing slightly less based purely on the objective value. What

it is capable of doing a little better however, is the reduction of the empty space ratio. Additionally, the proposed method is versatile enough and quite fast compared to other methods.

Would there have been space for additional research, other test problems of different sizes could have been explored more thoroughly. Additionally, a real-life example could have been used to test the proposed method on. An idea for generating even more promising lay-outs is experimenting with the way that the relational constraints are transferred from the first-stage model to the second-stage model.

The answer to the main research question is provided by this thesis: A layout for a factory can be sufficiently be generated using the presented approach, minimizing the travel distance of goods needed for a product to be manufactured. Additionally, other flows can be taken into account as well, but the material flow will in most cases be the dominant flow. Granted, this method is not the only approach that would answer the main research question, as proven in chapter 3, where numerous previous approaches have been discussed. This method does, however, make a valuable contribution to the collective knowledge of solving the FLP.

Limitations

The conclusion of this thesis leaves the thought that it was a big success. In the contrary, there is a number of things that could be improved and/or implemented differently. First of all, the core of this thesis, the link between the first- and the second-stage model could be questioned. For example, is the current way also the best way to extract the relative location constraints from the first-stage model? Would it work more efficiently if the constraints were selected using another rule? For example, instead of excluding some constraints, all constraints could be extracted and applied, maybe even in two direction instead of just one of the two. Another possibility is that only constraints are applied to sets of departments that have a flow between them that is not zero. Yet another idea is the selection of constraints only for sets of departments that are in close proximity.

Even though the implementation on the two different presented methods lead to the conclusion that the method is performing satisfactory, would this also have been the case on other problems? For example, problems that are of bigger size, or have more restrictions added to them. Another interesting test would be to implement the method on a real life factory that has, in fact, already been build. The objective of the layout of such a factory could be computed, and compared to the best result the method gives.

Appendix A addresses the problems that would arise when the temperature requirements of the departments would be included in the objective. Finding a way to academically include them using the models proposed in this thesis won't be possible since the second-stage model isn't cut out for it. In what ways would the second method need to change in order to overcome this? During the development of this thesis, attempts were made to implement the temperature requirements, but they were heavily underestimated from the beginning, leaving the question whether other parts of the thesis have not been underestimated.

Furthermore, the computational time of the method is quite fast, but it is not optimized. Improvements can and must be made in order to get the method smooth enough for problems that have a higher number of departments and thus require more computational power. A last remark on the limitations is that the method lacks an easy way of understanding so that a third party person would be able to operate it. To pick up on this, the implementation of a feedback loop between user and machine in order to eliminate some solutions or implement extra personal constraints would be interesting.

Bibliography

[1] Miguel F. Anjos and Manuel V.C. Vieira. An improved two-stage optimization-based framework for unequal-areas facility layout. *Optimization Letters*, 10(7):1379–1392, 2016. ISSN 18624480. doi: 10.1007/s11590-016-1008-6.

[2] Robert McNeel & Associates. Grashopper, . URL https://www.rhino3d.com/6/new/grasshopper.

[3] Robert McNeel & Associates. Rhinoceros 3d, . URL https://www.rhino3d.com.

[4] Farhad Azadivar and John Wang. Facility layout optimization using simulation and genetic algorithms. *International Journal of Production Research*, 38(17):4369–4383, 2000. ISSN 1366588X. doi: 10.1080/00207540050205154.

[5] Barry W. Boehm, editor. *Software Risk Management*. IEEE Press, 1989. ISBN 0818689064.

[6] Rainer Ernst Burkard. *Locations with spatial interactions: The quadratic assignment problem*, pages 387–437. Academic Press, United States, 1989.

[7] W. C. Chiang and P. Kouvelis. An improved tabu search heuristic for solving facility layout design problems. *International Journal of Production Research*, 34(9):2565–2585, 1996. ISSN 1366588X. doi: 10.1080/00207549608905045.

[8] Leonardo Chwif, Marcos Ribeiro Pereira Barretto, and Lucas Antonio Moscato. A solution to the facility layout problem using simulated annealing. *Computers in Industry*, 36(1-2):125–132, 1998. ISSN 01663615. doi: 10.1016/S0166-3615(97)00106-1.

[9] P M I Standards Committee. and Project Management Institute. *A Guide to the project management body of knowledge LK - https://tue.on.worldcat.org/oclc/33207934*. Project Management Institute, Upper Darby, PA SE - viii, 176 pages : illustrations ; 28 cm, 1996. ISBN 1880410125 9781880410127 1880410133 9781880410134.

[10] S. K. Das. A facility layout method for flexible manufacturing systems. *International Journal of Production Research*, 31(2):279–297, 1993. ISSN 1366588X. doi: 10.1080/00207549308956725.

[11] Amine Drira, Henri Pierreval, and Sonia Hajri-Gabouj. Facility layout problems: A survey. *Annual Reviews in Control*, 31(2):255–267, 2007. ISSN 13675788. doi: 10.1016/j.arcontrol.2007.04.001.

[12] Thomas Dunker, Günter Radons, and Engelbert Westkämper. Combining evolutionary computation and dynamic programming for solving a dynamic facility layout problem. *European Journal of Operational Research*, 165(1):55–69, 2005. ISSN 03772217. doi: 10.1016/j.ejor.2003.01.002.

[13] Welle Flager, Soremekun Bansal, F Haymaker ; Flager, B Welle, P Bansal, G Soremekun, and J Haymaker. Multidisciplinary process integration and design optimization of a classroom building. Technical report, 2009. URL http://www.itcon.org/2009/38.

[14] Fred Glover. FUTURE PATHS FOR INTEGER PROGRAMMING. 13(5):533–549, 1986.

[15] Benjamin Granger, Marta Yu, and Kathleen Zhou. Optimization with absolute values, May 2014. URL https://optimization.mccormick.northwestern.edu/index.php/Optimization_with_absolute_values.

[16] Tarun Gupta and Ham Id Seifoddini. Production data based similarity coefficient for machine-component grouping decisions in the design of a cellular manufacturing system. *International Journal of Production Research*, 28(7):1247–1269, 1990. ISSN 1366588X. doi: 10.1080/00207549008942791.

[17] Mohsen MD Hassan and Gary L. Hogg. A review of graph theory application to the facilities layout problem. *Omega*, 15(4):291–300, 1987. ISSN 03050483. doi: 10.1016/0305-0483(87)90017-X.

[18] Stephen Hawking. *Brief answers to the big questions*. John Murray Publishers, London, 2018. ISBN 978-1-473-69598-6.

[19] Sunderesh S. Heragu. *Facilities design*. PWS Pub. Co, 1997. ISBN 9780534951832.

[20] Hessing. Green future, logistieke keuzes 31 oktober 2019, 2019.

[21] Miguel Horta, Fábio Coelho, and Susana Relvas. Layout design modelling for a real world just-in-time warehouse. *Computers and Industrial Engineering*, 101:1–9, 2016. ISSN 03608352. doi: 10.1016/j.cie.2016.08.013. URL http://dx.doi.org/10.1016/j.cie.2016.08.013.

[22] DS Johnson. Computers and Intractability-A Guide to the Theory of NP-Completeness. 1979.

[23] L. Kaufman and F. Broeckx. An algorithm for the quadratic assignment problem using Bender's decomposition. *European Journal of Operational Research*, 2(3):207–211, 1978. ISSN 03772217. doi: 10.1016/0377-2217(78)90095-4.

[24] J-g Kim and Y-d Kim. A branch and bound algorithm for locating input and output points of departments on the block layout. pages 517–525, 1999.

[25] J. Y. Kim and Y. D. Kim. Graph theoretic heuristics for unequal-sized facility layout problems. *Omega*, 23(4):391–401, 1995. ISSN 03050483. doi: 10.1016/0305-0483(95)00016-H.

[26] T.C. Koopmans and M.J. Beckmann. Assignment Problems and the Location of Economic Activities Author (s): Tjalling C . Koopmans and Martin Beckmann. *Econometrica*, 25(1):53–76, 1957. URL http://www.jstor.org/stable/1907742.

[27] Panagiotis Kouvelis and Michael W Kim. Unidirectional Loop Network Layout Problem in Automated Manufacturing Systems. (May 2020), 1992.

[28] Andrew Kusiak and Sunderesh S. Heragu. The facility layout problem. *European Journal of Operational Research*, 29(3):229–251, 1987. ISSN 03772217. doi: 10.1016/0377-2217(87)90238-4.

[29] Young Hae Lee and Moon Hwan Lee. A shape-based block layout approach to facility layout problems using hybrid genetic algorithm. *Computers and Industrial Engineering*, 42(2-4):237–248, 2002. ISSN 03608352. doi: 10.1016/S0360-8352(02)00018-9.

[30] Janny Leung. A graph-theoretic heuristic for designing loop-layout manufacturing systems. *European Journal of Operational Research*, 57(2):243–252, 1992. ISSN 03772217. doi: 10.1016/0377-2217(92)90046-C.

[31] Russell D. Meller, Venkat Narayanan, and Pamela H. Vance. Optimal facility layout design. *Operations Research Letters*, 23(3-5):117–127, 1998. ISSN 01676377. doi: 10.1016/S0167-6377(98)00024-8.

[32] Paul Merrell, Eric Schkufza, and Vladlen Koltun. Computer-generated residential building layouts. *ACM SIGGRAPH Asia 2010 papers on - SIGGRAPH ASIA '10*, 29(10):1, 2010. ISSN 07300301. doi: 10.1145/1866158.1866203. URL http://portal.acm.org/citation.cfm?doid=1866158.1866203.

[33] Richard Muther. *Practical plant layout LK - https://tudelft.on.worldcat.org/oclc/781813870*. McGraw-Hill, New York SE - XVI, 363 p. : illustrations ; 29 cm, 1955.

[34] Christopher E. Nugent, Thomas E. Vollmann, and John Ruml. An Experimental Comparison of Techniques for the Assignment of Facilities to Locations. *Operations Research*, 16(1):150–173, 1968. ISSN 0030-364X. doi: 10.1287/opre.16.1.150.

[35] Albert L Page. Pdma new product development survey: Performance and best practices. In *PDMA Conference, Chicago*, volume 13, 1991.

[36] Laurent Perron and Vincent Furnon. Or-tools. URL https://developers.google.com/optimization/.

[37] Mohammad Reza Pourhassan and Sadigh Raissi. An integrated simulation-based optimization technique for multi-objective dynamic facility layout problem. *Journal of Industrial Information Integration*, 8:49–58, 2017. ISSN 2452414X. doi: 10.1016/j.jii.2017.06.001.

[38] Mahmoud Abdel Rahman. Cpython. URL https://www.food4rhino.com/app/ghcpythonr.

[39] Sartaj Sahni and Teofilo Gonzalez. P-complete approximation problems. *J. ACM*, 23(3):555–565, July 1976. ISSN 0004-5411. doi: 10.1145/321958.321975. URL https://doi.org/10.1145/321958.321975.

[40] Daniel Scholz, Anita Petrick, and Wolfgang Domschke. STaTS: A Slicing Tree and Tabu Search based heuristic for the unequal area facility layout problem. *European Journal of Operational Research*, 197(1):166–178, 2009. ISSN 03772217. doi: 10.1016/j.ejor.2008.06.028. URL http://dx.doi.org/10.1016/j.ejor.2008.06.028.

[41] E. Shayan and A. Chittilappilly. Genetic algorithm for facilities layout problems based on slicing tree structure. *International Journal of Production Research*, 42(19):4055–4067, 2004. ISSN 00207543. doi: 10.1080/00207540410001716471.

[42] Amir Mohammad Esmaieeli Sikaroudi and Kamran Shahanaghi. Facility layout by collision detection and force exertion heuristics. *Journal of Manufacturing Systems*, 41:21–30, 2016. ISSN 02786125. doi: 10.1016/j.jmsy.2016.07.001. URL http://dx.doi.org/10.1016/j.jmsy.2016.07.001.

[43] Kar Yan Tam. A simulated annealing algorithm for allocating space to manufacturing cells. *International Journal of Production Research*, 30(1):63–87, 1992. doi: 10.1080/00207549208942878. URL https://doi.org/10.1080/00207549208942878.

[44] Kar Yan Tam and Shih Gong Li. A hierarchical approach to the facility layout problem. *International Journal of Production Research*, 29(1):165–184, 1991. ISSN 1366588X. doi: 10.1080/00207549108930055.

[45] James A Tompkins, John A White, Yavuz A Bozer, and Jose Mario Azaña Tanchoco. *Facilities planning*. John Wiley & Sons, 2010.

[46] Wolfgang Sartorius Von Waltershausen. *Gauss: zum gedächtnis*. S. Hirzel, 1856.

[47] Ming Jaan Wang, Michael H. Hu, and Meei Yuh Ku. A solution to the unequal area facilities layout problem by genetic algorithm. *Computers in Industry*, 56(2):207–220, feb 2005. ISSN 01663615. doi: 10.1016/j.compind.2004.06.003.

[48] P. S. Welgama, P. R. Gibson, and L. A.R. Al-Hakim. Facilities layout: A knowledge-based approach for converting a dual graph into a block layout. *International Journal of Production Economics*, 33 (1-3):17–30, 1994. ISSN 09255273. doi: 10.1016/0925-5273(94)90115-5.

[49] PS Welgama and PR Gibson. A construction algorithm for the machine layout problem with fixed pick-up and drop-off points. *THE INTERNATIONAL JOURNAL OF PRODUCTION RESEARCH*, 31(11):2575–2589, 1993.

[50] Y. Wu and E. Appleton. Integrated design of the block layout and aisle structure by simulated annealing. *International Journal of Production Research*, 40(10):2353–2365, 2002. ISSN 00207543. doi: 10.1080/00207540210133534.

[51] Ding Yang, Yimin Sun, Danilo Di Stefano, and Michela Turrin. A computational design exploration platform supporting the formulation of design concepts. Technical report, 2017.

Appendix A: Temperature differences

A characteristic that is very specific to the food industry and that was also mentioned in chapter 4, is the inclusion of different temperature requirements for every department. This appendix describes the theoretical attempt that was made in order to include these temperature requirements into the objective.

The reason that the temperature differences are added to the objective is that, just like the material flow cost and other flow cost, the flow of temperature costs money due to constant heating/cooling requirements or the necessity to apply insulation where needed, which would bring a one-time cost. In order to implement the temperature in the objective, first, the temperature flow is computed using Fouriers Law, as seen in equation (A.1). This equation states that the temperature flow q is equal to the inverse conduction k multiplied by the temperature gradient ∇T of an object. In order to simplify the problem, the law is taken as 1-dimensional, meaning that the temperature between two departments can only directly flow between them through an invisible substance as seen in equation (A.2), where the temperature flow in the x direction q_x is equal to the conduction k multiplied by the temperature difference between the departments ΔT over the distance between the departments Δx. For all departments this would now be equal to heat flow in 1-dimension between all sets of departments, as seen in equation (A.3).

$$q = -k\nabla T \tag{A.1}$$

$$q_x = -k\frac{\Delta T}{\Delta x} \tag{A.2}$$

$$TemperatureFlow = \sum_{i=1}^{n}\sum_{j=1}^{n} k_{ij}\frac{\Delta T_{ij}}{d_{ij}} \tag{A.3}$$

For simplicity reasons, the conduction k is taken to be one. This ”temperature difference” objective as seen in equation (A.3) is an indicator of how well departments with similar temperature requirements are clustered together. An example of how this works can be seen in figure A.1, where 3 situations are shown: one situation where departments with different temperature requirements are mixed, a situation where they are clustered and a situation where they are clustered but the distance between them is not optimal.

In order to implement this objective to the objective that was thus far used in this thesis, is not simple. This is because the physical dimensions are not commensurate, the flow-cost objective's unit is euros, while the unit for temperature flow would be equal to $\frac{W}{m^2}$. In order to make these dimensions commensurate, a factor would be needed to turn the temperature difference into cost. This would in turn inquire the conduction k to have an actual value. To conclude, making the two dimensions commensurate is not possible given the current problem.

An alternative to add the two objective into one objective is by normalizing the separate objectives by diving them by their reasonable maximum and then adding an importance factor to both. These importance factors are to be different for every different problem, and thus are to be chosen by the owner

of the problem. This multi-objective can be seen in equation (A.4). To monitor the multi-objectives, a Pareto diagram can be made, which plots each iteration on a set of x- and y-axis, where each axis determines the height of that objective. An example can be seen in figure 7.8.

From a pragmatic point of view, the temperature flow objective can not be implemented into the second-stage model due to non-linearity. Since implementing this one way or another would imply yet another simplification, the conclusion is that it would be better to implement the temperature difference in the flow matrix. Then, problem would reduce again to ensure that some departments repel while some others are attracted, all based on one single weighted REL matrix. This would mean that sometimes, even though some departments have a high temperature difference would still be attracted to each other, since the flow between them is just vaster, thus making the cost of extra insulation inevitable.

$$MinFlowCost = \alpha\left(\frac{\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}f_{ij}d_{ij}}{Maximum}\right) + \beta\left(\frac{\sum_{i=1}^{n}\sum_{j=1}^{n} k_{ij}\frac{\Delta T_{ij}}{d_{ij}}}{Maximum}\right) \tag{A.4}$$

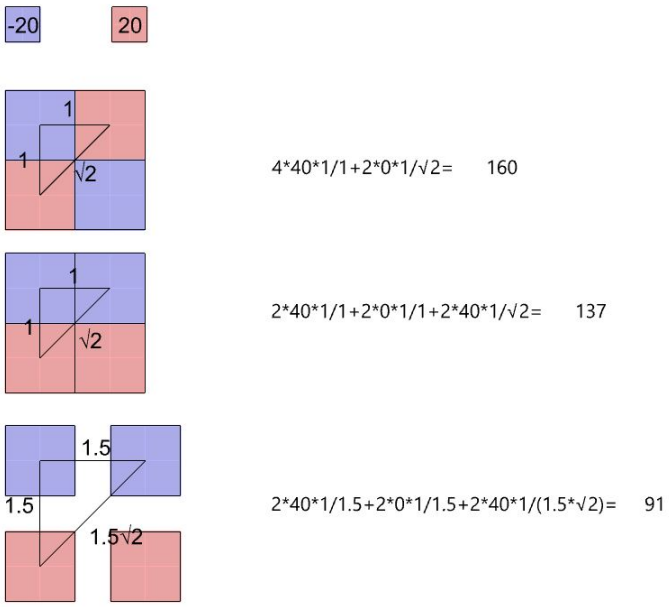


Figure A.1: Three examples showing the outcome of the temperature distance formula. The first example shows a mixed situation and the temperature distance is 160, the second example is a clustered situation and the temperature distance is lower: 137. The third example shows that the temperature distance decreases as department move apart.

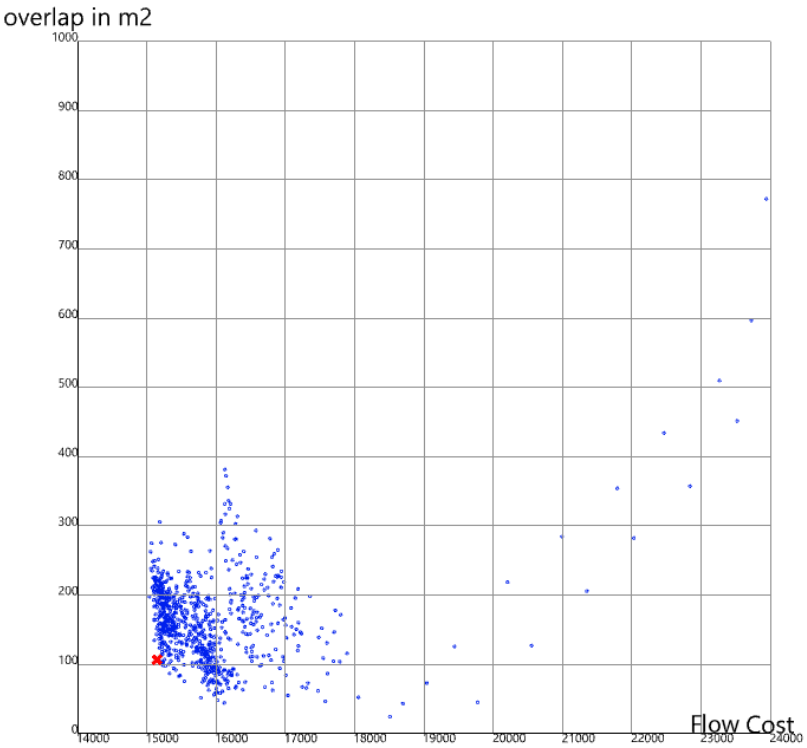


Figure A.2: An example of a Pareto diagram, where both axes portray the height of the respective objective and all points reflect the different solutions.

B

Appendix B: Previous solutions

This appendix describes 9 papers that present their solutions to the FLP. These 9 papers cover different fields of FLP's, from equal to unequal, static to dynamic, and concrete to continuous problems. These papers were tested in chapter 5 based on the criteria for a vegetable processing factory found in chapter 4.

B.1. A solution using simulated annealing

This section is reserved to explain, and is entirely based on, the solution by Chwif et al. [8] for solving the FLP titled "A solution to the facility layout problem using simulated annealing". This continual plane approach is based on the simulated annealing method, which is a resolution approach as discussed in section 3.3.2. This method is used to solve the FLP that is formulated in such a way to also include drop-off and pick-up points.

The problem is formulated so the departments are rectangular, they have an area A_i and an aspect ratio β_i, which is bounded by an upper and lower bound so the following conditions is always satisfied: $\beta_{il} \leqslant \beta_i \leqslant \beta_{iu}$. Since this approach allows departments to be rigid or movable, so in case of a rigid department the aspect ratio is equal to its upper and lower bounds: $\beta_{il} = \beta_i = \beta_{iu}$. In addition to these variables, a department has a specified drop-off and a pick-up point. Six different orientations for these drop-off and pick-up points were specified as seen in figure B.1.

$$d_{ik} = |x_{i,drop-off} - x_{j,pick_up}| + |y_{i,drop-off} - y_{j,pick_up}| \tag{B.1}$$

The distance between departments is measured as the rectilinear distance between them, but instead from center to center, it is now measured from drop-off point to pick-up point as seen in equation (B.1). The objective of the model is to minimize the transportation costs, while a penalty function is added to the objective to avoid department overlapping as seen in equations (B.2) through (B.6). Here f_{ij} is equal to the flow between departments i and j, d_{ij} is the distance as seen in equation (B.1) and α and β are weights for respectively the cost function and the penalty function. The algorithm then tries to search the solution space by placing the departments on a plane, using the simulated annealing method through a number of iterations which is exponential to the number of departments (although the maximum number of departments in this method is only 30). After all iterations, going through the solution space by testing random moves on the planar site, when the temperature is dropped to zero, the algorithm terminates and presents the best found solution $i \, \epsilon \, S$.

$$f = \alpha \sum_{i=1}^{N} \sum_{j=1}^{N} f_{ij} d_{ij} + 2\omega\beta \sum_{i=1}^{N} \sum_{j=1, i \neq j}^{N} I_{ij} \tag{B.2}$$

$$x_{ij} = \begin{cases} A_{ij} + B_{ij} & if\ A_{ij}, B_{ij} > 0 \\ 0 & otherwise \end{cases} \tag{B.3}$$

$$A_{ij} = \frac{W_i + W_j}{2} - |x_i - x_j| \tag{B.4}$$

$$B_{ij} = \frac{H_i + H_j}{2} - |y_i - y_j| \tag{B.5}$$

$$\omega = \frac{\sum_{i=1}^{N} \sum_{j=1, i \neq j}^{N} w_{ij}}{N(N-1)} \tag{B.6}$$

The results of the approach are mildly promising, although difficulties arose with the preventing of overlapping of departments. Dummy problems were tested and compared to a different method from Tam [43], where it is clear that the results by Tam [43] were better, however, the results from this approach laid the departments out more even, which is important to the transportation of materials on the floor as well.

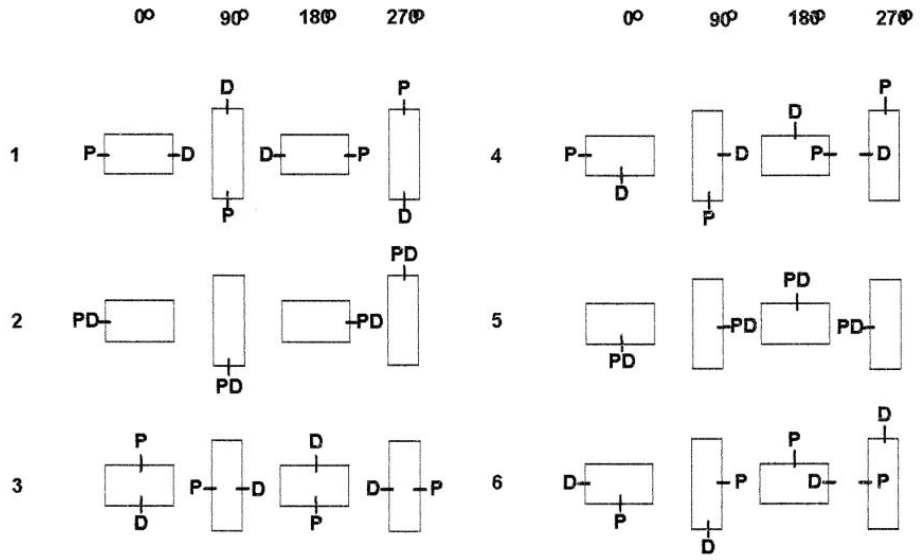


Figure B.1: Different orientations for the Drop-off and Pick-up points over 6 different configurations. Source: Chwif et al. [8]

B.2. A simulation based optimization technique

This section is reserved to explain, and is entirely based on, the solution by Pourhassan and Raissi [37] for solving the FLP titled ”An integrated simulation-based optimization technique for multi-objective dynamic facility layout problem”. This approach is a solution for the dynamic facility layout problem as explained in section 1.3.1. Not only the transportation cost of the layout is taken into account but also the probability of transportation machines interfering with each other (and possibly even crashing) through simulation.

