Quality of Service Routing in the Internet

Theory, Complexity and Algorithms

Quality of Service Routing in the Internet

Theory, Complexity and Algorithms

Proefschrift

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus Prof.dr.ir. J.T. Fokkema,
voorzitter van het College voor Promoties,
in het openbaar te verdedigen op dinsdag 14 september 2004 om 15.30 uur

door

Fernando Antonio KUIPERS

elektrotechnisch ingenieur
geboren te 's Gravenhage.

Dit proefschrift is goedgekeurd door de promotor:
Prof.dr.ir. P.F.A. Van Mieghem

Samenstelling promotiecommissie:

Rector Magnificus,	Voorzitter
Prof.dr.ir. P.F.A. Van Mieghem,	Technische Universiteit Delft, promotor
Prof.dr.ir. I.G.M.M. Niemegeers,	Technische Universiteit Delft
Prof.dr.ir. N.H.G. Baken,	Technische Universiteit Delft
Prof.dr.ir. C. Roos,	Technische Universiteit Delft
Prof.dr. J. Domingo-Pascual,	Universitat Politècnica de Catalunya
Prof. Ing. G. Ventre,	Università di Napoli Federico II
Dr.ir. H. De Neve,	Alcatel Belgium

Published and distributed by: DUP Science

DUP Science is an imprint of
Delft University Press
P.O. Box 98
2600 MG Delft
The Netherlands
Telephone: +31 15 27 85 678
Telefax: +31 15 27 85 706
E-mail: info@library.tudelft.nl

ISBN 90-407-2523-3

Keywords: QoS routing, algorithm, complexity

Printed in The Netherlands

to Theo, Maria and Carolina

Contents

Summary

Title: Quality of Service Routing in the Internet: Theory, Complexity and Algorithms

An enormous amount of packets daily traverse the Internet towards their intended destination. The Internet consists of many network elements that direct these packets on the correct path leading towards the destination. This process of finding and following a path to the destination is called routing. Of course, routing is not infallible and packets may get lost: the current Internet cannot give any guarantees regarding the packets it transports, i.e. there are no guarantees on the delay that packets experience, on the jitter, or the packet loss, nor can it guarantee the bandwidth available along the travelled path. However, many new multi-media applications cannot properly operate without such guarantees, e.g. for a voice conversation, the maximum delay must be bounded. Finding paths that can meet such demands is called Quality of Service (QoS) routing.

The aims of this thesis are to:

1. analyze the algorithmic concepts of QoS routing
2. investigate the complexity of QoS routing
3. discuss the dynamics of QoS routing

The first three chapters formalize the problems under consideration, define the notation used and provide the necessary background material, including the following definitions (Chapter 2):

Algorithm: *An algorithm is any well-defined computational procedure that takes some value, or set of values, as input and produces some value, or set of values, as output. An algorithm is thus a sequence of computational steps that transform the input into the output.*

Complexity: *Complexity refers to the intrinsic minimum amount of resources needed to solve a problem or execute an algorithm.*

QoS routing is NP-complete, which means that to find the exact solution, algorithms require, in the worst case, a running time that cannot be bounded by a polynomial function. The last section of Chapter 2 discusses the theory of NP-complexity.

To understand QoS algorithms we also need to be familiar with simple (one-dimensional) shortest path algorithms. Therefore, Chapter 3 explains the breadth-first search, the depth-first search, the Bellman-Ford algorithm, the Dijkstra algorithm, bi-directional search, the A* algorithm, and mathematical programming. An important property that these algorithms share is that *subpaths of shortest paths in one dimension are also shortest paths.*

After clarifying the background material we reach the heart of the matter in Chapter 4, namely the concepts underlying exact QoS routing. As a result of QoS routing with multiple constraints, *subsections of shortest paths in multiple dimensions are not necessarily shortest paths themselves.* In the computation of multi-constrained paths, it may for this reason be necessary to consider multiple subpaths. This has consequences for the size of the search space, which may grow exponentially. To reduce the size of the search space, two techniques, non-dominance and look-ahead, are used and incorporated into the SAMCRA algorithm. SAMCRA stands for Self-Adaptive Multiple Constraints Routing Algorithm and is an exact QoS algorithm proposed by us. Besides SAMCRA, many other QoS algorithms exist. By far the largest part of these QoS algorithms are heuristics. Chapter 5 discusses these QoS algorithms and evaluates their performance. This large-scale performance evaluation has never been conducted before. The conclusions indicate that the SAMCRA-like algorithms perform best.