The reason for the addition of the transporter interference objective is that in ideal cases where the material handling costs are optimized, a hefty number of interference between transporting units takes place, this principle is illustrated in figure B.2, where an optimized case for a QAP like problem is presented on the top, but the ideal case for the number of interference is presented at the bottom. This multi-objective is modeled as seen in equations (B.7) and (B.8), with constraints (3.3) and (3.4) as seen in section 3.2.1 and an additional constraint (B.9). This model is not only multi-objective, but also spanning multiple time-spawns that incorporate different flows between machines.

$$MinZ_1 = \left(\sum_{t=2}^{T} \sum_{i=1}^{N} \sum_{k=1}^{N} \sum_{l=1}^{N} A_{tikl} Y_{tikl} + \sum_{t=1}^{T} \sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{k=1}^{N} \sum_{l=1}^{N} F_{tij} D_{kl} C_{ij} X_{tik} X_{tjl} \right) \tag{B.7}$$

$$MinZ_2 = The\ number\ of\ possible\ transporters\ accident \tag{B.8}$$

$$Y_{tikl} = X_{(t-1)ik} X_{til}\ i, j, l = 1, 2, ...N, \forall t, 1, 2, ..., T \tag{B.9}$$

$$X_{tij} = \begin{cases} 1 \ \ if\ machine\ i\ is\ allocated\ to\ location\ j\ in\ period\ t \\ 0 \ \ otherwise \end{cases} \tag{B.10}$$

The model is a QAP like model where machines i and j are arranged to locations k and l. Furthermore, in this model N is the number of machines/locations, T is the number of periods in the planning horizon and t is it's index, A_{tikl} is the cost of moving machines i from location k to location l for time period t, F_{tij} is the flow between machines i and j for period t and lastly, D_{kl} is the distance between locations k and l. Constraint (3.3) ensures that each machines is located to one position, constraint (3.4) ensures that on each location only one machine is present. Additionally constraint (B.9) helps adding the rearranging cost once a machine is shifted from location.

While there is two objectives, only one can be expressed in this model, the material handling costs. Every solution will have to be run through a simulation model that is agent based and estimates the amount of interference. The model is then optimized for both of these objectives and improved iterative through genetic algorithm for a number of iterations, the variables that are being changed are the string that determine which machine is assigned to which location. The results are promising, however there are major drawbacks. Firstly, all the machines and locations are treated as equal-sized, which is unrealistic. Secondly, the method is very time consuming as the number of departments start to increase due to the need to run a simulation for every iterative solution.

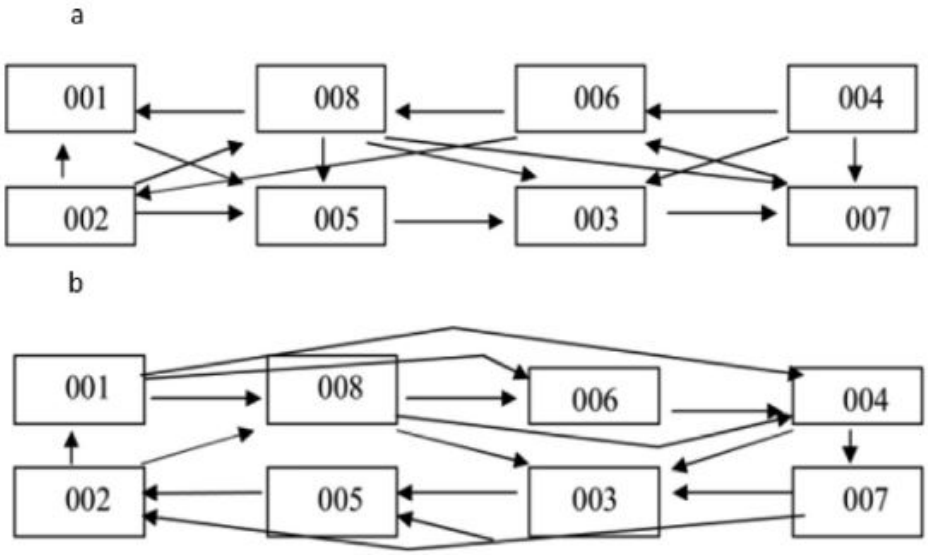


Figure B.2: A solution to an eight facility QAP like problem, with (a) the most effective material handling cost and (b) the least amount of interference. Source: Horta et al. [21]

B.3. A two-stage optimization-based framework

This section is reserved to explain, and is entirely based on, the solution by Anjos and Vieira [1] for solving the FLP titled "An improved two-stage optimization-based framework for unequal-areas facility layout". This framework provided is based on two different mathematical optimization models, where the first model is to find out the relative positions of the departments through nonlinear approximation, and the second model is there to determine the final layout through exact convex optimization.

The notation of the problem as a basis given in the paper is similar to the notation given in this chapter for the mixed integer programming. The objective is exactly the same as in (3.9), minus the drop-off and pick-up points. Constraints are set to make sure that departments remain inside the facility's width W_f and the facility's height H_f, the area requirement A_i is met, there is a max ration constraint β, and there is a minimum and maximum width and height W_i^{max} W_i^{min} H_i^{max} H_i^{min}. Also there is a constraint to prevent overlapping as seen in (B.11).

$$|x_i - x_j| \geqslant \frac{1}{2}(W_i + W_j) \quad or \quad |y_i - y_j| \geqslant \frac{1}{2}(H_i + H_j) \tag{B.11}$$

The first stage model's objective is to find a good estimate about the relative positions of the departments, therefor, in the first stage model, overlap constraints are relaxed and given a penalty function. These overlap relaxation is both for the overlapping of two departments as well as the overlap of a department and the outside perimeter of the entire facility. This overlap penalty comes in the form of $f\left(\frac{D_{ij}^2}{T_{ij}^2}\right)$ where $D_{ij}^2 = (x_i - x_j)^2 + (y_i - y_j)^2$ and $T_{ij}^2 = \frac{1}{4}\left((W_i + W_k)^2 + (H_i + H_j)^2\right)$ as seen in figure B.9. Here T_{ij}^2 is the squared target distance and D_{ij}^2 is the squared euclidean distance . This is finally implemented in the first stage model with a objective as seen in (B.12) and constraints as seen in (3.15 through 3.20). In this model, K is a variable to determine the severity of the penalty function as seen in (B.13).

$$MinCost = \sum_{1 \leqslant i \leqslant j \leqslant N} \left(c_{ij} D_{ij}^2 + K \frac{T_{ij}^2}{D_{ij}^2} - 1 \right) \tag{B.12}$$

$$K = \alpha \sum_{1 \leqslant i \leqslant j \leqslant N} c_{ij}, with 0 < \alpha \leqslant 1 \tag{B.13}$$

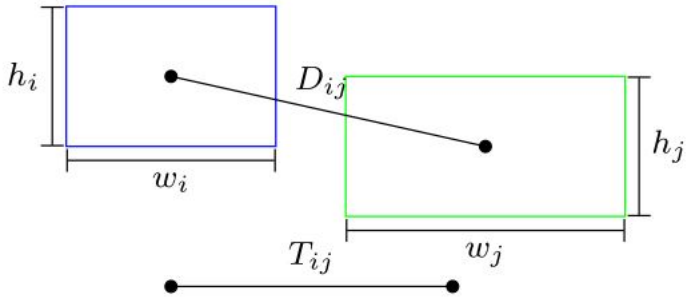


Figure B.3: The target distance for departments i and j. Source: Anjos and Vieira [1].

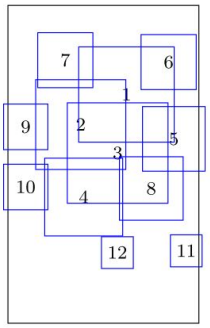


Figure B.4: An example of a first-stage optimal solution. Source: Anjos and Vieira [1].

The result for a dummy problem after being optimized by the first stage model can be seen in figure B.4, where it is very clear that the departments overlap. However, this is not a problem because the goal is not to find the final layout but the relative position of the departments, which becomes clear from this first stage model. A trick to find this clarity is to choose the variable α in such a way that the departments are not punished too much thus resulting in errors when trying to compute the layout and not to punish overlap to lightly to prevent unreadable results when all departments overlap each other

in one big cluster. With the results as shown in figure B.4, the relative positions of departments will be clear and can be transformed into constraints during the second stage of the framework.

The second stage model produces the feasible layout satisfying all the constraints and conditions, including those that were found in the first stage model. In this second stage, the objective becomes different, as seen in equation (B.14), minimizing the cost still but not adding a penalty function for overlapping, as these are now hard constraints. The constraint stay the same as in the first model, with the constraints for area, aspect ratio, outside perimeter overlapping and minimum and maximum width and height as seen in (3.15 through 3.20). New constraints are added to linearize the rectilinear distance as seen in equation (B.15) and equation (B.16). The constraints that are given from the first stage model and seen in equation (B.17) are selected by applying a rule to the first-stage solution. This rule is that the difference in x and y coordinates are computed, e.g. Δx and Δy, after which is evaluated which of the two is greater. If Δx is greater, a constraint is added to separate the two departments horizontally, and else vertically.

$$MinCost = \sum_{1 \leqslant i \leqslant j \leqslant N} c_{ij}(u_{ij} + v_{ij}) \tag{B.14}$$

$$u_{ij} \geqslant x_j - x_i, \ and \ u_{ij} \geqslant x_i - x_j, \quad for\ all\ 1 \leqslant i \leqslant j \leqslant N \tag{B.15}$$

$$v_{ij} \geqslant y_j - y_i, \ and \ v_{ij} \geqslant y_i - y_j, \quad for\ all\ 1 \leqslant i \leqslant j \leqslant N \tag{B.16}$$

$$No - overlap\ constraints \ \ for\ all\ 1 \leqslant i \leqslant j \leqslant N \tag{B.17}$$

The results of this paper show the effectiveness of the framework, showing great improvement over dummy and toy problems in segments with problems of sizes ranging from 5 to 100 departments. In the results is also discussed how the in the solution empty space is being recognized and compared to other problems in the results. Due to the recentness of the report and the results it can be concluded that this solution is very promising.

B.4. A solution by genetic algorithm and space filling curves

This section is reserved to explain, and is entirely based on, the solution by Wang et al. [47] for solving the FLP titled ”A solution to the unequal area facilities layout problem by genetic algorithm”. This approach makes use of space filling curves to fill the plane with the unequally sized departments without creating gaps. Furthermore, the approach tries to optimize the total layout costs, which does not only consist of the material flow factor cost, but also the shape ratio factor and the area utilization factor.

In addition to the material flow factor, as seen in equation (B.18), there are two objectives that are of importance to this author, which are the shape ratio factor SFR and the area utilization factor AUF, which were already discussed in section 3.2.1 and equations (3.5) and (3.6). Together these three objectives are combined into one objective in the shape of the total layout cost TLC, the reason for this being is that a good layout should not only take into account material flow costs but also a decent geometric shape of the departments and site and the area utilization. In equation (B.18), C_{ij} is the transportation cost for transporting between departments i and j, f_{ij} is the flow between those departments and d_{ij} is the distance between them.

The minimum TLC, which is the objective of the mathematical model, is formulated as seen in equation (B.19). The following constraints have also been added to this model: constraint (B.20) ensures that no more than one department is assigned to the same location, constraint (B.21) ensures that the number of locations that are assigned to each department is not bigger than the area that each department requires and constraint (B.22) represents that the total area of departments combined is never bigger than the total plant size. Where $A_{ijk} = 1$ if department j is assigned to the location at the ith row and the kth column or 0 otherwise, L and W are the maximum length and width of the plant site and A_j the area requirement of department j.

$$MinMFFC = \sum \sum C_{ij} f_{ij} d_{ij} \tag{B.18}$$

$$MinTLC = MFFC \frac{SRF_{whole}}{AUF_{whole}} \tag{B.19}$$

$$\sum_{k=1}^{N} a_{ijk} \leqslant 1 \quad for\ all\ i\ and\ j \tag{B.20}$$

$$\sum_{i=1}^{W}\sum_{j=1}^{L} a_{ijk} \leqslant A_j \quad for\ all\ j \tag{B.21}$$

$$\sum_{i=1}^{W}\sum_{j=1}^{L}\sum_{k=1}^{N} a_{ijk} \leqslant LW \tag{B.22}$$

In order to get a near optimal solution, the algorithm makes uses of the genetic algorithm principle, as explained in section 3.3.2. The data that is being mutated and transformed through genetic algorithm is a string of numbers that represents the way the layout is build. This string is namely responsible for constructing the space filling curve SFC, which is a curve over the different locations that fills the plane and its locations like a snake, first assigning a full department until its area requirement are met and then the next, until all departments are placed. The order of the departments together with its area requirements are hidden in the string, as well as the sweeping direction (horizontal or vertical) and the width of the sweep. This progress is best explained visually in figure B.5.

After a number of iterations the algorithm will have found the most optimal string of numbers, which is a solution that can be reproduced by reconstructing the layout with the space filling curve algorithm. The results for the mod are promising, leading to good solutions in reasonable time, improving upon existing solutions by other algorithms. However, the solutions has restrictions, as only certain configurations of shape are possible with this method. Also, the introduction of aspect ratio constraints or fixed dimensions are near impossible, if not impossible.

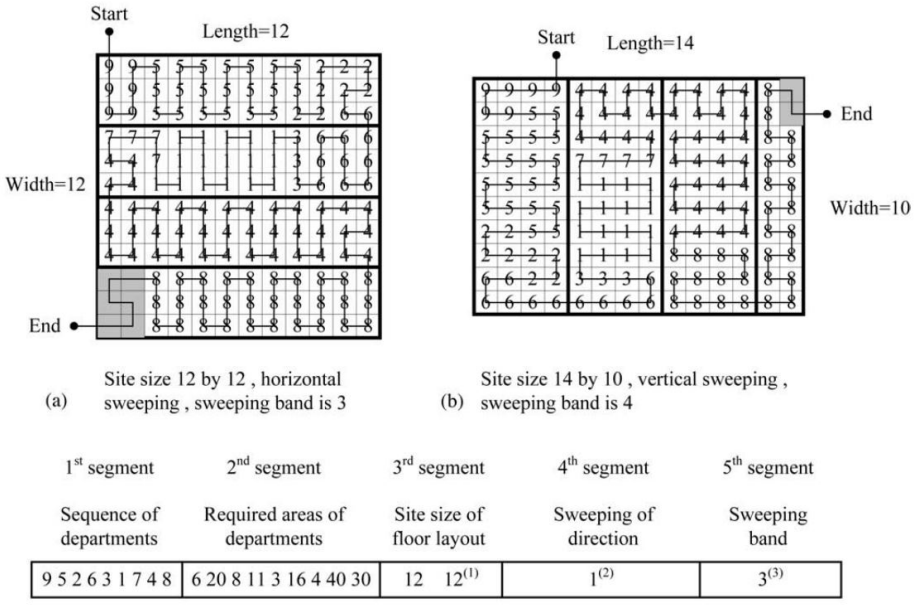


Figure B.5: Layout representation as a string by making use of the space filling curve example, the presented string yields the layout on the left. Source: Wang et al. [47]

B.5. A knowledge-based approach for converting a dual graph into a block layout

This section is reserved to explain, and is entirely based on, the solution by Welgama et al. [48] for solving the FLP titled ”A knowledge-based approach for converting a dual graph into a block layout”. This method’s entire purpose is based on the third step fromHassan and Hogg [17] for generating a layout

using graph theory. This third step was to convert the achieved dual graph into a block layout. This approach uses web-grammar rules to select facilities sequentially. It also determines the departments dimensions and reduces empty space.

Seeing as the main input is the maximal planar weighted graph, this model's objective is not mathematically quantifiable, seeing as this was already done when the maximal planar weighted graph was achieved. Rather, the objective is to maximise the number of connections that were originally found making the MPWG and reducing the empty space of the layout, all the while using regular shapes (rectangles) as department shapes, as they are most practical with placing machines and other layout filling facilities.

The algorithm is based on a knowledge-based system, which means it is an AI that goes through a set of IF, THEN, ELSE rules through several steps. In order to come to a final layout, seven steps are practised by this system. First of all, the MPWG is achieved. Secondly, the first department S is selected, which is often the department with the most connections.Thirdly, the placement sequence G_i, which consist of all departments connected to S, is determined by the M-matrix of connections and chosen in such a way that connected departments end up next to each other. Then fourthly, using a set of IF, THEN, ELSE rules, all departments in G_i are placed, also assigning a width and a height. Now, fifthly, a new starting department is chosen, again, this is the department with most connections to unplaced departments and steps three till 4 are repeated. Finally empty space is removed as much as possible.

Although the procedure is one of the most efficient for developing a MPWG into a block layout, which is thought of to be very hard, it has difficulties in always maintaining all connections specified by the MPWG. In most cases, this is not possible using only rectangular department shapes. This method is a good attempt at using graph theory to produce a layout, but like most construction algorithms, it lacks the ways to improve upon a solution, greatly reducing the chances to find a (near) optimal solution.

B.6. A graph theoretic heuristic

This section is reserved to explain, and is entirely based on, the solution by Kim and Kim [25] for solving the FLP titled "Graph theoretic heuristics for unequal-sized facility layout problem". This method is based on graph theory and departments that are of unequal size. The method constructs a planar graph that tries to maximise the flows attributed to the edges of the graph (between departments), and then generates a layout by a constructive method, following a QAP-like procedure. In order to optimise the layout, the graph is optimised instead of the physical layout.

The proposed method follows the three steps presented by Hassan and Hogg [17], where first, the maximal planar weighted graph is constructed, secondly, the dual of that graph is found and thirdly, the block layout can be constructed. Previous solutions have come across the umbrella effect problem, which is the phenomenon where one department would be attached to all other departments, creating problems when trying to maintain these connections in the actual block layout. In order to overcome this problem, departments are inserted into a face the graph in sets of triplets. Because the graph starts with a sub-graph of G, which consists of the three biggest flows (and their vertexes) making a triangle, the graph will remain triangulated. Furthermore, because graphs are inserted in sets of triplets, it is harder for the umbrella effect to occur.

Once the three biggest flows and their departments make the starting sub-graph of G, all possible triplets are considered for placement into one of the faces of the graph, which, at the start, is only one possible face. Then, the triplet that maximises the flows in a certain face are inserted in that certain face, where each vertex makes a new edge with two of the vertexes that made up the face in which the triplet was placed, see figure B.6. Then, the process is repeated until there are less than 3 departments left to place. After that, if the remaining departments is not 0, the remaining departments will be placed as a single department.

Next the block layout is constructed using a constructive method, where departments are placed one by one. The first department to be placed is the biggest department that has a connection with the exterior. Then each next department placed is the department that is adjacent to at least two departments that are already been placed but have not reached their potential adjacencies in the layout yet. This department is then placed on the best scoring candidate from a set of candidate squares (locations) in the matrix, also taking into account previously placed departments and departments yet to be placed.

Then, when the placement of all departments is done, the objective is calculated, which is equal to the sum of all costs multiplied by the flows and rectilinear distance for all sets of departments. This objective is then stored as the minimized objective, after which one of four optimization procedures is applied on the graph to try and optimise the objective. These procedures are based on the two following techniques. The first is the interchange of two vertexes and their edges and the second is the relocation of a vertex with exactly three edges into a different face.

Based on these two techniques, the four optimisation procedures are as follows. The INHE technique only swaps vertexes until no further improvements can be made. First all possible swaps are considered for moving (also reconstructing the layout every time to see its effect on the objective), then the best swap is performed and the procedure is repeated until there is no option left that improves the procedure. The second is the MOIN procedure, which moves vertexes until no further improvements can be made and then swaps vertexes until no further improvements can be made. Thirdly, the INMO procedures swaps vertexes until no further improvements can be made, and then proceeds to move vertexes until no further improvements can be made, essentially being the reverse procedure of MOIN. Lastly, the SIM procedure tries all option for moving and swapping at the same time, and then applies the most profitable one. This process is also repeated until no further improvements can be made.

The paper concludes that the vertex interchange worked better than the vertex movement technique, which is visible in the results, where INMO outperforms MOIN. INMO also outperforms the then popular algorithm CRAFT, showcasing the great potential of the method. This method also works with up to 30 departments, completing the improvement procedure in under 15 minutes. However, the method lacks ways to implement geometrical constraints, or at least, implementing them would change the nature of the method.

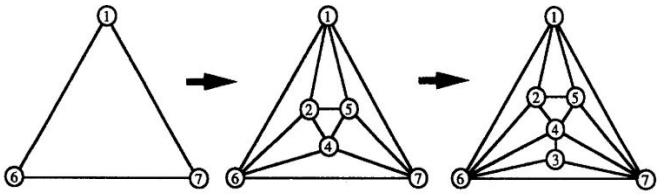


Figure B.6: The procedure for inserting triplets of vertices into the graph (middle) and inserting a single vertex (right). Source: Kim and Kim [25]

B.7. An improved tabu search heuristic

This section is reserved to explain, and is entirely based on, the solution by Chiang and Kouvelis [7] for solving the FLP titled ”An improved tabu search heuristic for solving facility layout design problems”. This method is based on a QAP formulation of the problem and departments that are of equal size. All departments i are assigned to specific locations k. By making use of a tabu search algorithm, a near optimal solution is found. The tabu search algorithm helps diversification in early stages of the solution.

The formulation of this problem is like the formulation seen in equation (3.1), in combination with the constraints found in equations with the exception that instead of $f_{ij}d_{kl}$, this paper uses the variable A_{ijkl}, which is equal to either $f_{ij}d_{kl}$, F_{ij} or ∞, as seen in equation (B.23). In this model, F_{ik} is the fixed cost for locating facility i at site k.

$$A_{ijkl} = \begin{cases} f_{ij}d_{kl} & if\ i{\neq}k\ or\ j{\neq}l \\ F_{ik} & if\ i{=}k\ and\ j{=}l \\ \infty & otherwise \end{cases} \tag{B.23}$$

The algorithm takes an initial solution and calculating its objective, then, all possible solutions that come forth by swapping two departments are evaluated and the best solution is kept. This continues until the program terminates. However, additional constraints are added that diversify the search of the solution space and therefor preventing from being trapped in a local optimum. These extra constraints are added in the form of ”tabu” solutions. These are solutions that are excluded from the solution space at a specific iteration using a set of rules that are specified below. The first rule is that no two departments can be exchanged too soon after they were already swapped. The second rule is that no two departments can be swapped a large number of times if others have not been swapped a large

number of times. These rules are dependant on the iteration number and help diversify the search in earlier stages of the algorithm.

The information for considering whether a solution is tabu or not is being stored in a two dimensional array. This array is divided into two triangles, from top left to the down right corner. The columns and rows represent the different departments. The upper triangles stores a value Tabu[i][k], which is awarded after departments i and j have been swapped and is equal to the current iteration number plus t, a factor that is scalar and dependant on the tabu size list. Now the first rule states that two departments can not be exchanged if Tabu[i][k] $\leqslant$ the current iteration. This helps the two departments from being swapped back and forth repeatedly in a few iterations time. The second rule is based on how many times the departments were already swapped, which is stored in the lower part of the triangle. This rule states that solution gained by the swapping of two departments has a penalty function added to it, based on how many times the departments were already swapped.

In addition to these two rules, the parameter t is used to diversify the search. This parameter t, which is presented in the first rule, has a lower bound and upper bound. If the yielded solution does not show an improvement α which is significant, the parameter t is set somewhere between these bounds proportionate to α. If the previous solution had a very good improvement $\alpha > \beta$, the search is intensified by setting $t = 2n$. Another strategy to intensify the search is to fix two departments when the improvement α is sufficiently high.

The method was applied to several problems from the literature ranging up to 100 departments. The results of this research are good. However, since this algorithm does not take unequal sized areas into account its results are hard to judge. Because equal-sized departments are used, the method is not applicable in this thesis. However, certain aspects and ways of diversifying and intensifying the search of the solution space remain applicable.

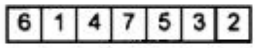


Figure 1. Initial layout.

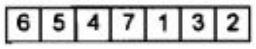


Figure 2. New layout.

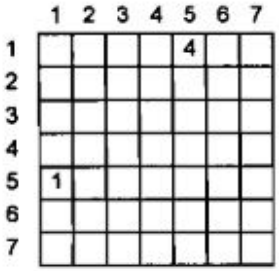


Figure B.7: The two dimensional array for tabu solutions, filled in after one iteration of swapping departments 1 and 5, with $t = 3$. Source: Chiang and Kouvelis [7]

B.8. Layout by collision detection and force exertion

This section is reserved to explain, and is entirely based on, the solution by Sikaroudi and Shahanaghi [42] for solving the FLP titled ”Facility layout collision detection and force exertion heuristics”. This method is based on the attraction of two forces, like a mass-spring system, that will want to relax until it has achieved an equilibrium. In this method, every department is presented by a randomly generated point on a plane, and the departments exert forces on each other based on a REL matrix. If points overlap, they are presented a repulsive force that is weighted based on the amount of overlap. After a number of iterations, the departments will have moved into this equilibrium into what is supposedly a near optimal solution.

The prerequisites for this method are the REL-matrix, which can be positive and negative in any scale, as opposed to the A, E, I, O, U and X relations first proposed by Muther [33], and the sizes of the departments, as they are needed to define when departments overlap. The paper uses two different methods to represent the departments, a mesh format and a non mesh format. In the former one, a facility consist of a number of vertexes on a grid that can approximate any shape, whereas in the non mesh format, a departments can only take rectangular shape, as they are presented with a width and height.