Chapters 6 and 7 may be considered elaborations, since they look at extensions to QoS routing. First, multicast QoS routing is discussed. Multicast refers to the communication between one source and multiple destinations. In multicast routing packets are duplicated at appropriate points, which leads to an efficiency gain over multiple unicast (single source-destination pair) sessions. Multicast QoS routing also relies on this principle, but the efficiency gain can be less than in the one-dimensional case. We propose the MAMCRA algorithm, which is the first general algorithm for multicast QoS routing.

In Chapter 7, link-disjoint QoS routing is targeted. Link-disjoint routing consists of finding two paths that do not share any links. These two paths are important if reliability is desired: one path can be used as the primary path and if this path fails, one can immediately switch to the second back-up path. Link-disjoint paths could also be used for load balancing. Similarly to Chapter 6 (multicast QoS routing), we discuss the problems surrounding link-disjoint QoS routing and propose the algorithm DIMCRA, which is the first general algorithm for link-disjoint QoS routing.

Chapters 4-7 extensively and uniquely contribute to the first aim of this thesis: to analyze the algorithmic concepts behind QoS routing. The second goal of investigating the complexity of QoS routing, is attained by Chapter 8. Chapter 8 argues that the complexity of QoS routing is feasible in practice and that worst cases are only encountered if the network simultaneously obeys four conditions on: (1) the underlying topology, (2) the size of the link weights, (3) the (negative) correlation among the link weights and (4) the values of the constraints.

The third and final aim of the thesis is to discuss the dynamics of QoS routing. Chapter 9 is devoted to this discussion and also provides some preliminary work in the area of QoS dynamics. The key research questions are clearly identified and basically reduce to the question of how to keep the network up to date on the current state of the QoS link weights. The work and simulations presented give some ideas about the stability of QoS paths and the performance of SAMCRA in a dynamic network: here, too, SAMCRA outperforms the other implemented algorithms. However, the conclusions presented Chapter 9 should merely be interpreted as guidelines, as more simulations should be conducted to substantiate them.

Author: Fernando A. Kuipers

Chapter 1

Introduction

"Quality of Service Routing in the Internet:" such a title deserves some scrutiny, which is provided in this Introduction.

1.1 Routing in the Internet

The Internet is a collection of networks interconnected to each other. As a public commodity, the Internet is considered to be a great success: almost everybody knows and uses the Internet. This success is contributed to several factors: the Internet is simple, affordable, fair (everybody receives about the same treatment), and it provides a sense of freedom (nobody owns the Internet). These factors were in fact part of the architectural design principles for the Internet, whose origin can be traced back to a 1969 project of the US Department of Defense. The project turned out to be a success and the Internet started to grow (first including government and research organizations and later also private companies). Nowadays the Internet connects millions of users to each other. These users have computers of varying capabilities, of various vendors, running different operating systems. Users can communicate with each other over the Internet thanks to the TCP/IP protocol suite. The TCP/IP protocol suite is the combination of different protocols at various layers [153]:

- The link layer handles all the hardware details.
- The network layer handles the movement of packets (i.e., routing) around the network.
- The transport layer provides a flow of data between two hosts for the application layer above.
- The application layer handles the details of the particular application.

The title of this thesis, "Quality of Service Routing in the Internet," therefore relates to the network layer. Routing in general involves two entities, namely the routing protocol and the routing algorithm. The routing protocol has the task of acquiring information about the current state of the network and to distribute this information to all the nodes (routers) in the network. Based on the view of the network that the routing protocol has provided, the routing algorithm is used to compute the paths that packets must follow in order to reach their destination. The routing algorithm therefore heavily depends on the accuracy of the routing protocol. If the routing protocol is not able to provide each node with a consistent view of the network, then routing loops may occur (i.e., packets do not reach their destination) and proper communication is hindered.