The forces between each department pair i and k is determined as seen in equations (B.24) and (B.25). In these equations a few things can be noticed. First of all that the distance is important in determining the force between two points, since the distance is used as a multiplication factor in equation (B.25), this means that the further the departments are, the bigger the force between two departments is. This makes sense, as a bigger distance between two departments leaves the most room for improvement. Secondly, the distance isn't measured exactly, but the Pythagoras's theorem is accomplished by the factors -5 and 1 in the numerator and denominator respectively, this is to prevent a division by 0 when points exactly overlap and infinitely large forces when the denominator approximates zero. The force vectors are also based on the coordinates of the two departments and on the relation between them, the bigger the relation, the bigger the force between them.

$$force(point_i, point_j) = ((x_j - x_i)xrelations_{i,j}xdistance_{i,j}, (y_j - y_i)xrelations_{i,j}xdistance_{i,j}) \quad \text{(B.24)}$$

$$distance_{i,j} = \frac{-((\sqrt{(x_j - x_i)^2 + (y_j - y_i)^2}) - 5)}{(x_j - x_i)^2 + (y_j - y_i)^2 + 1} \quad \text{(B.25)}$$

After the calculation of all forces on all points, the vectors are added to find the final vector for each point. Then, each point is moved a set distance in the direction of the forces (e.g. all points move 1 unit distance in the direction of the vector) and the coordinates of the departments are updated, after this, the overlap for each department is calculated using a certain method (depending on whether the mesh or non mesh format is used) and the departments are moved again. In the mesh format, the amount of movement depends on the amount of overlap, and is in the opposite direction of the overlap centre.

In the mesh format the department are represented by a plane of vertexes, if a vertex falls in a departments boundary, it is assigned the value 1, and if it overlaps, it will get assigned a lower value, depending on the distance to the closest 1 value, if the departments overlap with the outside facility, it will get a lower value assigned as well. After the attraction of departments, the vertexes values are computed and a new centre of the department's shape is determined by taking the average of all vertexes position multiplied by their value.

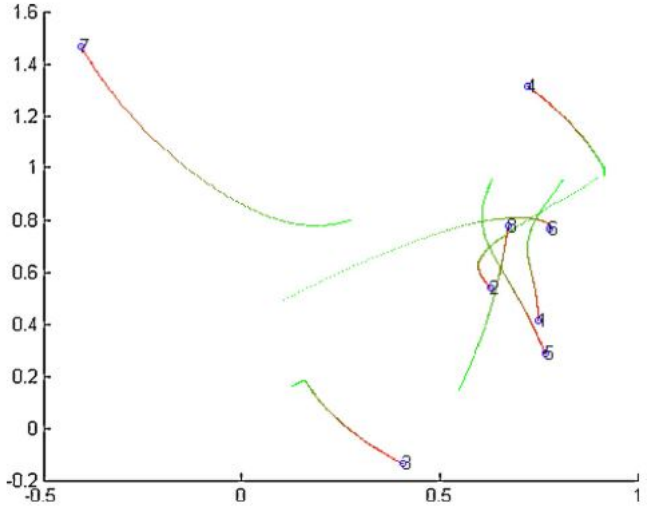


Figure B.8: The displacements of departments through a number of iterations, starting from green and ending at red. Source: Sikaroudi and Shahanaghi [42]

Figure B.9: The grid in mesh format, green points fall inside departments, blue points don't, yellow points are points occupied by two or more departments and red points fall outside the outer walls of the facility. Source: Sikaroudi and Shahanaghi [42]

The results of this paper were not compared to any other method, but the mesh and non mesh formats were compared with each other. The conclusion is that non mesh format outperforms mesh formats and although the run-time needed for 100 departments is reasonable with an average of 744.05 seconds, the method has some pitfalls as well. For example, free space is removed as a result of the the departments moving towards each other, but no measures are taken to explicitly remove all free space, thus, solutions may contain gaps and are punished for it instead of trying to resolve this. Furthermore, the results are different every time, as the random placement of departments is different. Also, the possibility to study different aspect ratios for free shaped departments is very limited. The possibility to rotate unsymmetrical departments is also not possible, limiting the chance to find a (near) optimal solution.

B.9. A slicing tree based heuristic

This section is reserved to explain, and is entirely based on, the solution by Scholz et al. [40] for solving the FLP titled "STaTS: A Slicing Tree and Tabu Search based heuristic for the unequal area facility layout problem". This method is based on the slicing tree procedure, which is a way to represent a layout, and then tries to optimize this slicing tree in order to get to better results. Slicing trees have been around longer, but Scholz et al. [40] introduces the ability to distinguish between constraint department and free-shaped departments for the first time.

The slicing tree generated layout is the result of dividing an initial rectangle in horizontal or vertical direction so that two new rectangles are born, and repeating this process until the layout is realised. The way that the instructions are given as to how to cut the rectangles is given with the slicing tree, which looks like a graph, where there is a starting node and a branching structure. In this branching structure, the leaf nodes are the departments and the branches where they intertwine are the horizontal or vertical division between them. Figure B.10 shows a slicing tree and its corresponding layout.

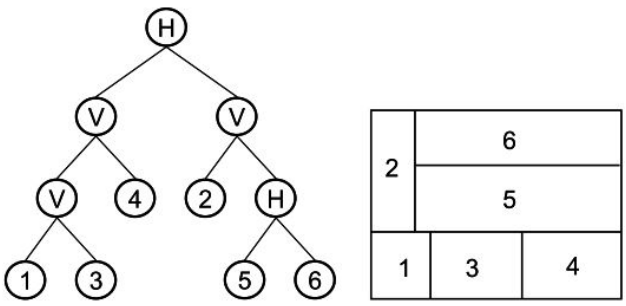


Figure B.10: The grid in mesh format, green points fall inside departments, blue points don't, yellow points are points occupied by two or more departments and red points fall outside the outer walls of the facility. Source: Sikaroudi and Shahanaghi [42]

What makes this method unique is its ability to implement geometric constraints for just part of the departments. The method is able to distinguish between 4 different types of facilities as follows:

- Departments with free aspect ratios
- Departments with fixed dimensions
- Departments with a restricted range of aspect ratios
- Departments which have a set of fixed dimensions to choose from

These constraints are implemented in the shape of bounding curves, which are curves on an axis of x and y coordinates that indicate all possible combinations 0f valid outcomes for the departments in question, as seen in figure ref. These bounding curves are stored in the algorithm for each department.

The way that the layout is now generated is as follows: Starting from the bottom leave nodes, their bounding curves are added in one direction, the y direction in case of a horizontal division and in the x direction in case of a vertical division. This process repeats until the bounding curve of the root node is computed, which now holds the information of the absolute minimum total facility dimensions. Next, the layout can be constructed by choosing a width or height of the total facility (x or y value on the root node's bounding curve) which, after being sliced, gives another x or y value that can be used on the next bounding curve. This process now constructs the layout following the chain of bounding curves.

$$Minimize F = \sum_{i=1}^{n} \sum_{j=1}^{n} f_{ij}(|x_j^I - x_i^O| + |y_j^I - y_i^O|) \tag{B.26}$$

For each slicing tree, multiple starting values on the root node's bounding curves are tried and the best performing layout is being kept. The algorithm tries to minimize the objective F as seen in equation (B.26). In order to improve the objective after trying an initial slicing tree, a tabu search heuristic is applied that tries the following procedures through a number of iterations:

- Swap two leave nodes
- Invert the H/V of one branching node
- Swap a branching tree with another tree or a leave node
- Swap two nodes, any node and its parent, with any other node and its parent

After the tabu search has terminated, the results compared to existing methods are very promising, where there is more often than not an improvement. Furthermore, the method is very fast and is thus able to even generate layouts for up to 100 departments quickly. A limitation of the method is that some layouts are impossible to create considering that only layouts that come from a slicing tree are possible.

C

Appendix C: a sample of results for the vanilla first-stage model

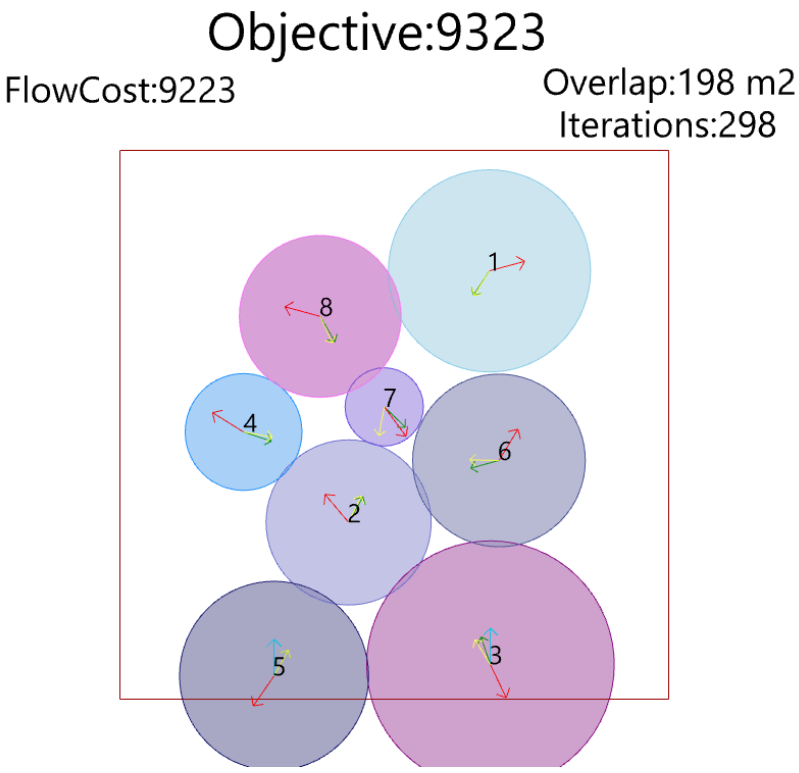


Figure C.1: The layout for sample nr 1 of the 8 department problem for the first-stage model considering the vanilla procedure.

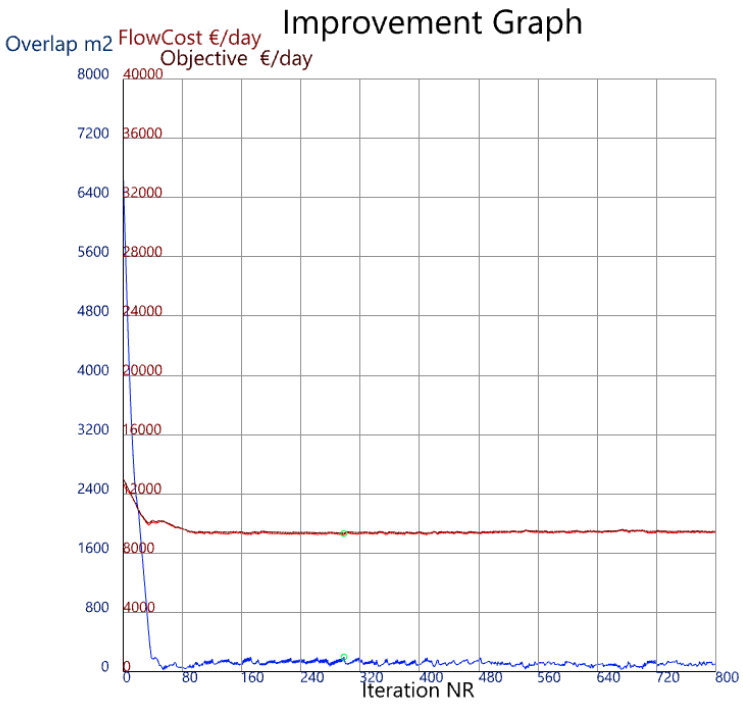


Figure C.2: The graph for sample nr 1 of the 8 department problem for the first-stage model considering the vanilla procedure.

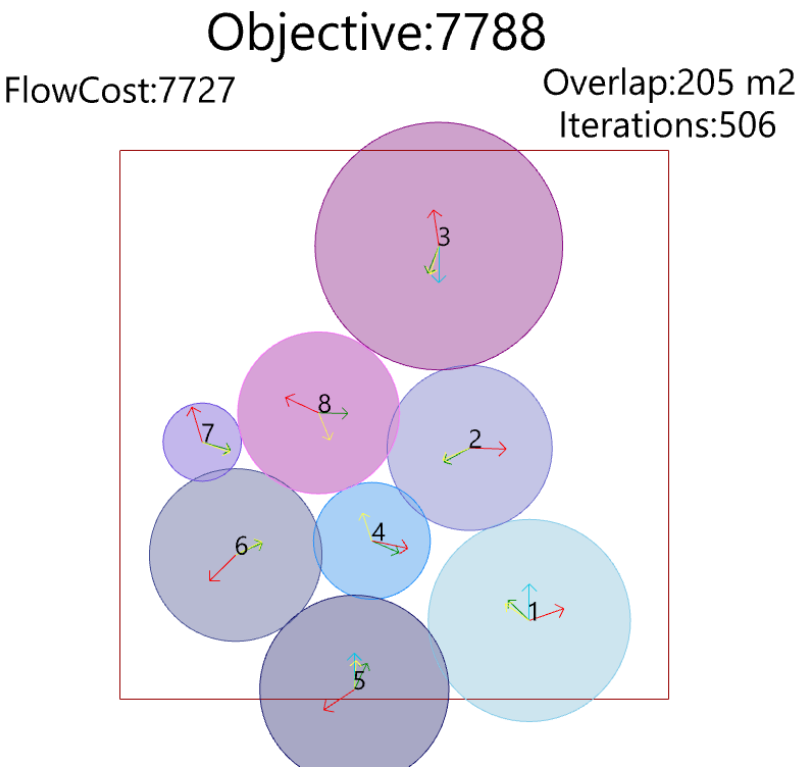


Figure C.3: The layout for sample nr 2 of the 8 department problem for the first-stage model considering the vanilla procedure.

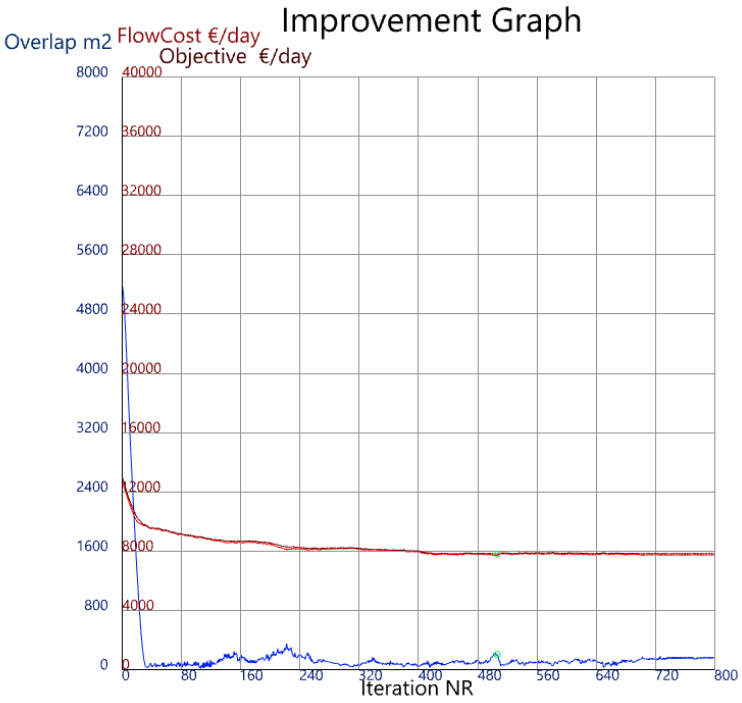


Figure C.4: The graph for sample nr 2 of the 8 department problem for the first-stage model considering the vanilla procedure.

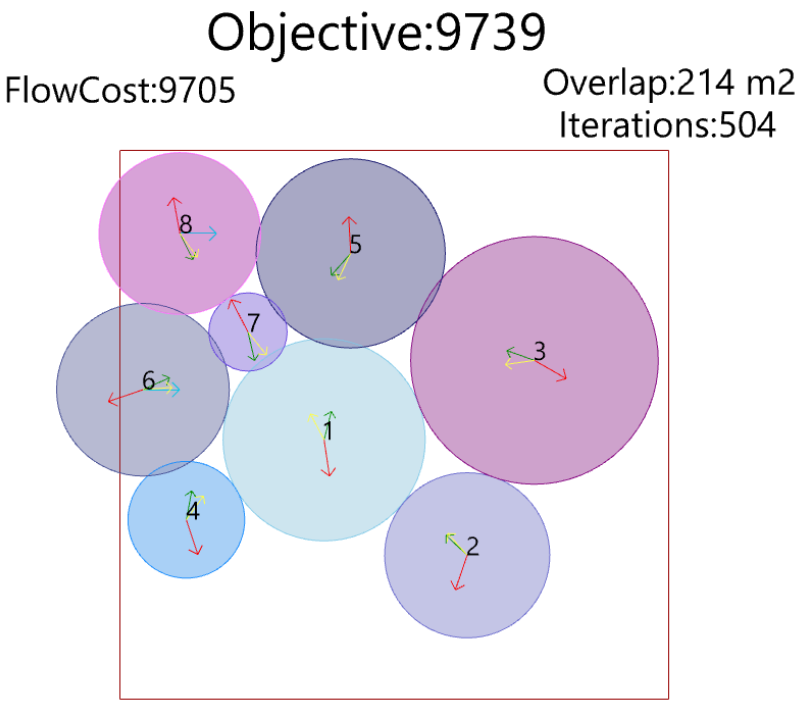


Figure C.5: The layout for sample nr 3 of the 8 department problem for the first-stage model considering the vanilla procedure.

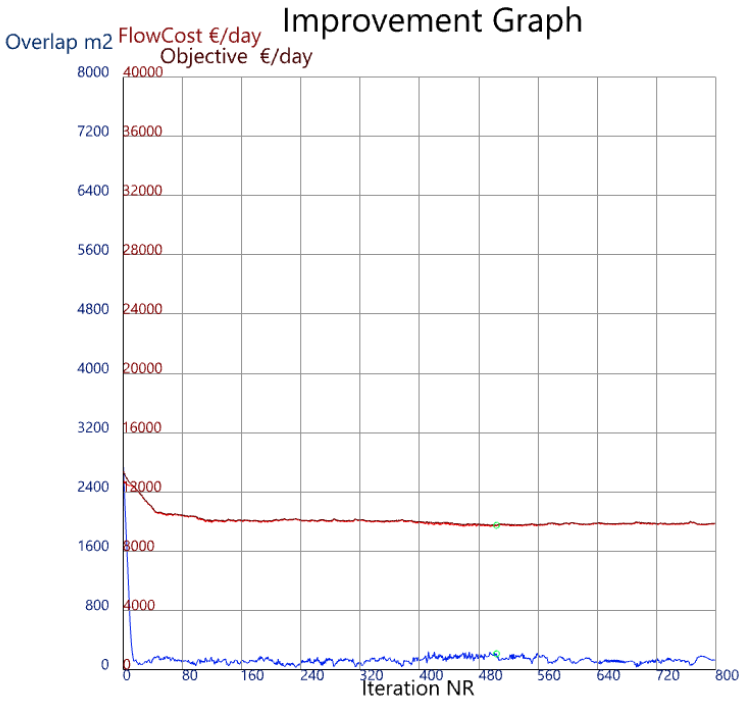


Figure C.6: The graph for sample nr 3 of the 8 department problem for the first-stage model considering the vanilla procedure.

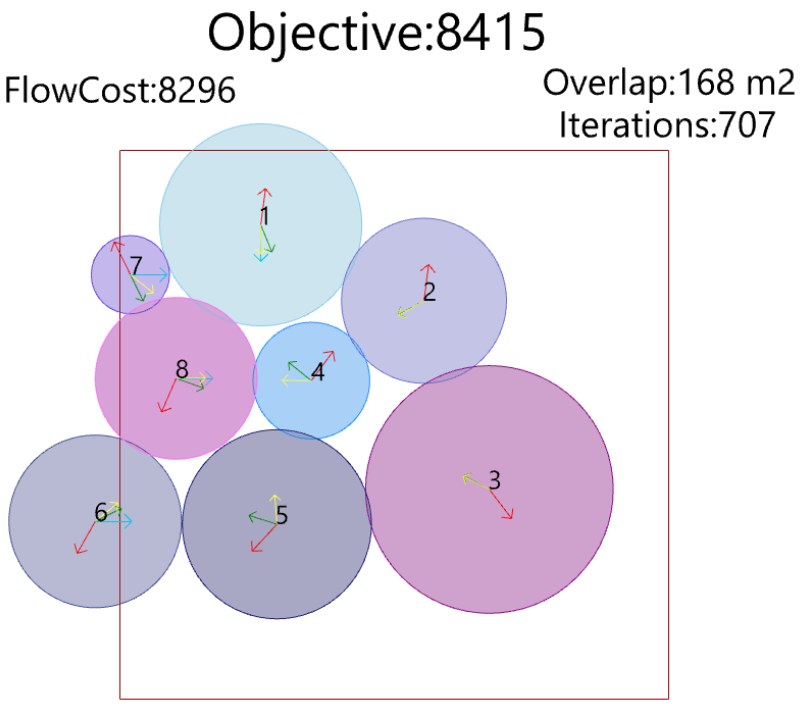


Figure C.7: The layout for sample nr 4 of the 8 department problem for the first-stage model considering the vanilla procedure.

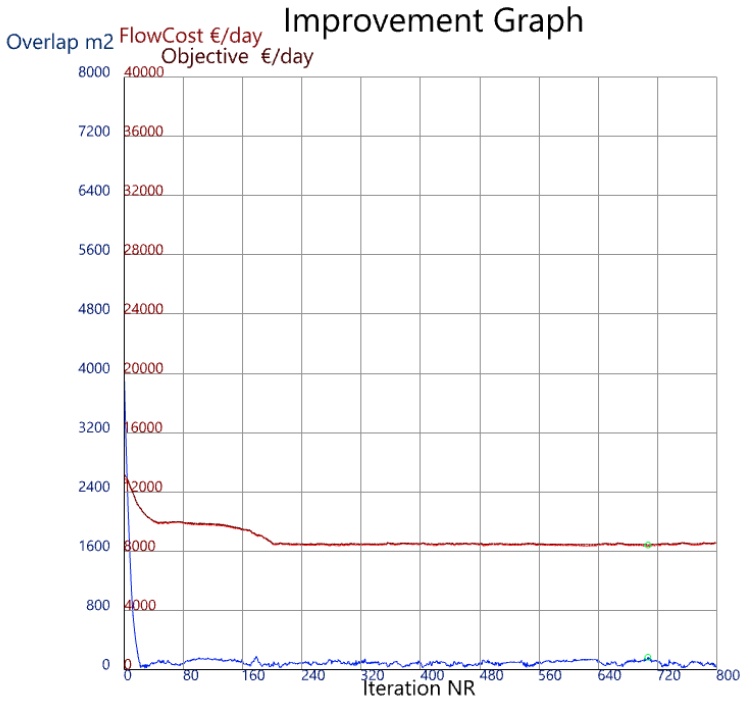


Figure C.8: The graph for sample nr 4 of the 8 department problem for the first-stage model considering the vanilla procedure.

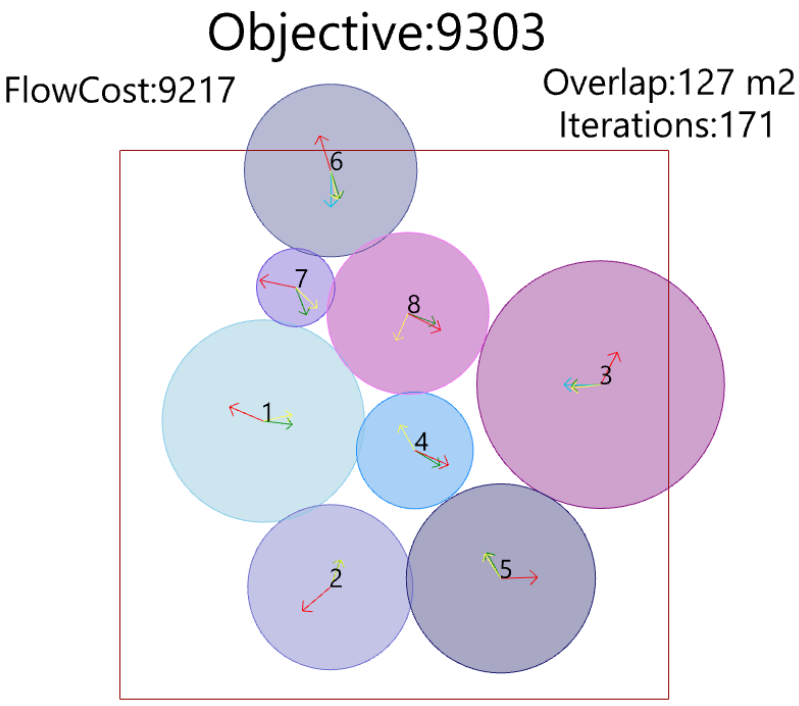


Figure C.9: The layout for sample nr 5 of the 8 department problem for the first-stage model considering the vanilla procedure.

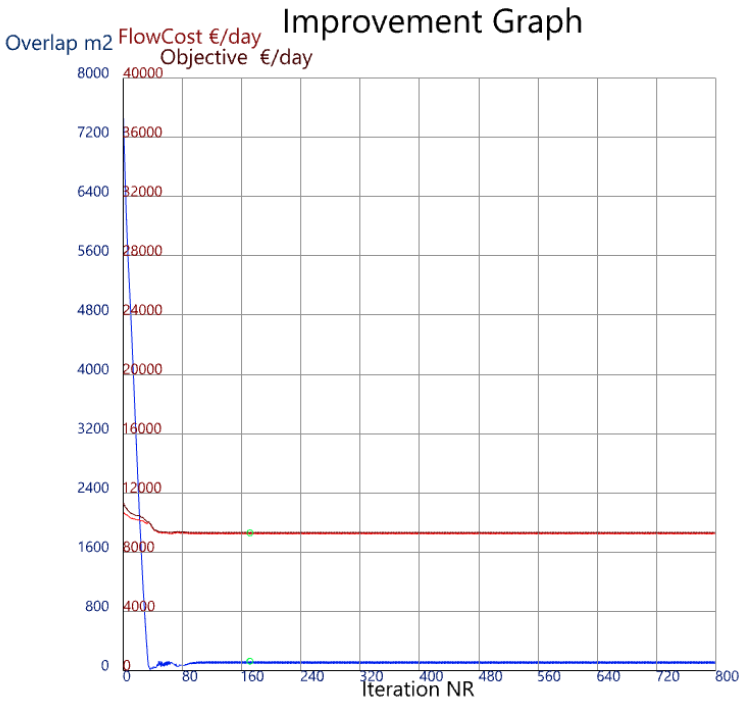


Figure C.10: The graph for sample nr 5 of the 8 department problem for the first-stage model considering the vanilla procedure.

Appendix D: a sample of results for the swapping first-stage model

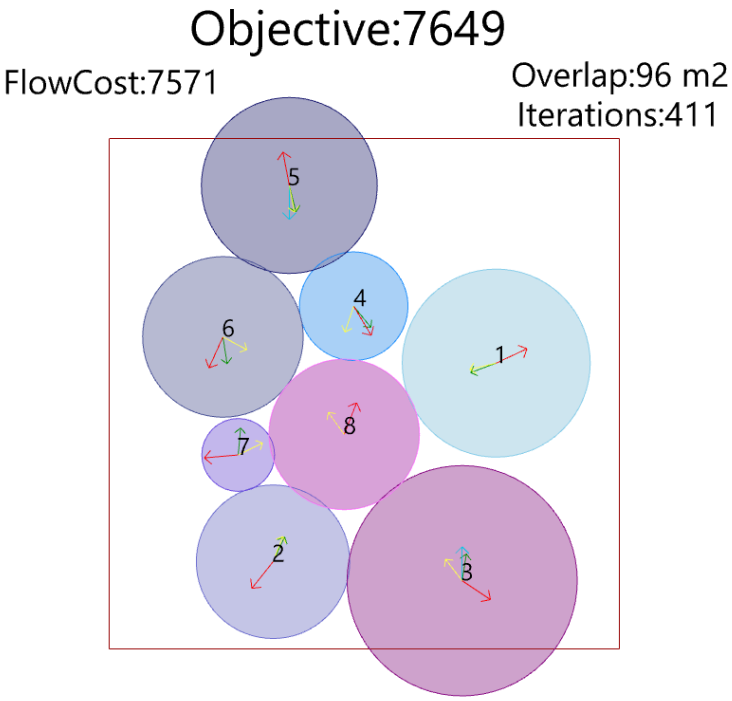


Figure D.1: The layout for sample nr 1 of the 8 department problem for the first-stage model considering the swapping procedure.

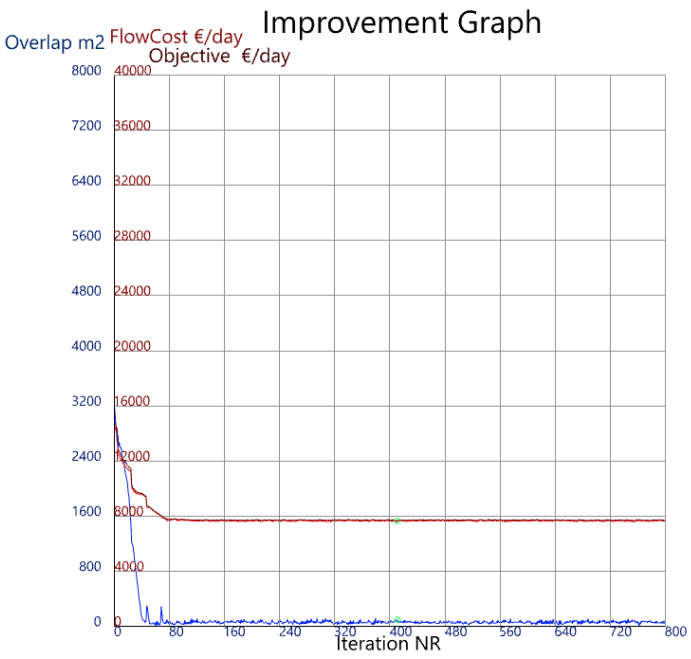


Figure D.2: The graph for sample nr 1 of the 8 department problem for the first-stage model considering the swapping procedure.

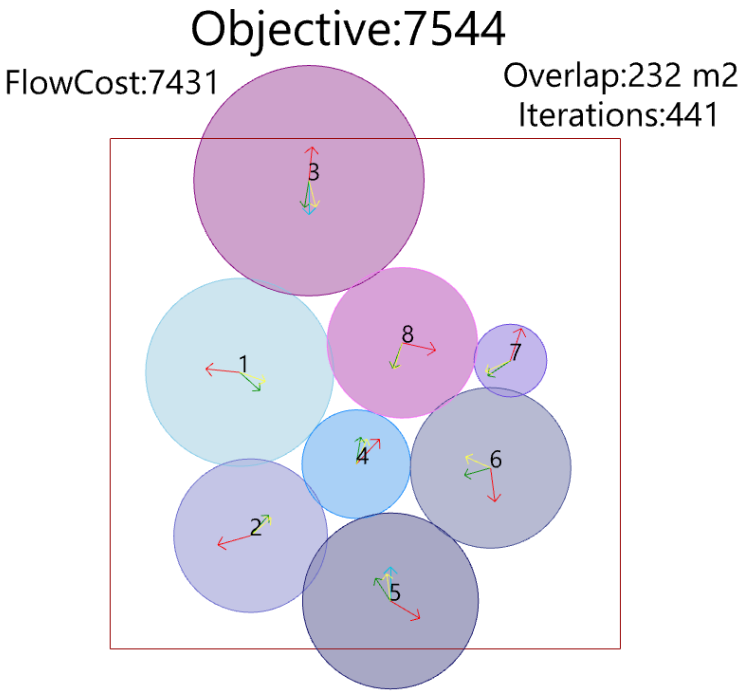


Figure D.3: The layout for sample nr 2 of the 8 department problem for the first-stage model considering the swapping procedure.

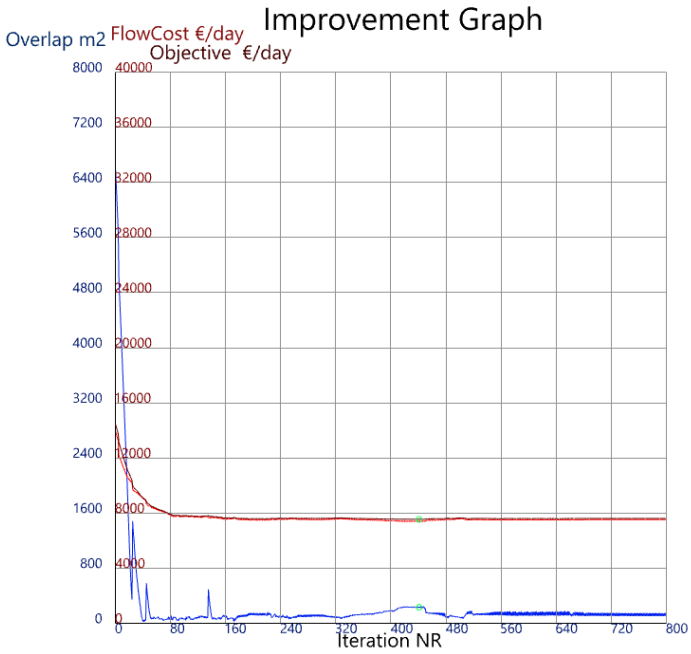


Figure D.4: The graph for sample nr 2 of the 8 department problem for the first-stage model considering the swapping procedure.

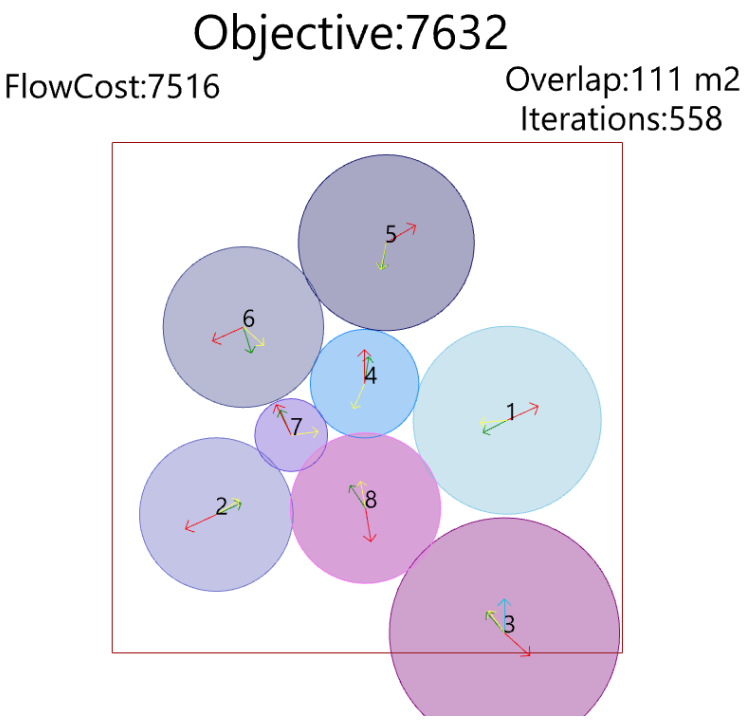


Figure D.5: The layout for sample nr 3 of the 8 department problem for the first-stage model considering the swapping procedure.

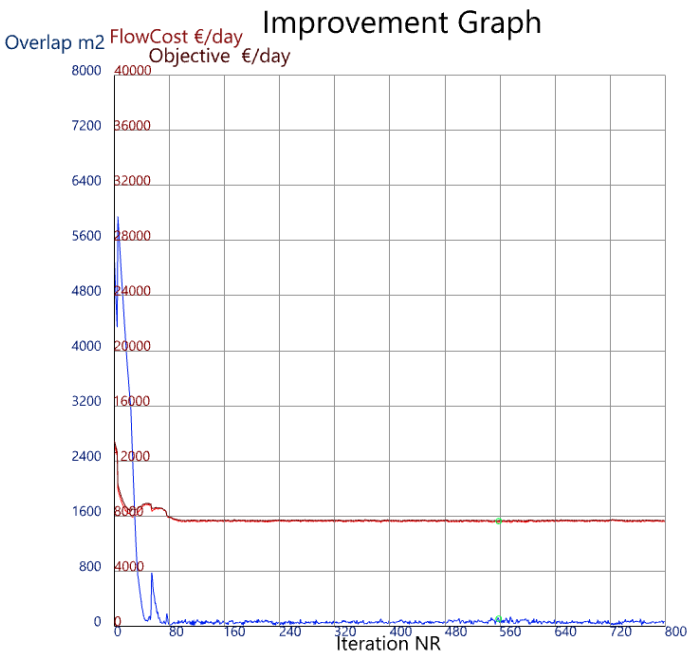


Figure D.6: The graph for sample nr 3 of the 8 department problem for the first-stage model considering the swapping procedure.

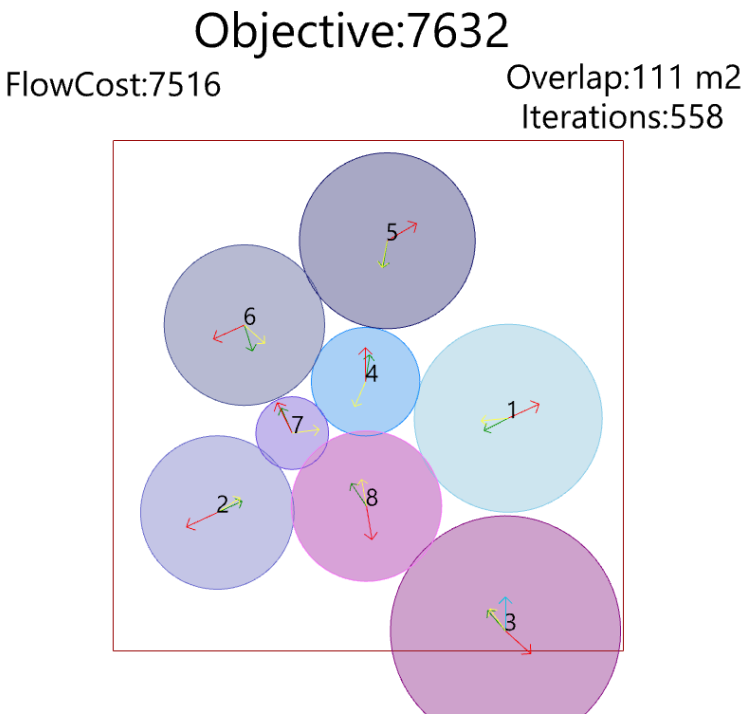


Figure D.7: The layout for sample nr 4 of the 8 department problem for the first-stage model considering the swapping procedure.

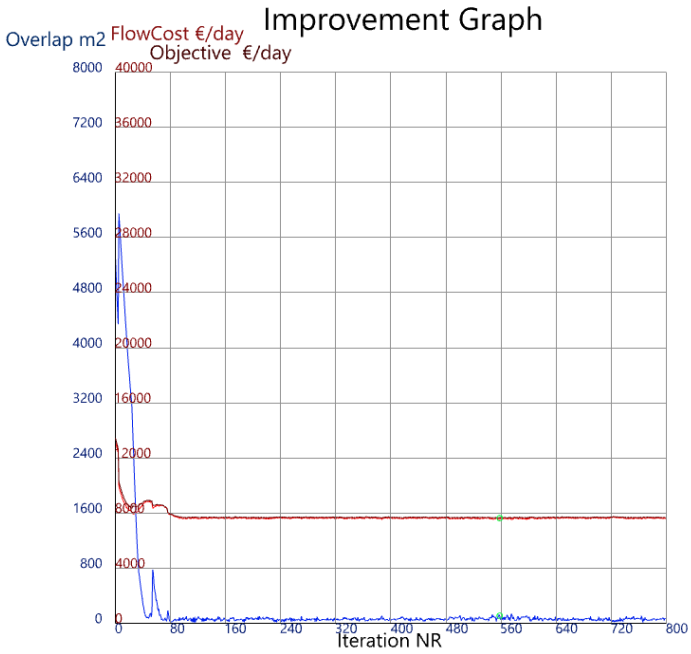


Figure D.8: The graph for sample nr 4 of the 8 department problem for the first-stage model considering the swapping procedure.

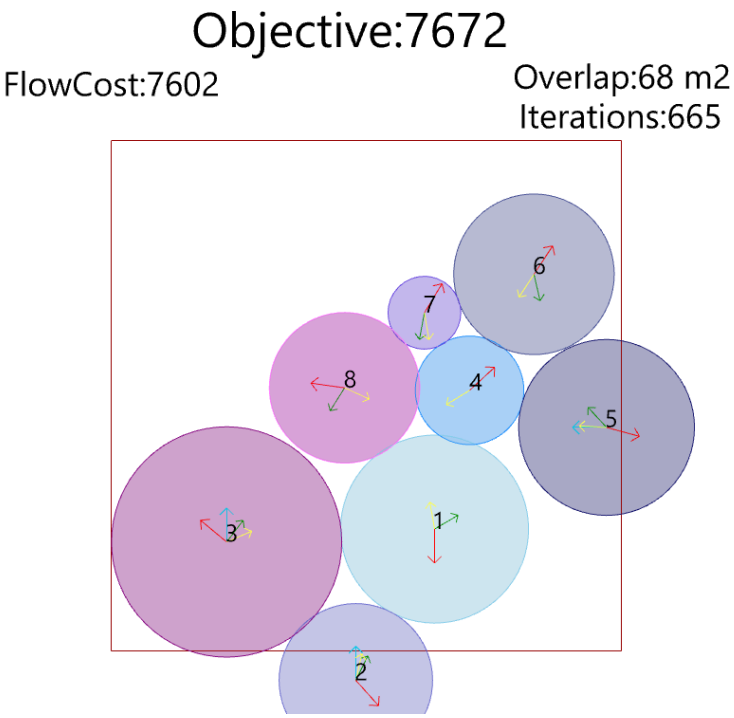


Figure D.9: The layout for sample nr 5 of the 8 department problem for the first-stage model considering the swapping procedure.

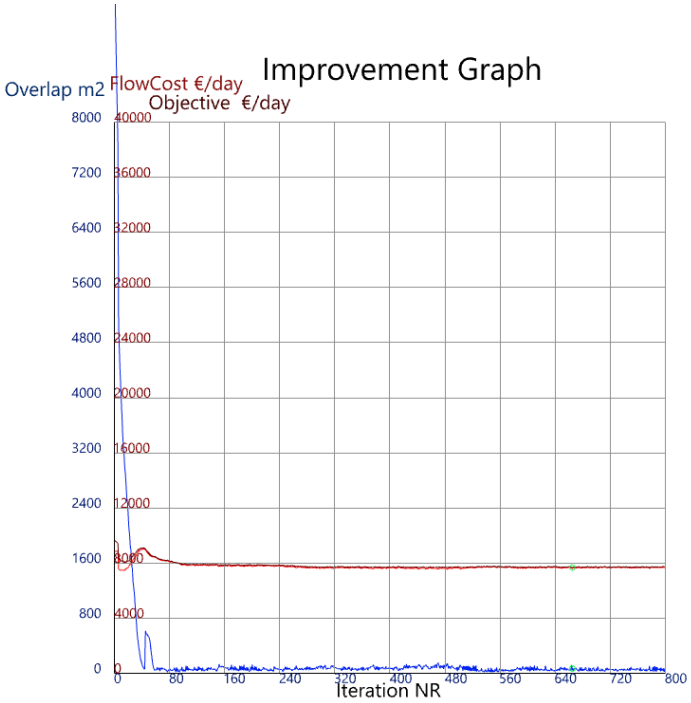


Figure D.10: The graph for sample nr 5 of the 8 department problem for the first-stage model considering the swapping procedure.

Appendix E: a sample of results for the shooting first-stage model

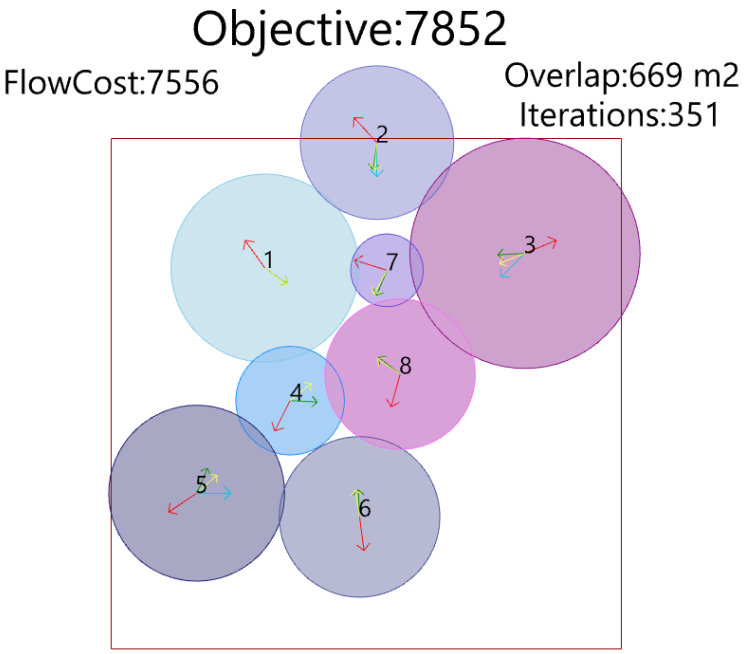


Figure E.1: The layout for sample nr 1 of the 8 department problem for the first-stage model considering the shooting procedure.

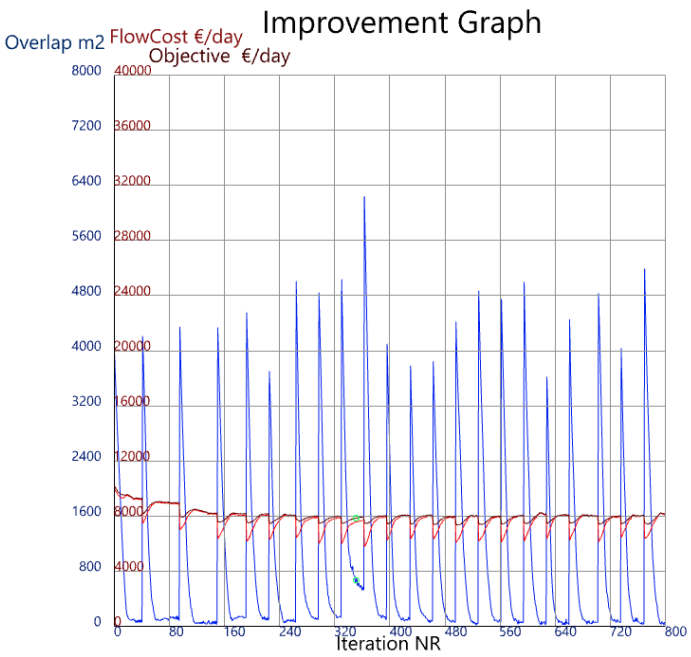


Figure E.2: The graph for sample nr 1 of the 8 department problem for the first-stage model considering the shooting procedure.

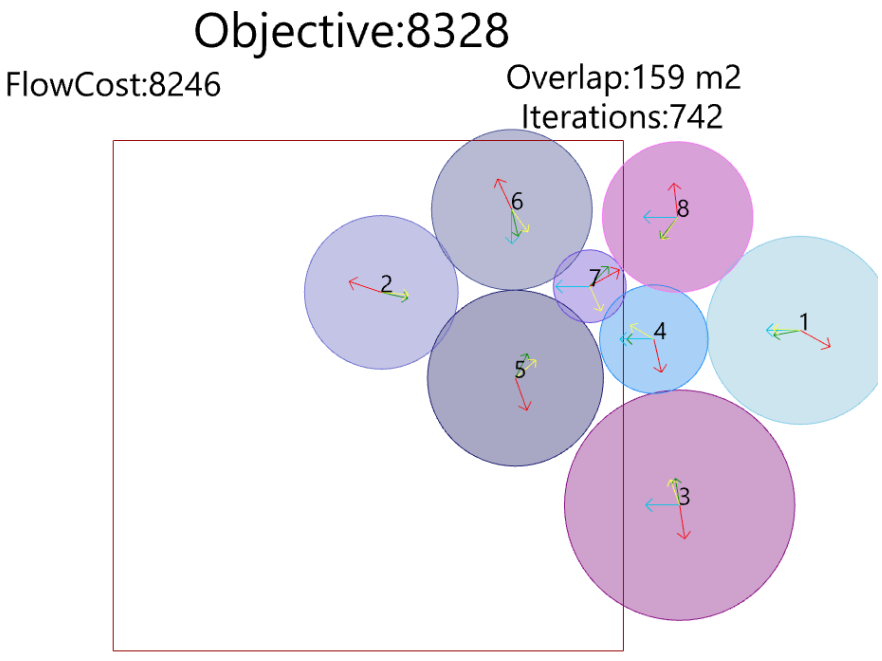


Figure E.3: The layout for sample nr 2 of the 8 department problem for the first-stage model considering the shooting procedure.

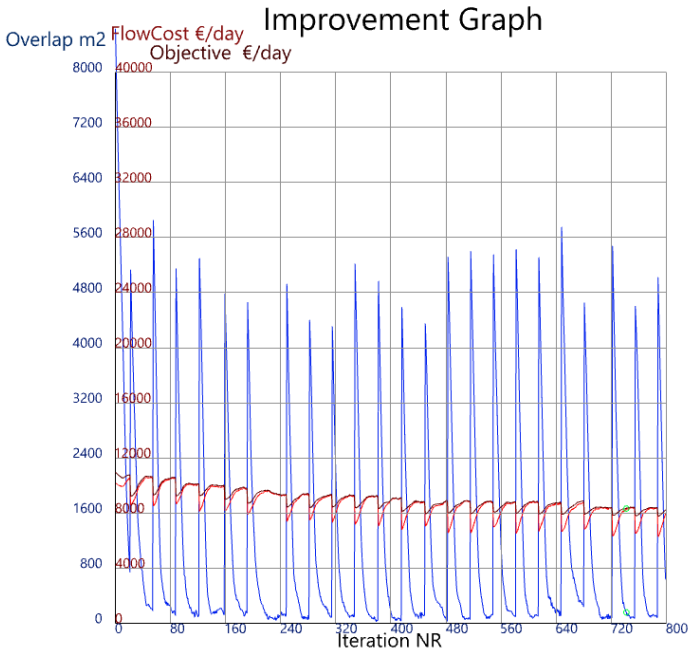


Figure E.4: The graph for sample nr 2 of the 8 department problem for the first-stage model considering the shooting procedure.

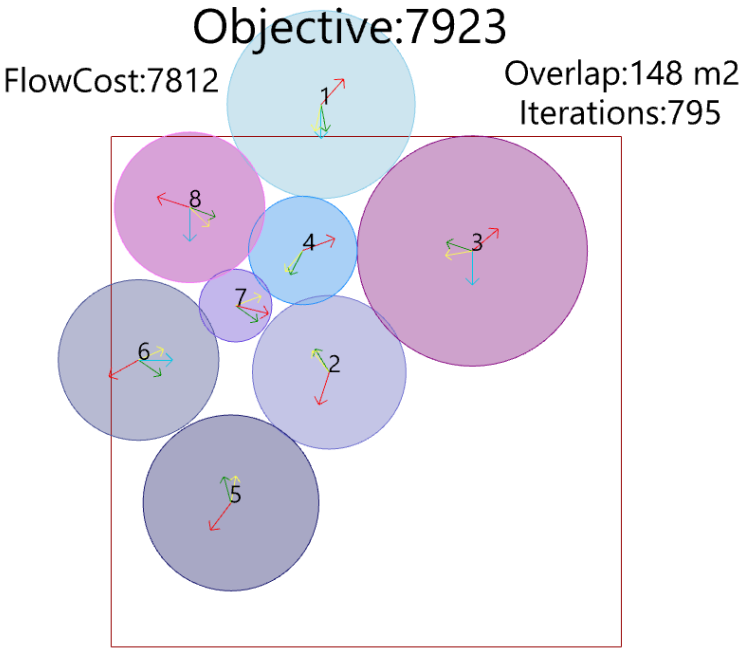


Figure E.5: The layout for sample nr 3 of the 8 department problem for the first-stage model considering the shooting procedure.

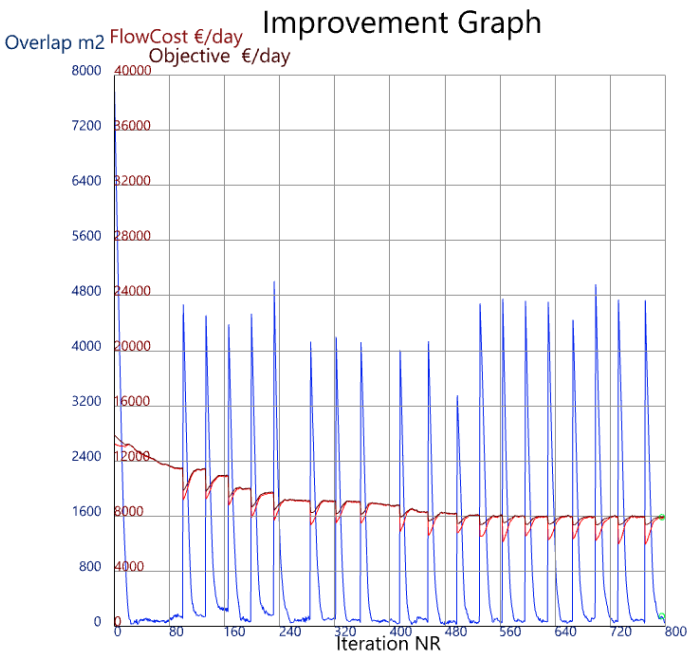


Figure E.6: The graph for sample nr 3 of the 8 department problem for the first-stage model considering the shooting procedure.

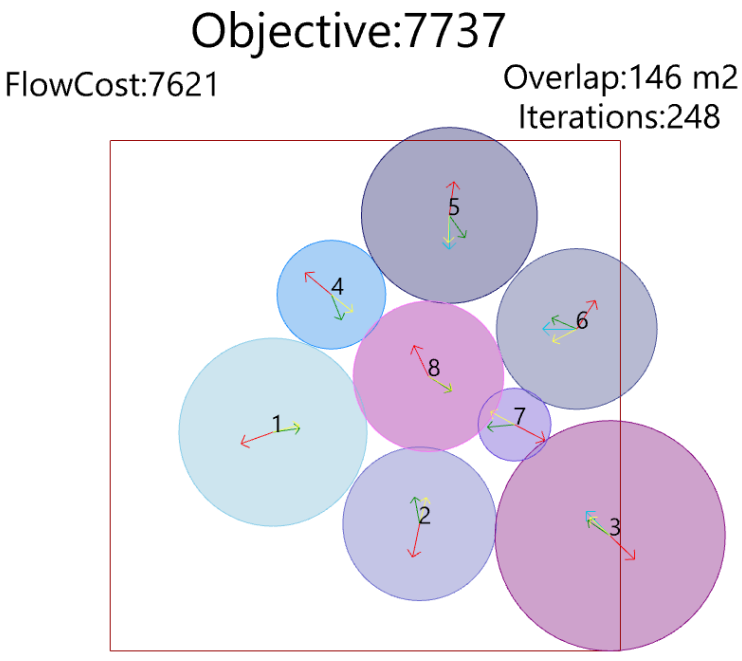


Figure E.7: The layout for sample nr 4 of the 8 department problem for the first-stage model considering the shooting procedure.

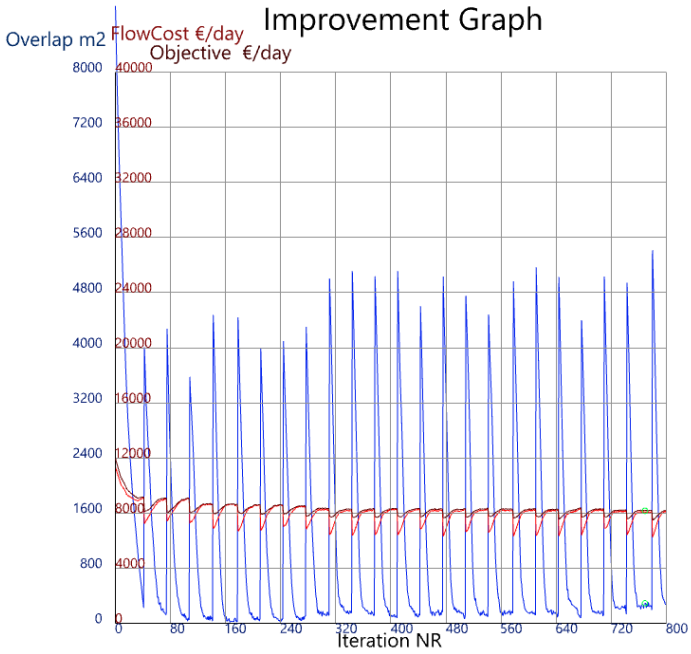


Figure E.8: The graph for sample nr 4 of the 8 department problem for the first-stage model considering the shooting procedure.

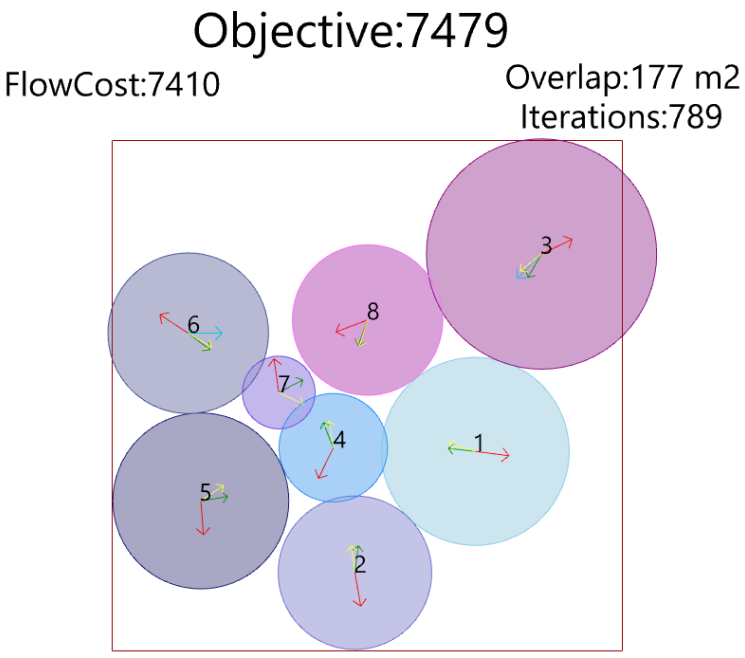


Figure E.9: The layout for sample nr 5 of the 8 department problem for the first-stage model considering the shooting procedure.

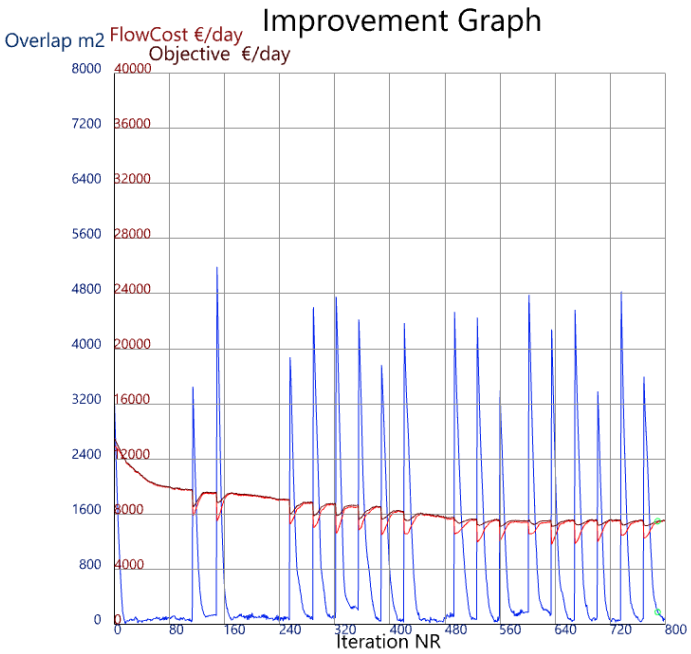


Figure E.10: The graph for sample nr 5 of the 8 department problem for the first-stage model considering the shooting procedure.

Appendix F: a sample of results for the swapping and shooting first-stage model

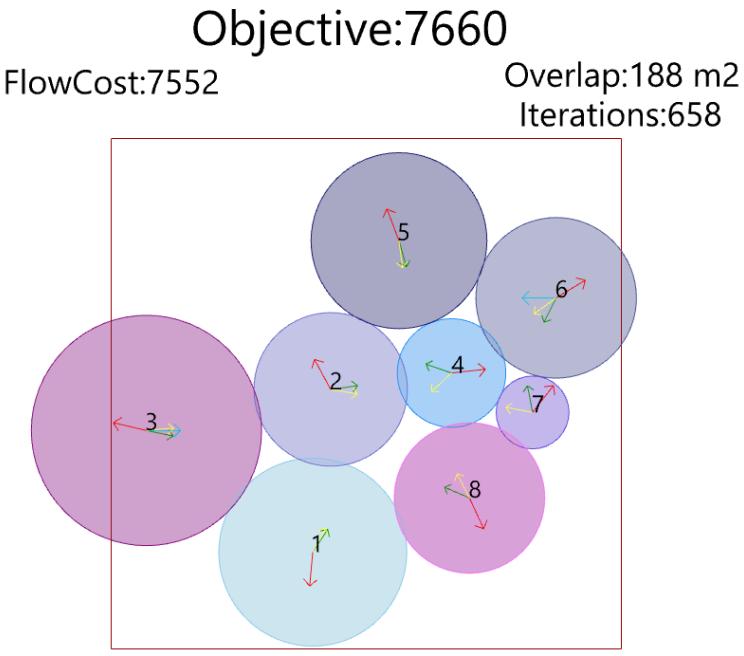


Figure F.1: The layout for sample nr 1 of the 8 department problem for the first-stage model considering the swapping and shooting procedure.

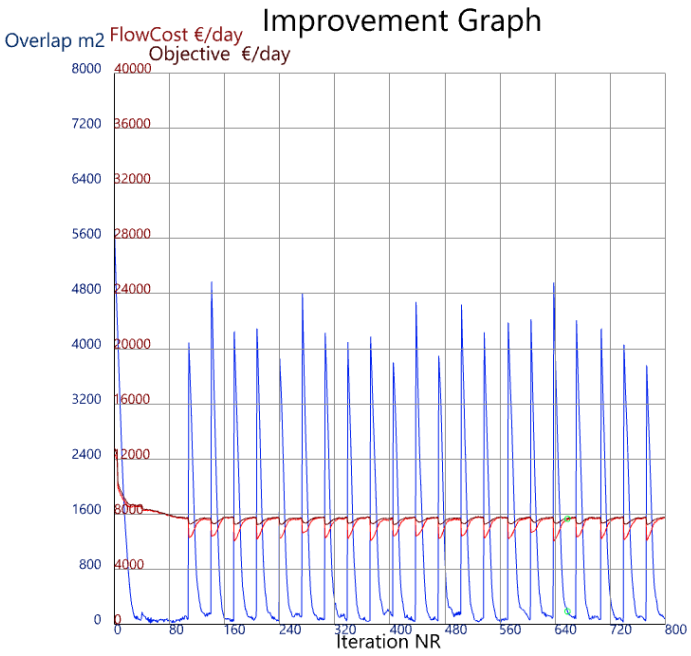


Figure F.2: The graph for sample nr 1 of the 8 department problem for the first-stage model considering the swapping and shooting procedure.

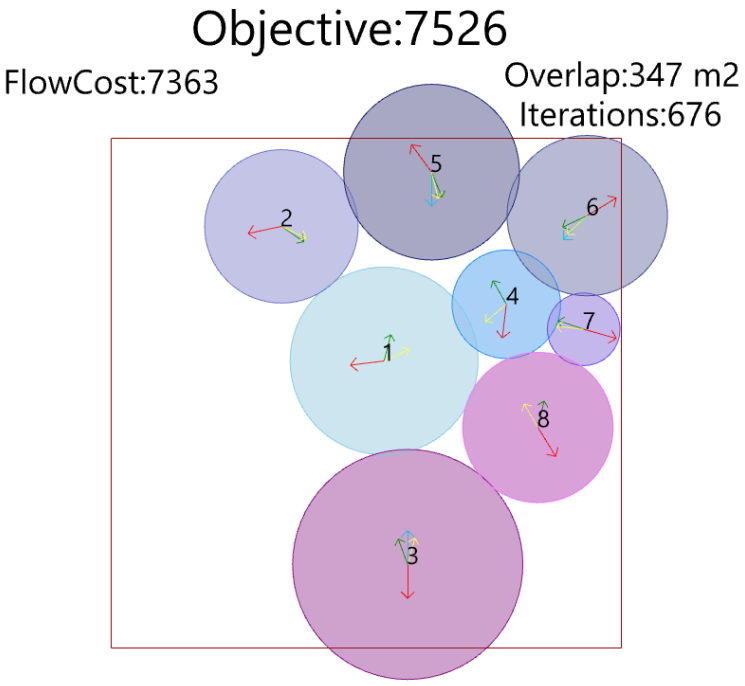


Figure F.3: The layout for sample nr 2 of the 8 department problem for the first-stage model considering the swapping and shooting procedure.

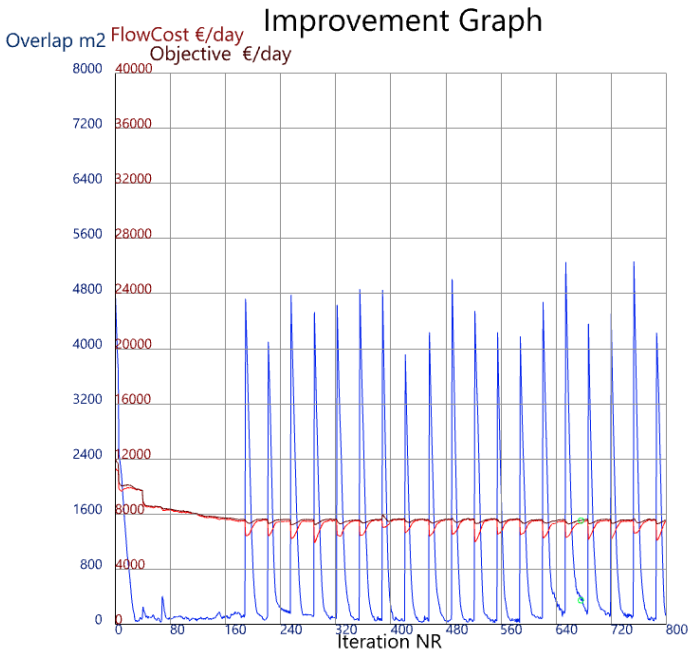


Figure F.4: The graph for sample nr 2 of the 8 department problem for the first-stage model considering the swapping and shooting procedure.

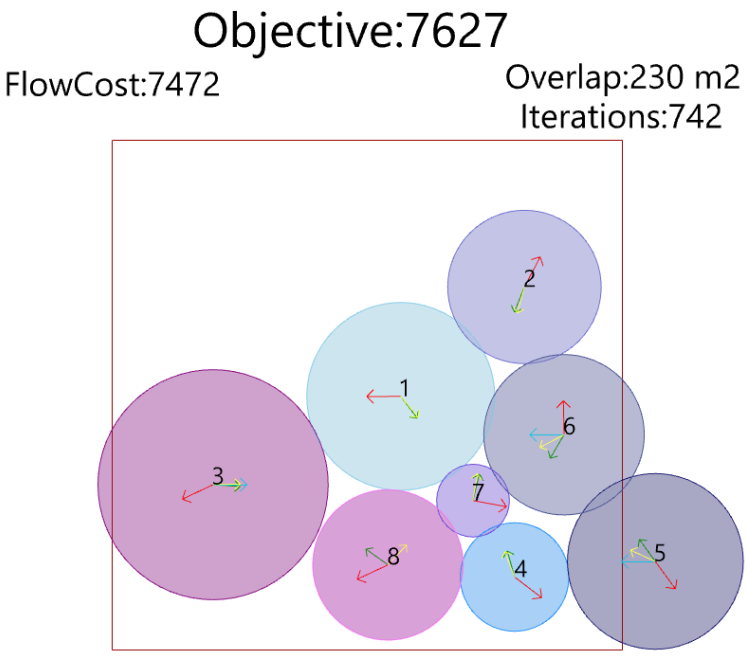


Figure F.5: The layout for sample nr 3 of the 8 department problem for the first-stage model considering the swapping and shooting procedure.

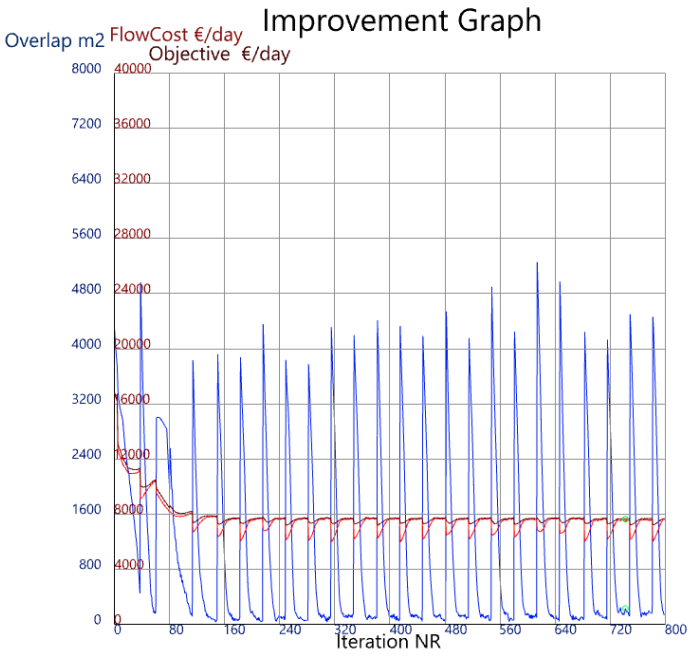


Figure F.6: The graph for sample nr 3 of the 8 department problem for the first-stage model considering the swapping and shooting procedure.

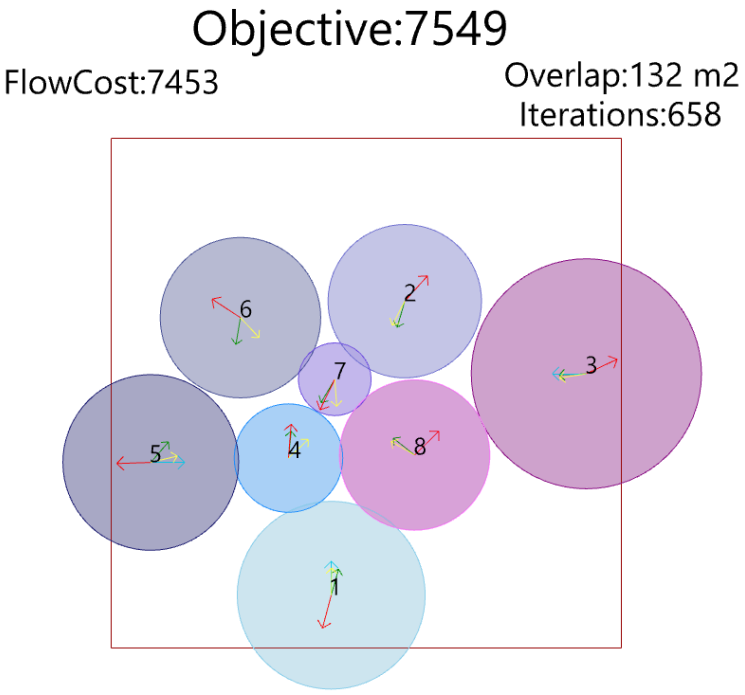


Figure F.7: The layout for sample nr 4 of the 8 department problem for the first-stage model considering the swapping and shooting procedure.

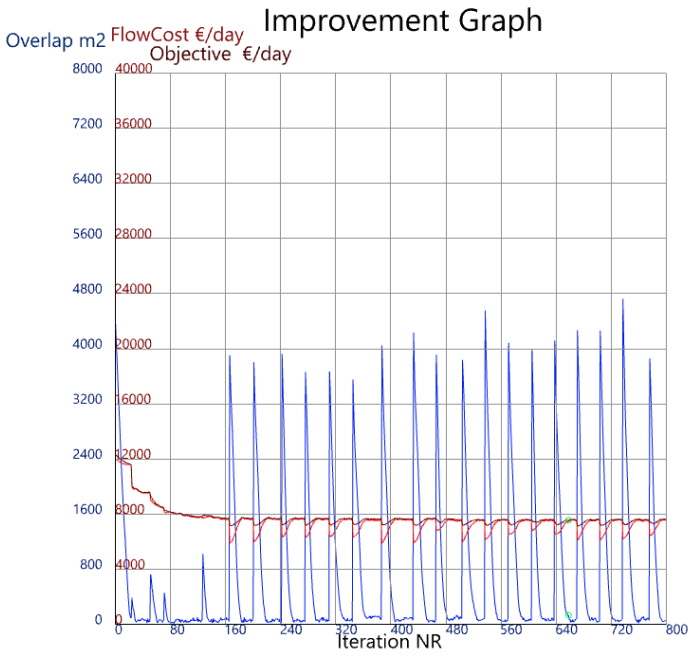


Figure F.8: The graph for sample nr 4 of the 8 department problem for the first-stage model considering the swapping and shooting procedure.

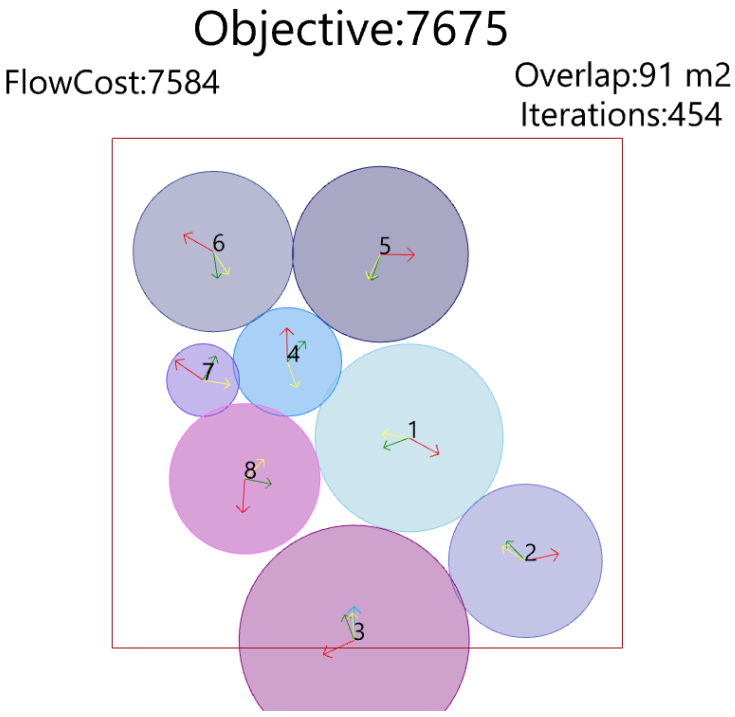


Figure F.9: The layout for sample nr 5 of the 8 department problem for the first-stage model considering the swapping and shooting procedure.

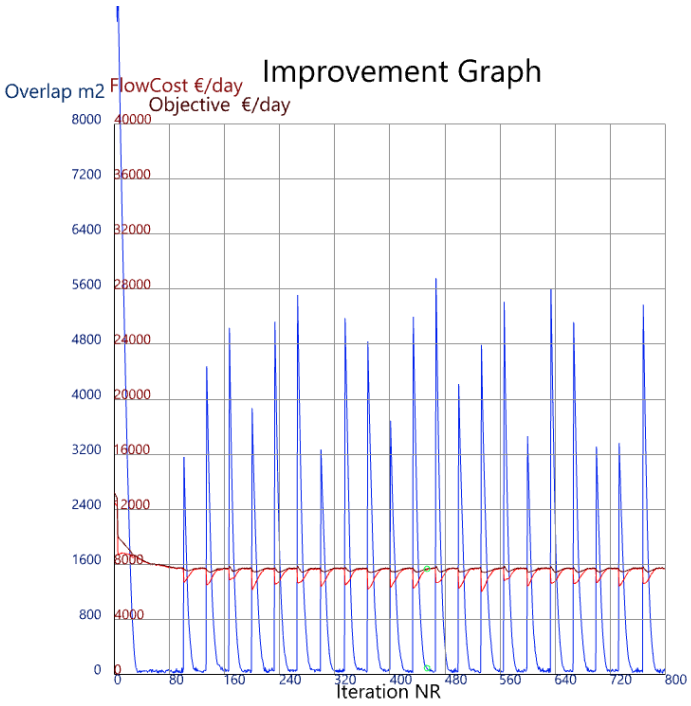


Figure F.10: The graph for sample nr 5 of the 8 department problem for the first-stage model considering the swapping and shooting procedure.

G

Appendix G: the results for all time-frames for the 8 department second-stage model

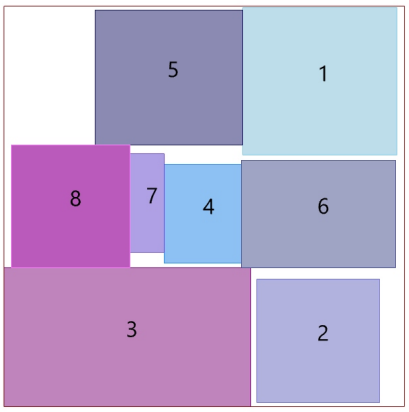


Figure G.1: The layout for the 8 department problem for the second-stage model excluding the first-stage model's constraints in 25 seconds.

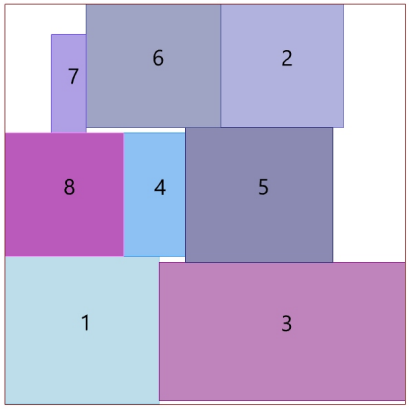


Figure G.2: The layout for the 8 department problem for the second-stage model including the first-stage model's constraints in 25 seconds.

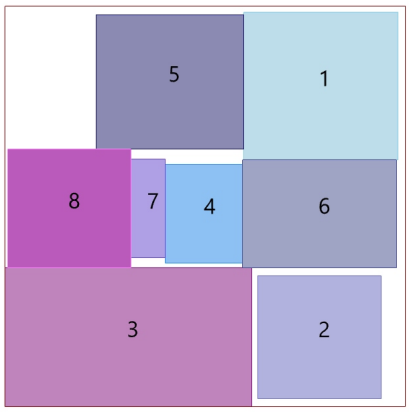


Figure G.3: The layout for the 8 department problem for the second-stage model excluding the first-stage model's constraints in 100 seconds.

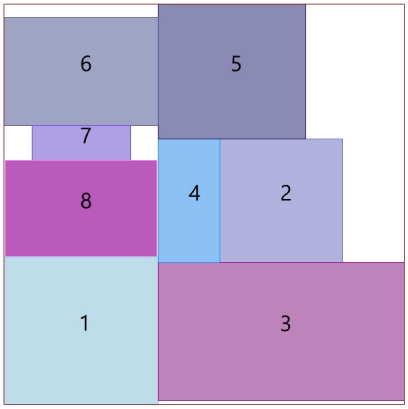


Figure G.4: The layout for the 8 department problem for the second-stage model including the first-stage model's constraints in 100 seconds.

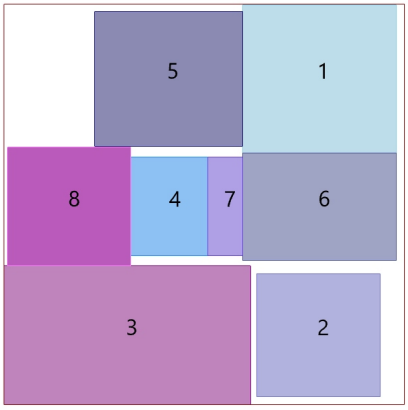


Figure G.5: The layout for the 8 department problem for the second-stage model excluding the first-stage model's constraints in 300 seconds.

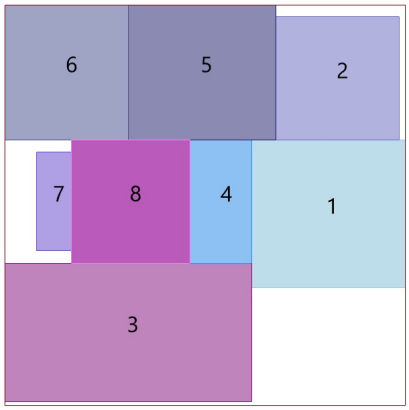


Figure G.6: The layout for the 8 department problem for the second-stage model including the first-stage model's constraints in 300 seconds.

Appendix H: the results for all time-frames for the 12 department second-stage model

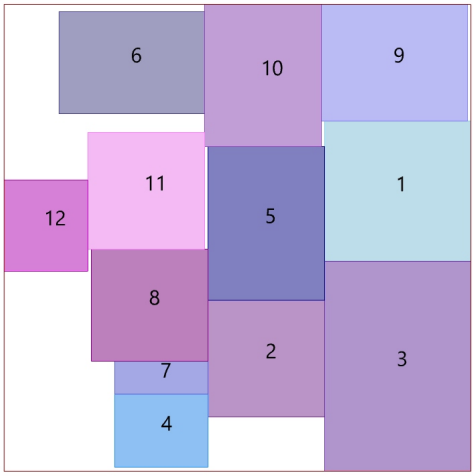


Figure H.1: The layout for the 12 department problem for the second-stage model excluding the first-stage model's constraints in 25 seconds.

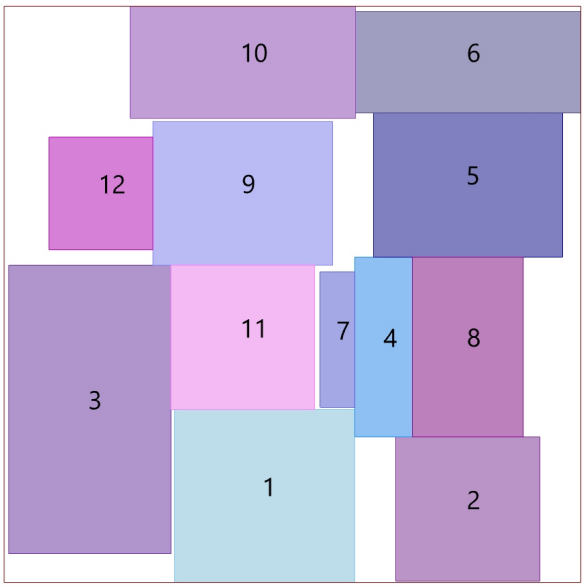


Figure H.2: The layout for the 12 department problem for the second-stage model including the first-stage model's constraints in 25 seconds.

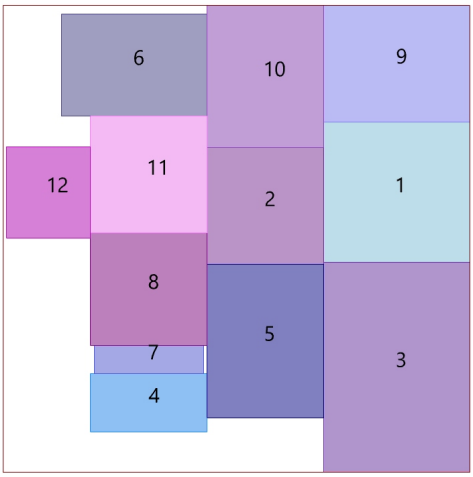


Figure H.3: The layout for the 12 department problem for the second-stage model excluding the first-stage model's constraints in 100 seconds.

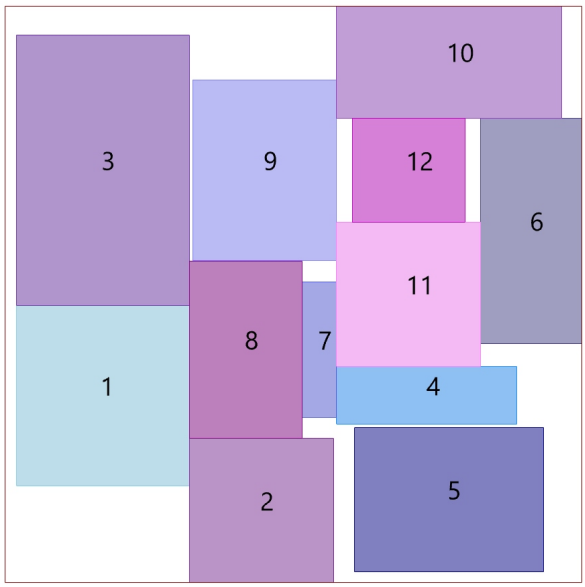


Figure H.4: The layout for the 12 department problem for the second-stage model including the first-stage model's constraints in 100 seconds.

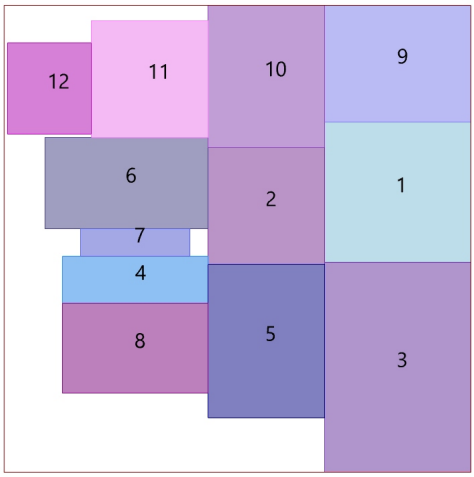


Figure H.5: The layout for the 12 department problem for the second-stage model excluding the first-stage model's constraints in 300 seconds.

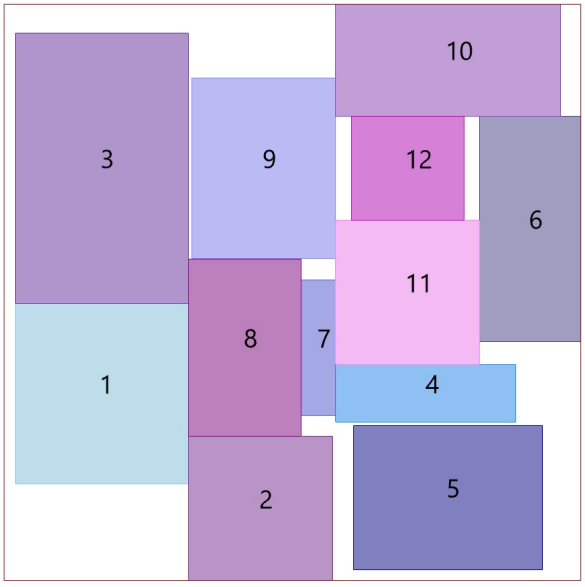


Figure H.6: The layout for the 12 department problem for the second-stage model including the first-stage model's constraints in 300 seconds.

Appendix I: first-stage model main python component

```
import ortools
import numpy as np
import pandas as pd
from pathlib import Path
import math
import random

def FindOverlapCenter(x1_l,y1_l,x1_r,y1_r,x2_l,y2_l,x2_r,y2_r):
   """a method to find the diagonal points of the intersecting area of 2 given rectangles and th
    x1 = max(x1_l,x2_l)
    y1 = max(y1_l,y2_l)
    x2 = min(x1_r,x2_r)
    y2 = min(y1_r,y2_r)
    center_x = ((abs(x1-x2))/2)+x1
    center_y = ((abs(y1-y2))/2)+y1
    center = (center_x,center_y)
    return center

def FindPoints(x1,y1,w1,h1):
   """A method to find the bottom left and top right points of a given rectangle"""
    dl_x= x1 -(0.5*w1)
    dl_y= y1 -(0.5*h1)
    tr_x= x1 +(0.5*w1)
    tr_y= y1 +(0.5*h1)
    points = (dl_x,dl_y,tr_x,tr_y)
    return points

def intersection_area(d, R, r):
    """Return the area of intersection of two circles.

    The circles have radii R and r, and their centres are separated by d.

    """

    if d <= abs(R-r):
        # One circle is entirely enclosed in the other.
        return np.pi * min(R, r)**2
```

```
    if d >= r + R:
        # The circles don't overlap at all.
        return 0

    r2, R2, d2 = r**2, R**2, d**2
    alpha = np.arccos((d2 + r2 - R2) / (2*d*r))
    beta = np.arccos((d2 + R2 - r2) / (2*d*R))
    return ( r2 * alpha + R2 * beta -
             0.5 * (r2 * np.sin(2*alpha) + R2 * np.sin(2*beta))
           )


#reading from the input excel files
path = Path("D:/bouwkunde/Master/thesis/Afterp4/test_problems.xlsx")
xlsx = pd.ExcelFile(path)
data = pd.read_excel(xlsx, "12_req")
flows = pd.read_excel(xlsx, "12_flows")
pre =pd.read_excel(xlsx, "pre")
#end reading excel files


#calculate the total area
total_area = 0
for i in range(len(data.index)):
    total_area=total_area+data.iloc[i,1]

#initiate random starting positions for all departments
x_coordinates = random_x
y_coordinates = random_y
#for i in range(len(data.index)):
    #x_coordinates_initiate.append(int(random_x[i]))
    #y_coordinates_initiate.append(int(random_y[i]))

data["x"]= x_coordinates #6
data["y"]= y_coordinates #7

#overwrite the starting location for fixed departments
for i in range(len(pre.index)):
    data.iloc[pre.iloc[i,0]-1,6]= (facility_width/100)*pre.iloc[i,1]
    data.iloc[pre.iloc[i,0]-1,7]= (facility_height/100)*pre.iloc[i,2]

#add all departments that are fixed into a list to later prevent them from moving
dont_move = []
for i in range(len(pre.index)):
    dont_move.append(pre.iloc[i,0]-1)

#transform data into a list
areas = data["Required Area"].values.tolist()

#compute the widths
radi=[]
for i in range(len(data.index)):
    area = data.iloc[i,1]
    radius = math.sqrt(area/math.pi)
    radi.append(radius)
data["widths"]=radi #8
```

```
#compute the heights
heights=[]
for i in range(len(radi)):
    heights.append(areas[i]/radi[i])
data[”heights”]=heights #9



##############################
#### HERE STARTS THE LOOP ####
##############################
x_max = []
x_max = []
overlaps = []
fitnesses = []
objectives = []
valid_objectives = []
temp_distances = []
swapping_iterations =[]
taboo = 0
counter = 1
trigger = 0
for k in range(iterations):
    #compute the vectors for departments i to departments j
    vectors = []
    vector_lists =[]

    for i in range(len(data.index)):
        #vector = np.array([0,0])
        dfdx= 0
        dfdy= 0
        for j in range(len(data.index)):
            if i != j:
                dx = data.iloc[j,6] - data.iloc[i,6]
                dy = data.iloc[j,7] - data.iloc[i,7]
                dxy = math.sqrt((dx)**2+((dy)**2))
                if radi[i]+radi[j]-dxy>=0:
                    pass
                else:
                    flip = -1
                    xi = data.iloc[i,6]
                    xj = data.iloc[j,6]
                    yi = data.iloc[i,7]
                    yj = data.iloc[j,7]
                    flow = int(flows.iloc[i,(j+1)])
                    print(flow)
               dfdx_add = ((xi-xj)*flow)/(math.sqrt((xi-xj)**2+(yi-yj)**2))
               dfdy_add = ((yi-yj)*flow)/(math.sqrt((xi-xj)**2+(yi-yj)**2))
                    dfdx = dfdx + dfdx_add
                    dfdy = dfdy + dfdy_add

        vector = np.array([flip*dfdx,flip*dfdy])
        vector_list = list(vector)

                #vectoradd = np.array([dx,dy])
```

```
            #multiplier = int(flows.iloc[i,(j+1)])
            #vectoradd = vectoradd*multiplier
            #vector =  vectoradd+ vector
            #vector_list = list(vector)
    vectors.append(vector)
    vector_lists.append(vector_list)
    print(vector)
data[”vectors”]= vectors #10

#compute the repelling vectors for overlapping
overlap = []
vectors_o =[]
vector_o_lists =[]
for i in range(len(data.index)):
    vector_o = np.array([0,0])
    for j in range(len(data.index)):
        if i!= j:
            dx = data.iloc[j,6] - data.iloc[i,6]
            dy = data.iloc[j,7] - data.iloc[i,7]
            dxy = math.sqrt((dx)**2+((dy)**2))
            overlap_area = intersection_area(dxy, radi[i], radi[j])
      points1 = FindPoints(data.iloc[i,6],data.iloc[i,7],data.iloc[i,8],data.iloc
      points2 = FindPoints(data.iloc[j,6],data.iloc[j,7],data.iloc[j,8],data.iloc
      center = FindOverlapCenter(points1[0],points1[1],points1[2],points1[3],poin
            vectoradd = np.array([dx,dy])
            vectoradd = vectoradd * overlap_area * -1
            vector_o = vector_o + vectoradd
            overlap.append(overlap_area)
    vector_o_list = list(vector_o)
    vectors_o.append(vector_o)
    vector_o_lists.append(vector_o_list)
overlaps.append(sum(overlap))
data[”vectors_o”]= vectors_o #11

#compute the repelling vectors for overlapping with outside facility
vectors_out =[]
vector_out_lists =[]
for i in range(len(data.index)):
    vector_out = np.array([0,0])
    if data.iloc[i,6] - radi[i]<0:
        vector_out = vector_out + np.array([1,0])
    if data.iloc[i,6] + radi[i] > facility_width:
        vector_out = vector_out + np.array([-1,0])
    if data.iloc[i,7] - radi[i]<0:
        vector_out = vector_out + np.array([0,1])
    if data.iloc[i,7] + radi[i] > facility_height:
        vector_out = vector_out + np.array([0,-1])
    vector_out_list = list(vector_out)
    vectors_out.append(vector_out)
    vector_out_lists.append(vector_out_list)

data[”vectors_out”]=vectors_out #12

#calculating the objective
objctv = 0
fitness = 0
```

```
    cost = 1
    for i in range(len(data.index)):
        for j in range(len(data.index)):
            if i != j:
                dx = abs(data.iloc[j,6] - data.iloc[i,6])
                dy = abs(data.iloc[j,7] - data.iloc[i,7])
                dyx = math.sqrt((dx)**2+((dy)**2))
                overlap_ij = data.iloc[i,8]+data.iloc[j,8] - dyx
                if overlap_ij <0:
                    overlap_ij = 0
                flow = int(flows.iloc[i,(j+1)])
                objctv_add = (dyx+overlap_ij) * cost * flow
                fitness_add = (dyx) * cost * flow
                objctv = objctv + objctv_add
                fitness = fitness + fitness_add
    objective = objctv
    objectives.append(objective)
    fitnesses.append(fitness)
    if taboo ==0:
        valid_objectives.append(objective)
    else:
        valid_objectives.append(404404)

    #saving x and y if the objective is the current minimum
    #and save maximum fitness
    #but only if there has not been swapped recently
    if taboo == 0:
        if objective == min(valid_objectives):
            x_max = data[”x”].values.tolist()
            y_max = data[”y”].values.tolist()
            objective_max = objective
            fitness_max = fitness
            vector_lists_max = vector_lists
            vector_o_lists_max = vector_o_lists
            vector_temp_lists_max = vector_temp_lists
            temp_distance_max = temp_distance
            vector_out_lists_max = vector_out_lists
            overlap_max=sum(overlap)
            k_max = k

    #check for trigger for shooting and update the taboo counter
    if taboo != 0:
        taboo = taboo -1
    if k > 2*shoot and taboo == 0:
        objective_trigger = valid_objectives[i-(2*shoot)]
        if objective >= objective_trigger:
            trigger = 1

    #move the departments normally, OR when the trigger is active
   #have the departments ignore all overlapping vectors and multiply the movement by 10 SHOOTIN
    if trigger == 1:
        for i in range(len(data.index)):
            if i not in dont_move:
                #attraction
                if data.iloc[i,10][0] != 0 and data.iloc[i,10][1]!=0:
                    x_10=data.iloc[i,10][0]
```

```
                y_10=data.iloc[i,10][1]
                xy_10= math.sqrt((x_10)**2+((y_10)**2))
                vector_10= np.array([(x_10/xy_10),(y_10/xy_10)])
            else:
                vector_10= np.array([0,0])
            #temperature differences
            x_13=data.iloc[i,13][0]
            y_13=data.iloc[i,13][1]
            xy_13= math.sqrt((x_13)**2+((y_13)**2))
            vector_13= np.array([(x_13/xy_13),(y_13/xy_13)])
            #mastervector
        mastervector = vector_10 * 1*stepsize #+  vector_13 * 0 *stepsize
            x_m = mastervector[0]
            y_m = mastervector[1]
            if x_m == 0 or y_m ==0:
                pass
            else:
                xy_m = math.sqrt((x_m)**2+((y_m)**2))
                data.iloc[i,6]= data.iloc[i,6]+((x_m)/xy_m)*shoot
                data.iloc[i,7]= data.iloc[i,7]+((y_m)/xy_m)*shoot
    trigger = 0
    taboo = 2*shoot

else:
    for i in range(len(data.index)):
        if i not in dont_move:
            #attraction
            if data.iloc[i,10][0] != 0 and data.iloc[i,10][1]!=0:
                x_10=data.iloc[i,10][0]
                y_10=data.iloc[i,10][1]
                xy_10= math.sqrt((x_10)**2+((y_10)**2))
                vector_10= np.array([(x_10/xy_10),(y_10/xy_10)])
            else:
                vector_10= np.array([0,0])
            #overlap
            if data.iloc[i,11][0] != 0 and data.iloc[i,11][1]!=0:
                x_11=data.iloc[i,11][0]
                y_11=data.iloc[i,11][1]
                xy_11= math.sqrt((x_11)**2+((y_11)**2))
                vector_11= np.array([(x_11/xy_11),(y_11/xy_11)])
            else:
                vector_11= np.array([0,0])
            #overlap outside facility
            if data.iloc[i,12][0] != 0 and data.iloc[i,12][1]!=0:
                x_12=data.iloc[i,12][0]
                y_12=data.iloc[i,12][1]
                xy_12= math.sqrt((x_12)**2+((y_12)**2))
                vector_12= np.array([(x_12/xy_12),(y_12/xy_12)])
            else:
                vector_12= np.array([0,0])
            #temperature differences
            x_13=data.iloc[i,13][0]
            y_13=data.iloc[i,13][1]
            xy_13= math.sqrt((x_13)**2+((y_13)**2))
            vector_13= np.array([(x_13/xy_13),(y_13/xy_13)])
            #mastervector
```

```
        mastervector = vector_10 * 1*stepsize + vector_11 * stepsize_o  + vector_12 *stepsi
            x_m = mastervector[0]
            y_m = mastervector[1]
            xy_m = math.sqrt((x_m)**2+((y_m)**2))
            data.iloc[i,6]= data.iloc[i,6]+((x_m)/xy_m)
            data.iloc[i,7]= data.iloc[i,7]+((y_m)/xy_m)


 #check wether the objective is improving fast enough to enable swapping
 trigger2 = False
 if k>3 and taboo == 0:
     if objectives[k-1]/objective > 1.005:
         trigger2 = True

#make the most profitable swap of two departments if the objective is improving by more than
 flag3 = False
 if trigger2 == True:
     max_fitness = 100000
     for i in range(len(data.index)):
         for j in range(len(data.index)):
             if i!=j and i not in dont_move and j not in dont_move:
                 #copy the dataframe
                 data2= data.copy()
                 #swap two departments location
                 buffer_x = data2.iloc[i,6]
                 buffer_y = data2.iloc[i,7]
                 data2.iloc[i,6]= data2.iloc[j,6]
                 data2.iloc[i,7]= data2.iloc[j,7]
                 data2.iloc[j,6]= buffer_x
                 data2.iloc[j,7]= buffer_y

                 #calculating the fitness
                 fitness_swap_ij = 0
                 cost = 1
                 for l in range(len(data2.index)):
                     for m in range(len(data2.index)):
                         if l != m:
                             dx = data2.iloc[m,6] - data2.iloc[l,6]
                             dy = data2.iloc[m,7] - data2.iloc[l,7]
                             dyx = math.sqrt((dx)**2+((dy)**2))
                             flow = int(flows.iloc[l,(m+1)])
                             fitness_add = dyx * cost * flow
                         fitness_swap_ij = fitness_swap_ij + fitness_add

             #calculate FITNESS and save if its an improvement over the current FITNESS (this
              if fitness_swap_ij < fitness and fitness_swap_ij <max_fitness:
                         max_fitness= fitness_swap_ij
                         save_i = i
                         save_j = j
                         save_i = int(save_i)
                         save_j = int(save_j)
                         flag3 = True
     #actually swap the most profitable swap
     if flag3 == True:
         b_x = data.iloc[save_i,6]
         b_y = data.iloc[save_i,7]
```

```
            data.iloc[save_i,6]= data.iloc[save_j,6]
            data.iloc[save_i,7]= data.iloc[save_j,7]
            data.iloc[save_j,6]= b_x
            data.iloc[save_j,7]= b_y
            taboo = shoot * 2
            swapping_iterations.append(k)
        fitness_swap_sisj = 0
        cost = 1
        for l in range(len(data.index)):
            for m in range(len(data.index)):
                if l != m:
                    dx = data.iloc[m,6] - data.iloc[l,6]
                    dy = data.iloc[m,7] - data.iloc[l,7]
                    dyx = math.sqrt((dx)**2+((dy)**2))
                    flow = int(flows.iloc[l,(m+1)])
                    fitness_add = dyx * cost * flow
                    fitness_swap_sisj = fitness_swap_sisj + fitness_add
```

Appendix J: second-stage model main python component

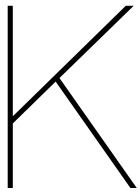
```
from ortools.sat.python import cp_model
from ortools.linear_solver import pywraplp
from pathlib import Path
import numpy as np
import pandas as pd

def SimpleSatProgram(data,lc,flows,FW,FH,minwh,s):
    """Minimal CP-SAT example to showcase calling the solver."""
    # Creates the model.
    model = cp_model.CpModel()


    # Creates the variables.
    x_vars = [model.NewIntVar(0,(FW), 'x%i' % i)
            for i in range(aa)]
    y_vars = [model.NewIntVar(0,(FH), 'y%i' % i)
            for i in range(aa)]
  heights = [model.NewIntVar(((minwh[i]-1)*(s-5)),(minwh[i]*int(data.iloc[i,3]*(s*s))), 'h
            for i in range(aa)]
  widths = [model.NewIntVar(((minwh[i]-1)*(s-5)),(minwh[i]*int(data.iloc[i,3]*(s*s))), 'wi
            for i in range(aa)]



    # include the constraint department width height constraints
    # make booleans first
    b_ar = []
    for i in range(aa):
        b_ar.append(model.NewBoolVar('b_ar%i'%i))

    ar_cons = []
    ar_cons2 = []
    for i in range(aa):
        if data.iloc[i,2]==1:
            f = (data.iloc[i,3]*s)
           ar_cons.append(model.Add(heights[i]==f).OnlyEnforceIf(b_ar[i]))
```

```
        ar_cons2.append(model.Add(widths[i]==f).OnlyEnforceIf(b_ar[i].Not()))


    # Create a boolean variables to ensure the abs() nonlinear formulation at the overlap c

     b = []
     b2 = []
     b3 = []
     for i in range(aa):
         c =[]
         c2 = []
         c3 = []
         for j in range(aa):
                 c.append(model.NewBoolVar('b%i%i' %( i,j)))
                 c2.append(model.NewBoolVar('b2_%i%i' %( i,j)))
                 c3.append(model.NewBoolVar('b3_%i%i' %( i,j)))
         b.append(c)
         b2.append(c2)
         b3.append(c3)

    # Create constraints that allow to see wether i>j or j>i depending on wether the boolea
     i_bt_j = []
     i_st_j = []
     i_bt_j2 = []
     i_st_j2 = []
     for i in range(aa):
         d = []
         e = []
         d2 = []
         e2 = []
         for j in range(aa):
             if i>j:
            d.append(model.Add(x_vars[i]>=x_vars[j]).OnlyEnforceIf(b[i][j]))
            e.append(model.Add(x_vars[j]>x_vars[i]).OnlyEnforceIf(b[i][j].Not()))
            d2.append(model.Add(y_vars[i]>=y_vars[j]).OnlyEnforceIf(b2[i][j]))
            e2.append(model.Add(y_vars[j]>y_vars[i]).OnlyEnforceIf(b2[i][j].Not()))
         i_bt_j.append(d)
         i_st_j.append(e)
         i_bt_j2.append(d2)
         i_st_j2.append(e2)

     # Creates the area constraints.
     area_cons = [0] *aa
     for i in range(aa):
         eq = data.iloc[i,1]*s*s
      area_cons[i] = model.AddMultiplicationEquality(eq,[widths[i],heights[i]])


     # Creates the no overlap constraints.
     overlap_cons = []
     overlap_cons2 = []
     overlap_cons3 = []
     overlap_cons4 = []
     overlap_cons5 = []
     overlap_cons6 = []
     for i in range(aa):
```

```
        ov1 = []
        ov2 = []
        ov3 = []
        ov4 = []
        ov5 = []
        ov6 = []

        for j in range(aa):
            if i>j:
           #these constraints make sure that boolean b3 True means that there is overlap in x d
              #constraint b makes sure that the absolute value becomes linear
                        ov1.append(model.Add(-2*(x_vars[i]-x_vars[j])<=  -
1*(widths[i]+widths[j])).OnlyEnforceIf([b[i][j],b3[i][j].Not()]))#i>j, no overlap
                        ov2.append(model.Add(-2*(x_vars[j]-x_vars[i])<=  -
1*(widths[i]+widths[j])).OnlyEnforceIf([b[i][j].Not(),b3[i][j].Not()]))#i<j, no overlap
           ov3.append(model.Add(2*(x_vars[i]-x_vars[j])<= 1*(widths[i]+widths[j])).OnlyEnfo
           ov4.append(model.Add(2*(x_vars[j]-x_vars[i])<= 1*(widths[i]+widths[j])).OnlyEnfo
                        ov5.append(model.Add(-2*(y_vars[i]-y_vars[j])<=  -
1*(heights[i]+heights[j])).OnlyEnforceIf([b2[i][j],b3[i][j]]))
                        ov6.append(model.Add(-2*(y_vars[j]-y_vars[i])<=  -
1*(heights[i]+heights[j])).OnlyEnforceIf([b2[i][j].Not(),b3[i][j]]))

             #or (- 2*(y_vars[i]-y_vars[j])<= -(heigths[i]+heigths[j]) and -
1*(- 2*(y_vars[i]-y_vars[j]))<= -(heigths[i]+heigths[j]) ) or (- 2*(y_vars[j]-
y_vars[i])<= -(heigths[i]+heigths[j]) and -1*(- 2*(y_vars[j]-y_vars[i]))<= -
(heigths[i]+heigths[j]) ))
        overlap_cons.append(ov1)
        overlap_cons2.append(ov2)
        overlap_cons3.append(ov3)
        overlap_cons4.append(ov4)
        overlap_cons5.append(ov5)
        overlap_cons6.append(ov6)

    # Creates the no overlap with facility constraints
    overlap_facility_cons = []
    for i in range(aa):
        overlap_facility_cons.append(model.Add(widths[i]<=2*x_vars[i]))
       overlap_facility_cons.append(model.Add(widths[i]<=2*(FW-x_vars[i])))
        overlap_facility_cons.append(model.Add(heights[i]<=2*y_vars[i]))
      overlap_facility_cons.append(model.Add(heights[i]<=2*(FH-y_vars[i])))

    # Create the positional constraints that were found in the first model!

    """
    xy_cons = []
    for i in range(aa):
        xy = []
        for j in range(aa):#dont forget j+1
            if i<j:
                nr = int(lc.iloc[i,j+1])
                if nr == 0:
                    xy.append(model.Add(x_vars[i]<x_vars[j]))
                elif nr == 1:
                    xy.append(model.Add(y_vars[i]>y_vars[j]))
                elif nr == 2:
                    xy.append(model.Add(x_vars[i]>x_vars[j]))
```

```
                elif nr == 3:
                    xy.append(model.Add(y_vars[i]<y_vars[j]))
        xy_cons.append(xy)
    """

    # Creates the objective
    #first create variables that are either xi>xj when xi>xj or xj>xi when xj>xi (same for
    constants_x= []
    constants_y= []
    for i in range(aa):
        constants_x_add =[]
        constants_y_add =[]
        for j in range(aa):
            if i >j:
               constants_x_add.append(model.NewIntVar(-FW,FW,'xb%i%i' %( i,j)))
               constants_y_add.append(model.NewIntVar(-FH,FH,'yb%i%i' %( i,j)))
        constants_x.append(constants_x_add)
        constants_y.append(constants_y_add)

    #now create constraints setting the sort of boolean variables to either 1 or minus 1
    for i in range(aa):
        for j in range(aa):
            if i>j:
              model.Add(constants_x[i][j]== x_vars[i]-x_vars[j]).OnlyEnforceIf(b[i][j])
              model.Add(constants_x[i][j]== x_vars[j]-x_vars[i]).OnlyEnforceIf(b[i][j].N
              model.Add(constants_y[i][j]== y_vars[i]-y_vars[j]).OnlyEnforceIf(b2[i][j])
              model.Add(constants_y[i][j]== y_vars[j]-y_vars[i]).OnlyEnforceIf(b2[i][j].

    #now actually define the objective

    objective = 0
    for i in range(aa):
        for j in range(aa):
            if i>j:
                flow = int(flows.iloc[i,j+1])
                if flow > 0:
             objective = objective  + flow*(constants_x[i][j]+constants_y[i][j])


    model.Minimize(objective)

    # Creates a solver and solves the model.
    solver = cp_model.CpSolver()
    solver.parameters.max_time_in_seconds = 25

    status = solver.Solve(model)

    if status == cp_model.OPTIMAL or status == cp_model.FEASIBLE:
        rtrn = []
        for i in range(aa):
            rtrn.append(solver.Value(widths[i]))
        for i in range(aa):
            rtrn.append(solver.Value(heights[i]))
        for i in range(aa):
            rtrn.append(solver.Value(x_vars[i]))
        for i in range(aa):
```

```
            rtrn.append(solver.Value(y_vars[i]))
        for i in range(aa):
            for j in range(aa):
                rtrn.append(solver.Value(b[i][j]))
        print(solver.ObjectiveValue())
        return rtrn
    else:
        print("RIP")


#reading from the input excel files
path = Path("D:/bouwkunde/Master/thesis/Afterp4/test_problems.xlsx")
xlsx = pd.ExcelFile(path)
data = pd.read_excel(xlsx, "12_req")
flows = pd.read_excel(xlsx, "12_flows")
pre =pd.read_excel(xlsx, "pre")
lc =pd.read_excel(xlsx, "12_matrix_2")
print(flows.iloc[0,2])
print(minwh[0]*s)

FW = FW *s
FH = FH *s
print(FW)


print(minwh[6])
out = SimpleSatProgram(data,lc,flows,FW,FH,minwh,s)

w = []
h = []
x = []
y = []

counter = 0
for i in range(aa*4):
    if counter <aa:
        w.append(out[i])
        counter = counter +1
    elif counter <aa*2:
        h.append(out[i])
        counter = counter +1
    elif counter <aa*3:
        x.append(out[i])
        counter = counter +1
    else:
        y.append(out[i])
        counter = counter +1
```



Appendix K: A vegetable processing factory’s requirements

A vegetable food factory's criteria

A report analyzing and concluding the different requirements and criteria for generating a vegetable food factory

1 JULY

TU Delft
David den Ouden
4469712
Mentors: Pirouz Nourian & Petra Heijnen

Foreword

This report is part of my thesis for building technology and was delivered after the P2 to find out the different criteria any proposed method for solving the factory layout problem should be assessed on. This was done solely by studying a very recent example of a vegetable food processing factory by Hessing Supervers. Most information that is derived in this report will be based on this example factory without providing confidential details.

After the assembly of this document, a meeting will be held with my first and second mentor to assess its contents and discuss the further steps needed. These further steps include the proposal and assessment of different methods and finishing writing the literature research.

Contents

Foreword 3
Introduction 6
Process 7
- Workflow 7
- Layout 8

Departments 10
- Department shapes 10
  - Constraint departments 10
  - Free departments 10
- Number of departments 11
- Area and temperature of departments 11
- Buffer spaces 14
- Location of departments 15

Flows 17
- Kinds of flows 17
- Numbers 18

Transportation 21
- Horizontal transportation 21
  - Forklifts 21
  - Automated guided vehicles 21
  - Machine internal transportation 21
  - Conveyor belts 22
  - Robots on rails 22
- Vertical transportation 22

Conclusion 24
Bibliography 25

Introduction

In order to feed the entire city of London, the field needed would be about 12.1 times the size of the city.[1] This will further increase to 13.3 times as big by the year 2031. In a world where the food consumption and population rises, so does the production of food. In all fields must we as humanity try to improve the way of producing and processing food in order to reduce the impact on the environment.

A missed opportunity is already addressed in the thesis, where computational design and computational power have been an increasing subject of interest the past years. However, in the designing of factories here in the Netherlands, this implementation is far from ideal. It is for this reason that the subject of the thesis is to develop a method that will efficiently generate a layout for a vegetable food processing factory and optimize it to certain objectives.

In this report the criteria a certain proposed method will have to meet are researched and presented. This is done to be able to assess a proposed method based on an actual vegetable food processing factory's data [2]. This will be done by looking at the layout and workflow of the factory, the departments, the flows between the departments and the transportation of these flows. Finally, a conclusion will be presented.

Process

The vegetable food processing factory in question is a future vision of a factory by Hessing Supervers [2]. This factory processes different kinds of vegetables, but also mushrooms and products that include vegetables but where other ingredients are added as well, like meal salads. This means that a lot of different processes are present that are specialized in making a certain end product. Part of this production is standardized, making sure that there is a small buffer of finished products, but part of this production is based on orders from clients, and thus can be hard to predict. A trend may be that during certain times of the month or year, a certain product may be ordered more than another. This creates a situation where in one time period, a process may be utilized more than in another time period.

Over the entire year, the flows between departments can be averaged out so that working with an average flow is still possible to formulate the problem as a static facility layout problem. However, a conclusion must be made that a method that can work with a dynamic facility layout problem must be favored. Another option is to include some sort of adaptability: the ability to rearrange the layout without too much problems, also looking at future changes.

When looking at future changes, the factory in question mean to expand rapidly over the coming years, with the amount of crates that are being handled every day will rise from 300000 to 360000 in the period between 2025 and 2030 [2]. This would mean an increase of almost 20%, certainly putting pressure on the factory's production lines. A way to work around this is to design the factory in such a way that it can be expanded upon easily in the future. To conclude, a method that could take the dynamic facility layout problem into account as well being able to assess future expandability would be favorable over any method that is not able to do this.

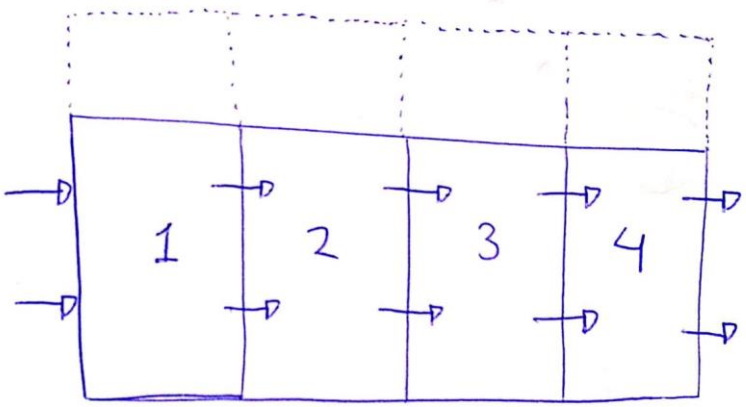


Figure 1. *The asset of expandability, in the figure, all departments have room for expanding. Should one department be blocked in however, this would not be the case. Source: own work.*

Workflow

The workflow in the factory is divided into three parts, where inbound raw materials are being put into crates and stored, then in the second part the raw materials are being processed into semi-finished products and

stored in a buffer and in the last part orders are being picked, the products are being packed and put into crates and send. Now additionally to this basic workflow, there are three types of crates that are being used to transport the products in. The first one are the crates the products arrive in, after they are being emptied, they are washed and send back out again. Then secondly there are the internal crates where the products are being transported in internally, which also need to be checked and washed. Lastly, there are the crates where the final products are being put in to de delivered to all parts of the country, these crates are presumably clean when they arrive, but even so, they need to be checked and cleaned if needed.

This workflow is visualized in the figure below, but it does not add up all the different flows that appear within the factory, it is merely a quick visualization to understand the general purpose flow in the factory. In the section flows, a more detailed analysis will be given of the different flows. Extra flows that are not addressed here but play an important role are for example the flow of people and the flow of garbage.

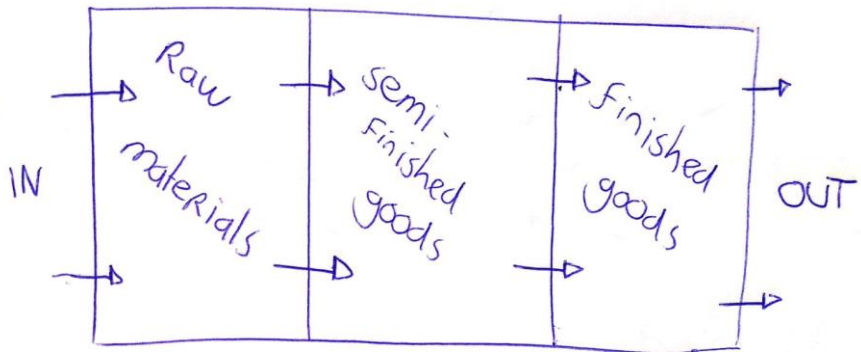


Figure 2. *The simplified flow in the case study factory. Source: own work.*

Layout

The layout of the factory is not very important to determine the different criteria, as the criteria are researched in order to be able to find a method that generates the layout. Hence, assessing the current layout is of little importance for the actual criteria. However, it is still a layout that is currently working, and should therefore be assessed to get a better picture of the whole factory to find constraints or objectives that should otherwise be hard to find. The layout of the factory is given in the two figures below, a plan of the ground floor and a plan of the first floor.

The first thing that catches the eye is the fact that the factory is multi leveled. This implies that multilayering a factory might actually be of an interesting benefit and might be wise to consider when generating a layout. The upper level is almost empty, except for the mixing and buffering functions and a platform crossing from one side of the building, which is used to supervise the various processes that take place on the ground level.

It can be concluded that a multilevel layout may be a good solution, but might not be. Therefore, a method that would be able to assess these multilevel layouts is favored. The nature that drives towards a favor for multiple levels comes from the objective that supervisors should be able to easily supervise all processes and this is easily achieved using a second level. However, contrary to this, a lot of empty space is left over, which might be considered to have a negative impact on the cost objective. Nevertheless, this is an option that could be investigated further.

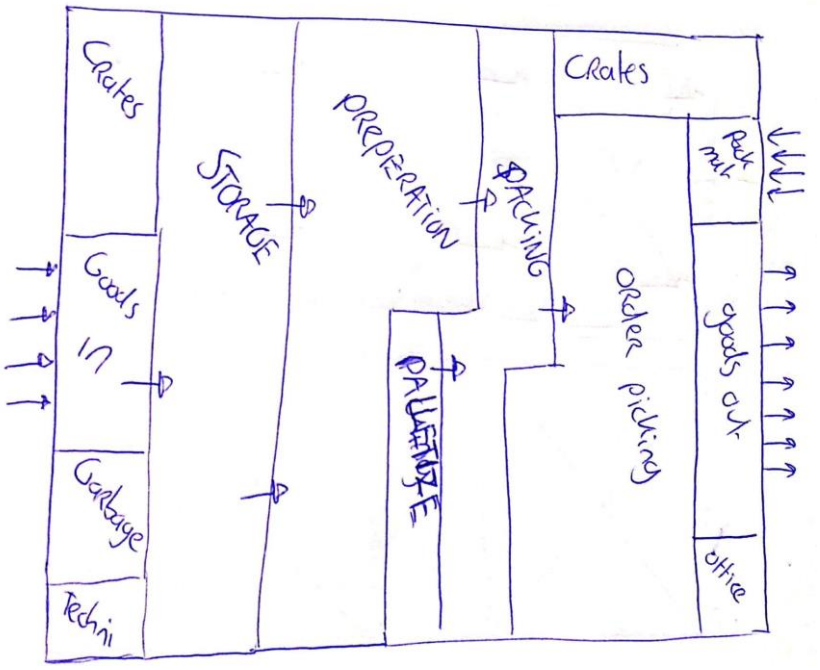


Figure 3. *The simplified layout of the ground floor of the case study. Source: own work based on* [2].

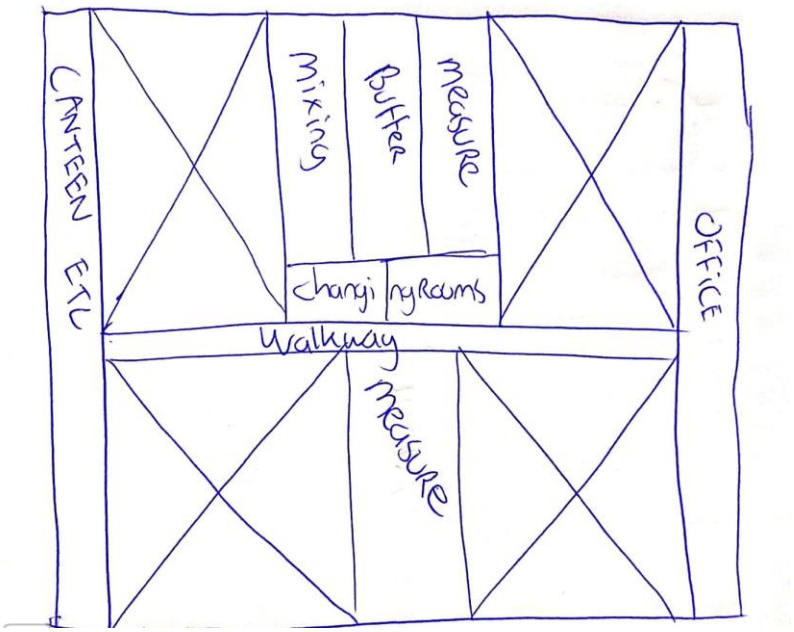


Figure 4. *The simplified layout of the first floor of the case study, notice that the big x's are open to allow supervising. Source: own work based on* [2].

Departments

A typical facility layout problem for a vegetable food processing factory contains a large number of departments, since the different types of vegetables all have different needs. These departments are analyzed in this chapter, concerning their shapes, number, areas, temperature requirements, buffer spaces and the location of departments.

Department shapes

As most methods are either good or not so good in handling certain shape requirements and constraints, it is essential to research whether there will be any such constraints present in this case.

Constraint departments

Constraint departments have as a trait that they are constrained to a certain shape or shape ratio. This is because the department cannot be changed into any other shape without losing its functionality. In this study case, such departments are present and are typically those departments that house a machine or multiple machines. Since these machines are not fluid but come in one shape and one shape only it is not possible to reshape them, just to rearrange machines amongst each other.

While rearranging machines amongst each other might be an option, most of the times the arrangement is pretty straightforward. In the cases where such an arrangement might not be straight forward, a separate study could be made to find the most optimal arrangement, and go with this arrangement in the total study.

In total there are 21 departments that have constraints in their shape, so sitting just under half of the total, table 1 shows the departments that are considered to be constraint in bold. This concludes that any method should definitely be able to model such constraints.

Free departments

Free departments are departments that can be furnished however their shapes may be, with the exception of excessively high width length ratios. These departments might have machines in them that are so small that every arrangement is possible or it might have a function that is not dependent much on the shape.

In the case at hand, over half of the departments are classified as such “shapeless” departments, these departments are shown in regular text in table 1. Examples are the offices or canteen, but also the storage departments, which are packed with racks that can effectively be placed to take any desired shape. Of course these departments have some limitations considering that the width height ratio cannot exceed certain values, but other than that, the departments are easy to work with.

Still there is the problem of buffer spaces, which are spaces that are needed in order to have enough room to operate the machinery or other furniture. These buffer spaces will be discussed further on. To conclude, any method should be able to deal with constraint free and constraint departments and should be able to distinguish between them instead of just labeling the free departments as constraint departments.

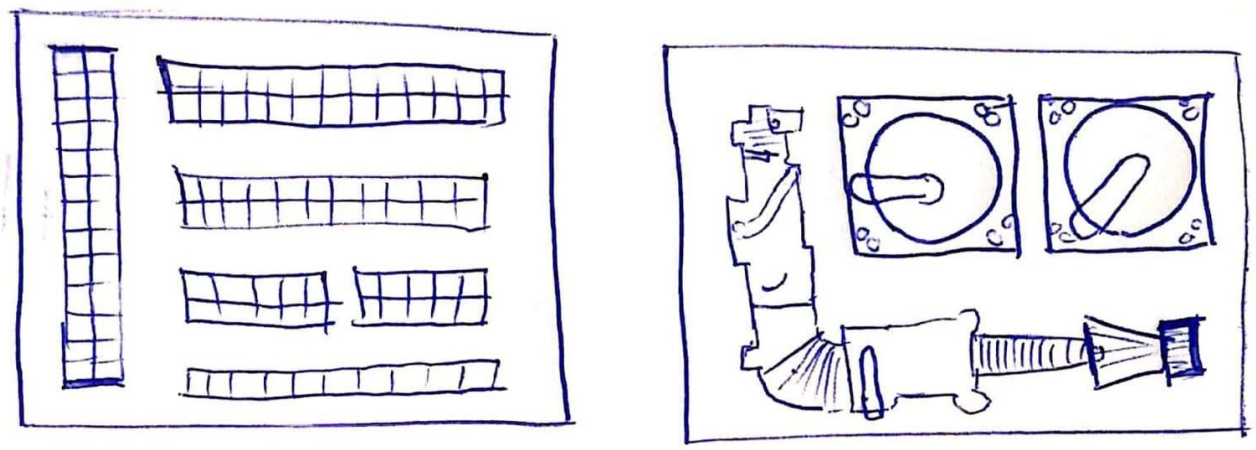


Figure 4. *The difference between a free department on the left, where furniture can practically be placed anywhere and a constraint department on the right, where the machines only have one real appropriate position. Source: own work*.

Number of departments

The number of departments is important because some methods for solving the facility layout problem are more suitable for small size problems, while other methods may be more suitable for medium or large sized problems. For example, in the literature, the maximum number of departments for solving the facility layout problem using an exact method is 15 departments [3].

The case study factory contains in total **53 departments**, of which 42 are situated on ground floor level, 11 on the first floor level and 2 departments partially spawn both. While 53 seems like a very big number, it is reasonably small considering that a lot of different vegetables and products are being processed. Of these 53 departments, 11 are supporting facilities which don't contribute to the processes directly, like offices, supervising rooms and technical rooms. 5 of the departments are dedicated to handling of the crates, 1 department is dedicated to handling garbage, 12 are dedicated to storage and 25 are dedicated to one of the processes.

From this information we can conclude that the chosen method will in fact have to be able to handle at least 50 facilities. Surely an exact method will then not work since almost infinite computational time would be required.

Area and temperature of departments

The areas of the departments are listed in the table below after their respective department name. For each department a minimum and maximum temperature for the departments is also given.

Since the temperature varies from minus 20 to positive 20 degrees centigrade, we can assume that it is likely that departments with different temperatures are going to mix, causing extra costs needed to insulate the walls

to accommodate for the difference (or the cost of energy that is being used to cool the departments if they are not insulated). This concludes that the objective of the problem should not only be to minimize the material handling costs, but also to minimize the cost of placing departments with different temperatures next to each other. This principle is explained in figure 5.

Looking at the areas of the different departments we can be quick to notice that the ranges differ from 75 square meters to 6328 square meters, a substantial difference. The average area is just above 1000 square meters. The conclusion thus is that any method should be able to deal with the range in area of the different departments. Furthermore, the total area is a mere 52 thousand square meters, so a method should be able to handle larger numbers like these, or the problem should be scaled down appropriately.

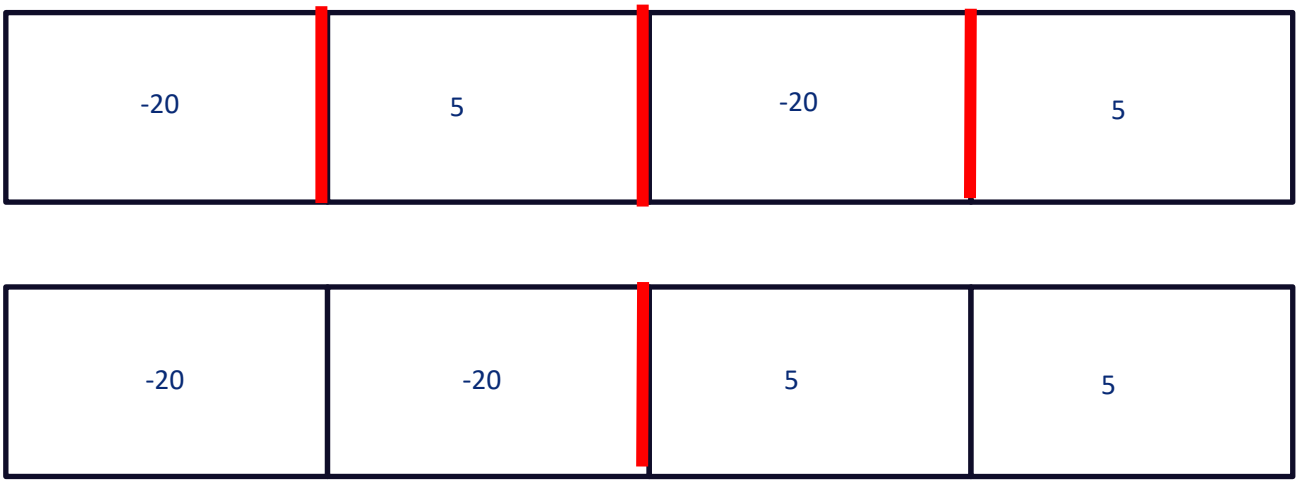


Figure 5. *A configuration of four departments, where in the above configuration, the temperature difference is handled very inefficient. In the bottom variant, only one wall needs to be insulated/has heat loss, whereas the above configuration has three times as much walls. Source: own work.*

	Name	Area	Temperature	
			Min	Max
Process related				
Crate Handling				
	Inbound crate washing	480	5	20
	Inbound crate storage	1056	5	20
	Outbound crate washing/storage	1440	5	20
Storage & Logistics				
	Vegetable storage	2268	2	4
	Specialties storage	648	2	4
	Freezer storage	450	-20	-20
	Prepare outbound raw material crates	230	-	-
	Receiving raw materials	921	5	7
	Storage finished goods	3780	2	4
	(De)palletizing*	1250	2	4

	Order picking	6326	2	4
	Expedition (outbound goods)	2323	2	4
	Receive finished good crates	460	5	20
	Receive packing material	230	-	-
	Buffer storage 1	1426	5	7
	Buffer storage 2	1200	5	7
Garbage				
	Garbage handling	691	5	20
Packing				
	Packing bags continuous 1	735	5	7
	Packing bags continuous 2	735	5	7
	Packing bags not continuous 1	735	5	7
	Packing bags not continuous 2	588	5	7
	Packing bags not continuous 3	147	5	7
	Packing vegetable dish	606	5	7
	Packing lettuce dish	1285	5	7
	Packing meal salad	1977	5	7
Preparing				
	Preparing soft lettuce	2688	5	7
	Preparing hard lettuce	1344	5	7
	Preparing Kale	896	5	7
	Preparing cabbage & leek	870	5	7
	Preparing green beans	422	5	7
	Preparing carrot	422	5	7
	Preparing union	422	5	7
	Preparing paprika	422	5	7
	Preparing cauliflower & Broccoli	355	5	7
	Preparing cucumber pumpkin Champignon	533	5	7
	Manual preparing	533	5	7
Mixing & Measuring				
	Mixing lines (5 in total)	1426	5	7
	Measuring bags (4 in total)	1650	2	4
	Measuring vegetable dishes	266	2	4
	Measuring lettuce dishes	710	2	4
	Measuring meal salads	710	2	4
Non-process related				
	Canteen	1116	-	-
	Installations	900	-	-
	Technical	768	-	-
	General room	497	-	-
	Changing rooms	524	-	-
	Offices	2500	-	-
	Logistics 1	86	-	-
	Logistics 2	86	-	-

	Logistics 3	86	-	-
	Supervising room 1	75	-	-
	Supervising room 2	75	-	-
Total		52369	-	-
Average		-	3,631579	7,289474

Table 1. *A table containing information on the nature of the different departments, their areas and their minimum and maximum temperatures allowed, shape constraint departments are labeled in bold. Source:* [2] *and own work.*

Buffer spaces

As mentioned before, buffer spaces are needed most of the time to operate certain machinery. This is present in either constraint departments and free departments. An example for a constraint department is a department with a large machine, which needs access from all sides for cleaning and maintenance. On the other hand, storage departments also need buffer spaces as racks cannot be stacked side by side indefinitely without walking/access in between them.

This leads to certain inconveniences when taken into account, because every department needs to have appropriate buffer spaces. In the case of departments that are mostly made up of one big machine, the buffer spaces could be shared with neighboring departments, allowing for **overlap** in certain places, even favoring it. In this case, it would be ultimately satisfying if a proposed method could take this into account.

However, considering the complexity of such tasks, it is wise to have an alternative. This alternative comes in the shape of partially neglecting buffer spaces. In the case of the storage areas, it could be assumed that buffer spaces can be efficiently arranged manually in a later stage. In the case of the one machinery departments, it is likely that the neighboring departments will also start with buffer space / empty space and not with a wall or any other kind of solid matter. Therefore, in the cases of the one machine departments, taking into account only half of the buffer spaces on every side should be satisfying. When looking at the study case, these machines are already arranged in such a way.

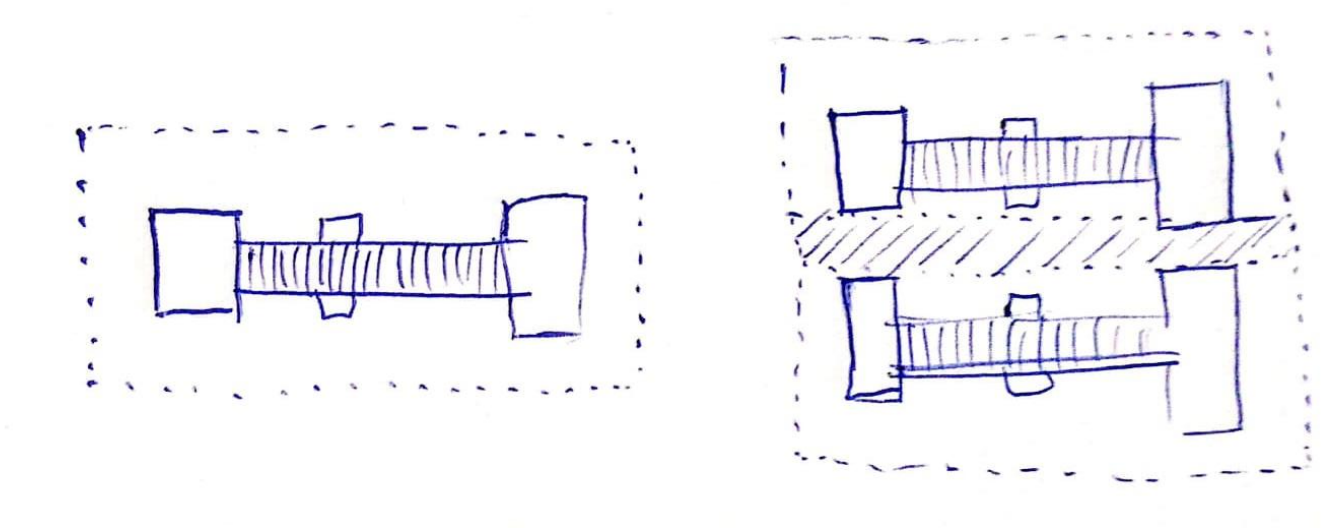


Figure 6. *The buffer spaces of a single machine on the left. On the right is showcased that space can be won when they are in close approximation of each other, since the buffer space for one machine works just as well for the other. Source: Own work.*

Location of departments

Some department have certain requirements for the location that have to be taken into account. These requirements for example are that certain departments need to be next to the outer perimeter like the offices but most importantly: the inbound and outbound goods. Some of these requirements can already be put in the flows between facilities when the outside is also considered a facility. However, in some cases this does not work appropriately, since the offices have no flow to the outside for example.

Other location requirements would be the supervising rooms, canteen, crate washing, garbage and the logistics rooms. The requirements for these rooms are put in the table below. To conclude, these restrictions have to be somehow transformed into a constraint, in order to get valid results.

	Name	Restriction
Location restrictions	Prepare outbound raw material crates	On the outside wall (Flow)
	Receiving raw materials	On the outside wall (Flow)
	Expedition (outbound goods)	On the outside wall (Flow)
	Receive finished good crates	On the outside wall (Flow)
	Receive packing material	On the outside wall (Flow)
	Canteen	On the outside wall (Visual)
	Offices	On the outside wall (Visual)
	Garbage	On the outside wall (Flow)
	Logistics 1	On the outside wall and far from #2 #3
	Logistics 2	On the outside wall and far from #1 #3
	Logistics 3	On the outside wall and far from #1 #2
	Supervising room 1	Attached to important processes (Visual)
	Supervising room 2	Attached to important processes (Visual)

Table 2. *A table containing information on the rooms which have a locational restriction and the nature of that restriction. Source:* [2] *and own work.*

Flows

Now that the departments have been identified and analyzed, the next step is to analyze how these departments interact with each other. This interaction comes in the shape of flows between the departments, which is defined as a stream of people or goods from one department to another. The amount of flow between two departments influences the necessity of the two departments to be close to each other, as the further departments are, the higher the cost is to transport anything between them. In this section the flows will be analyzed by researching the flows in the case study to find matching criteria for a proposed method later on.

Kinds of flows

The most common objective in the literature for the facility layout problem is the minimization of the material handling cost, which is the sum of the all the product of all the material flows between the facilities and the distance between them. The flow of material in indeed an important flow, however, it is not the only flow of importance. What to say for example, of the flow of people moving through the building? A walking laborer is not a very productive one, so this, in term, will also have an effect on the costs of the factory.

In total there are 8 different kinds of flows, which are:

-People
-Raw materials
-Packing materials
-Semi-finished goods
-Finished products
-Outbound crates
-Inbound crates
-Garbage

While some flows may be considered to be practically the same (like the raw materials, semi-finished goods and finished goods) and all flows should essentially all be minimized as they have a certain cost per distance, but the cost and the different needs that are bound to each flow are different, and therefore, should be treated as different.

The people flow is the flow of all people within the building, like machine handlers, janitors, mechanics, and even the CEO. These people mostly flow from the entrance to the offices, canteen and changing rooms and bathrooms. Another important flow is between the supervising rooms and the supervised rooms, or the logistics rooms and the outside. Flows of people will almost always originate or end from or to these rooms:

-Canteen
-Offices
-Supervising rooms
-Logistics rooms
-Offices

Also, almost all rooms require a person to supervise or operate, so all of them have a flow of people originating or ending, though it is relatively small.

The material flow in the building (raw materials, semi-finished products and finished products) follows a more rational path than the flow of people. Raw materials enter the building through the receiving raw material room, where they are quickly distributed to the storage rooms. Keep in mind that the storage room is very big, but all goods are stored in places where they are conveniently close to their respective processing machines, for measuring distance then, it could be argued that taking the distance from the center to the center is not accurate enough, but the simplicity that is gained outvalues the little extra accuracy.
After the raw materials enter the preparing processing machines they become semi-finished products and are transported to the second level to be washed mixed and weighted. Mixing here means that several different ingredients come together, whilst washing and weighing are for single elements (which may be mixed beforehand). Next, different semi-finished products are send back downstairs to be packed, where different products may make use of different packing materials, but some use the same (and therefor use the same kind of packing lines). After packing the products are now finished products and are stored until they are ready for order picking, after which they are send to the outbound room.

The packing material arrives at the packing material arrival room, from there it is sorted and stored (while still in the same room), after which it is send to all the different packing lines when needed, including different kind of packing materials for the different lines and products. The packing material then merges with the semi-finished goods to become finished goods.

Both type of crates serve to move the raw materials and the finished goods, so they accompany them on their journey through the building, additionally, the crates arrive and are returned. However, before they are used, they have to be washed, which is done in the inbound and outbound crate washing rooms. Additionally the finished goods crates are de-stacked to allow the products to be added during packing, and straight after the pallets are stacked in the same room again. This makes the (de)stacking a very **important** room as a lot of flows come together here.

The garbage flows all end in the garbage handling room. These flows all originate from the rooms that have processes (so excluding storage rooms and non-process related rooms). All of these rooms produce garbage, which is collected and send to the garbage handling room to be disposed of.

This concludes that a proposed method should be able to handle all these different kind of flows, whether they be simplified or not, probably hidden within the objective. This also means that the method should be able to identify what kind of flow it is currently referring to and the information containing the different kind of flows should be one of the initial starting points needed for the method to work.

Numbers

The numbers of the different kind of flows between the departments are hard to come by, the exact movement between the different departments are not recorded, not for the materials and let alone for the movement of the people. However, the exact numbers are not super important, as they can be scaled infinitely and the result for the layout will still be the same (considering no fixed costs for building are included). Only the ratio between the flows from department to department are important.

These ratios can be more or less determined when looking at the areas reserved for each activity and the number of in and outbound ports reserved for each activity. From [2] it is given that the packing lines combined

handle 21500 crates every day and that the preparation lines combined handle 38000 crates every day. Furthermore, the mixing lines handle 13500 crates every day. Additionally, there is space for 11500 crates in the semi-finished product buffer.

From these data, combined with the area ratios between different departments, the following flows can already be determined:

	Name	Area	Flow	
Packing		6808	21500	Crates/day
	Packing bags continuous 1	735	2321	Crates/day
	Packing bags continuous 2	735	2321	Crates/day
	Packing bags not continuous 1	735	2321	Crates/day
	Packing bags not continuous 2	588	1857	Crates/day
	Packing bags not continuous 3	147	464	Crates/day
	Packing vegetable dish	606	1914	Crates/day
	Packing lettuce dish	1285	4058	Crates/day
	Packing meal salad	1977	6243	Crates/day
Preparing		8907	38000	Crates/day
	Preparing soft lettuce	2688	11468	Crates/day
	Preparing hard lettuce	1344	5734	Crates/day
	Preparing Kale	896	3823	Crates/day
	Preparing cabbage & leek	870	3712	Crates/day
	Preparing green beans	422	1800	Crates/day
	Preparing carrot	422	1800	Crates/day
	Preparing union	422	1800	Crates/day
	Preparing paprika	422	1800	Crates/day
	Preparing cauliflower & Broccoli	355	1515	Crates/day
	Preparing cucumber pumpkin Champignon	533	2274	Crates/day
	Manual preparing	533	2274	Crates/day
Mixing & Measuring		4762	13500	Crates/day
	Mixing lines (5 in total)	1426	4043	Crates/day
	Measuring bags (4 in total)	1650	4678	Crates/day
	Measuring vegetable dishes	266	754	Crates/day
	Measuring lettuce dishes	710	2013	Crates/day
	Measuring meal salads	710	2013	Crates/day

Table 3*: Flows that could be established by looking at the area ratios between the different departments. Source:* [2] *and own work.*

Other flows that can be determined are the amount of inbound and outbound crates, and the amount of crates that are being washed every day, this is based on the crates that are being processed in preparation and packing every day respectively. The packing material flow and the garbage flow can also be determined, which are based on the ratios between the number of ports for trucks that are reserved for packing/garbage in comparison to the ports reserved for the inbound and outbound goods. For every other flow no data is available, and thus, will have to be made up for now.

	Name	Area	Flow	
Crate Handling				
	Inbound crate washing	480	38000	Crates/day
	Inbound crate storage	1056		Crates/day
	Outbound crate washing/storage	1440	21500	Crates/day
Storage & Logistics				
	Vegetable storage	2268	25604	Crates/day
	Specialties storage	648	7316	Crates/day
	Freezer storage	450	5080	Crates/day
	Prepare outbound raw material crates	230	38000	Crates/day
	Receiving raw materials	921	38000	Crates/day
	Storage finished goods	3780	2	4
	(De)palletizing*	1250	2	4
	Order picking	6326	21500	Crates/day
	Expedition (outbound goods)	2323	21500	Crates/day
	Receive finished good crates	460	21500	Crates/day
	Receive packing material	230	10750	Crates/day
	Buffer storage 1	1426		
	Buffer storage 2	1200		
Garbage				
	Garbage	691	19000	Crates/day

Table 4. *Additional flows that could be established when looking at the inbound and outbound crates and at the ratio between the ports for trucks. Source:* [2] *and own work.*

Transportation

The main goal of this section is to try to determine the ratio between the cost for transport, as this, together with the distance between the departments and the flow, will eventually determine the fitness of the objective. For this reason, different modes of transportation will be explicated. This explication is divided in vertical and horizontal transportation. Another goal is to determine the means of how the distance between two facilities is calculated.

Horizontal transportation

Horizontal transportation is the most common form of transportation trough the building. This kind of transportation can not solely be attribute to one form of transportation, but in many cases, a factory uses more than one kind of transportation. In this case study, it is no different, for example: Hessing states that it is in fact against forklift trucks inside the production perimeter . This is probably to prevent crowded situations in the production process. Inside the storage perimeters for example, the forklift is the main use of transportation. This already goes to show that no one means of transportation is dominating, and **thus** a proposed method should be able to keep in mind the different kinds of transportation.

Forklifts

Forklifts are mainly used to transport pallets stacked full of crates within the storage areas, as they are not permitted inside production perimeter. They carry the stacks of pallets from the inbound raw material rooms to the storage rooms, where they are stored until needed, after that, they carry the goods (from the same place they ended up being stored) to the corresponding production machine. The distance travelled between the drop-off point and the storage, can be anywhere, however, for the next distance, which is from storage room to machine, the starting point needs to be the ending point of the previous movement. In theory all materials should be stored in a different place, however, for simplicity reasons, the distance shall be calculated as the Euclidian distance between the two centroids of departments.

Forklifts are generally quite expensive to run, but cheap to buy initially, as the machines are cheaper than for example conveyor belts and automated guided vehicles. However, they are run manually and in the long run, this will cost more than the robotic variant.

Automated guided vehicles

Automated guided vehicles are essentially the same as forklifts but they are not manual but automatic, as the name implies. This means that they can easily communicate with each other and so create a more organized transportation, so it is more ideal to use within the production perimeter. Automated guided vehicles charge no money to run except for the electricity, but the initial cost weigh much higher when compared to a traditional forklift. Automated guided vehicles can be used in the production perimeter to transfer crates, alternatively, conveyor belts can be used.

Machine internal transportation

Machine internal transportation is the transportation of goods through the machine, which is done by the machine itself. This transportation is always going to be the same, no matter how the machine is placed, this

means that the distance and the cost of these transportations have to influence on the objective, as they are set. However, what is very important with these machines, is that they have drop on and drop-off points, which are the only places where goods can be delivered or taken from. This is a very interesting constraint which should be taken into account. All machines that are listed as constrained machines in table X (bolt) are also the machines that have drop-on and drop-off points.

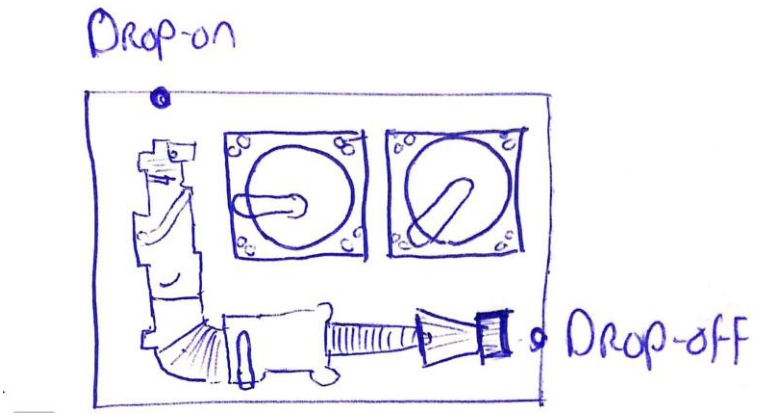


Figure 8. *The concept of drop-on and drop-off points for machines. Source: own work.*

Conveyor belts

Conveyor belts are another means of automated transport. However, a big difference with AGV's is that the conveyor belt is a continuous transportation of material. Examples of where the conveyor belt is used is to deliver empty crates to the packing machines (although they come directly from the depalletize machine, and could therefor be counted as machine internal transportation). Another example is the possibility to move goods vertically or under an angle, but this will be covered later. Conveyor belts mostly use the Euclidean distance when connecting two points. Conveyor belts are also used in the case study to move pallets from and to the buffer zone.

Robots on rails

In the order picking department, the orders in the case study are picked using artificial intelligent machines on "rollies" e.g.: guided rails. These robots take the orders based on what the clients want and batch them together on the expedition. The robots are very efficient and relatively cheap to operate but a are a major investment. As for the distance calculation of the packing machines to the order picking department to the expedition, it can again best be simplified to the Euclidian distance. Another possible way of measuring the distance is using the Manhattan or rectilinear distance, since most movement in factories are perpendicular.

Vertical transportation

Vertical transportation is needed when the facility contains more than one level. This transportation can be done using different means. The first one is a elevator, which works in badges, this has an advantage when goods don't have to be transported continuously but in badges, as there is usually more space available. Another means is to have a conveyor belt run under an angle from one floor to the other, which is a continuous process. This means of transportation has a preference when there is a steady supply of goods that need to be transported. A disadvantage is that it takes up more space than an elevator.

Calculating distance in vertical direction is not needed, as the height of the levels are constant and thus do not affect the fitness value for one solution compared to others. It should however be added when comparing single level layouts to multi-level layouts. Additionally, when computing the horizontal distance between two machines at different levels, the goods will first have to travel from machine A to the escalator/belt and only then to machine B (where both movements can be straight lines). Ideally, machines A and B would be placed closed to the vertical transportation means.

Conclusion

Following the conducted research, the following conclusion can be drawn when looking at criteria a method would have to meet when solving this particular facility layout problem. Per section these criteria are as follows:

From the process section the following conclusions on criteria can be drawn:

- a method that can work with a dynamic facility layout problem is favored, however not necessary, as flows can be averaged out.
- A method that could assess future expandability would be favored.
- A multilevel layout may be an objective, but might not be. Therefore, a method that would be able to assess these multilevel layouts is favored.
- Vision between two departments should count as a quality and might be interesting to implement in a method (especially with multi-floors).
- Empty space should be taken into account and punished.

From the departments section the following conclusions on criteria can be drawn:

- There are both free shaped departments (one size/different shapes) and constraint departments (one shape/size), both should be modelled.
- The free departments have a height/width ratio constraint.
- The number of departments is at least 50, a method should be able to handle that.
- Temperature loss should be taken into account and punished.
- The areas of the department range from 75 square meters to 6000 square meters, a smallest unit should therefor be 75 square meters maximum.
- A method that takes buffer spaces into account is favored, however, since this is difficult, a simplification is allowed.
- Some departments have location restrictions that have to be put into a constraint.

From the flows section the following conclusions on criteria can be drawn:

- A method should be able to handle all these different kind of flows, whether they be simplified or not, probably hidden within the objective.
- A method should be able to distinguish different kinds of flows.

From the transportation section the following conclusions on criteria can be drawn:

- Different kinds of transportation are excluded between two sets of facilities and they have different costs attached to them, a method should be able to work with this.
- Distance is measured differently depending on the types of departments, so a method should be able identify and apply different ways to calculate distance.
- Some machines have drop-on and drop-off points (most of the constraint machines)

When combining these criteria with a weight (based on the necessity of the criteria) different methods can be benchmarked and compared. The weights are needed to distinguish the indispensable criteria from the optional ones.

Bibliography

[1] Evening Standard, “How will we feed and water 10 million Londoners by 2031?,” 2017. [Online]. Available: https://www.standard.co.uk/news/london/how-will-we-feed-and-water-10-million-londoners-by-2031-a3629106.html. [Accessed: 23-Jan-2020].

[2] Hessing, “Green future, logistieke keuzes 31 oktober 2019.” Hessing, p. 33, 2019.

[3] A. Drira, H. Pierreval, and S. Hajri-Gabouj, “Facility layout problems: A survey,” *Annu. Rev. Control*, vol. 31, no. 2, pp. 255–267, 2007.