In the current Internet (since the beginning), routing focusses on connectivity and is referred to as best-effort routing. It is only important to know whether links or nodes are connected to the network. This kind of network state information is (quasi) static because links/nodes go down only sporadically. Routing in the Internet is decomposed into two levels: (1) intra-domain routing, where routing is performed within a network, and (2) inter-domain routing, for routing between networks. The current dominant intra-domain routing protocol is called Open Shortest Path First (OSPF). OSPF is called a link state protocol, because it monitors links (and consequently also nodes), and if a change in link state has occurred, it floods this information through the entire network. It only does so periodically, typically every 15 minutes, because the link state is (quasi) static. Based on this information, the Dijkstra algorithm is used at each node to compute the shortest paths (usually based on the hop count) to all other nodes in the network.

1.2 Quality of service

Quality of service, abbreviated as QoS, has many definitions. For example, according to the QoS Forum: "*Quality of Service is the ability of a network element to have some level of assurance that its traffic and service requirements can be satisfied.*" QoS can be considered to be subjective, because users can differ in their perception of what is good quality and what is not. However, the level of assurance at which traffic and service requirements are satisfied can be quantized. The best-effort paradigm does not provide any assurance on the traffic handled and therefore we classify best-effort routing as a paradigm that does not offer QoS. Still, best-effort routing seems to function properly, which questions the need for new QoS mechanisms. We will argue why QoS is needed for the future. In the business world, QoS could determine whether you can have a normal voice conversation, whether a video conference is productive, and whether a multimedia application actually improves productivity for your staff. At home, it could for instance determine whether you will have cause to complain about the quality

of a video-on-demand movie. Overall, we see that new applications are increasingly demanding higher quality than the one-size-fits-all best-effort service offered by the initial Internet design.

We can think of many other situations in which QoS is needed. For instance, if we look at the distinction between one-way and two-way communication, we notice that one-way communication can accept relatively long delays. However, delay hinders two-way, interactive communication if the round-trip time exceeds 300 ms. For example, conducting a voice conversation over a satellite link illustrates the problem with long delays. Combined video and audio is very sensitive to differential delays. We for instance quickly notice when the speech is out of sync with lip movement. Data communication protocols are very sensitive to errors and loss. An undetected error can have severe consequences if it is part of a downloaded program. Loss of a packet frequently requires retransmission, which decreases throughput and increases response time. On the other hand, many data communication protocols are less sensitive to delay variation. These are all examples that illustrate that it is important to assure that traffic and service requirements like delay, jitter, loss and throughput can be satisfied.

An other reason for introducing QoS is that it could enhance the performance of operational networks. For instance, QoS mechanisms could lead to a better balancing of the load in a network and consequently a more efficient use of the network's resources. This efficiency gain will result in an increase of revenues for the network providers, which leads us to another important argument in favor of QoS, namely "money." Some people argue that bandwidth[1] is the answer to the question how to obtain QoS. Bandwidth is indeed a key component for offering QoS, because without a proficient infrastructure many services cannot be delivered. Such a QoS-infrastructure will most likely consist of optical fibers that extend all the way to the end-users (FttH). However, bandwidth alone is not the answer, since it cannot optimize network performance and the only way to improve the network is therefore via over-dimensioning. Over-dimensioning is very costly, resource inefficient and it still cannot guarantee QoS. By offering different levels of QoS one can differentiate between users and hence provide tailor-made service/pricing. This extra flexibility over best-effort Internetting opens new business opportunities. Unfortunately solid business cases accompanied with good billing and accounting models are still missing, which may explain the fact that QoS is still a scarce commodity in the Internet. Another factor hindering the global breakthrough of QoS routing is the increased complexity compared to simple best-effort routing. The complexity of QoS routing is investigated in this thesis. Despite these difficulties surrounding QoS routing, its merit is globally recognized and much research has al-

[1] The formal definition of bandwidth stems from the field of electrical engineering, where it represents the difference between the highest and lowest frequencies (Hz) of a transmission channel/band. In the field of computer networking, the term bandwidth is often used to denote the data rate or capacity, i.e. the amount of data (bits) that is or can be sent through a network connection per second. For the sake of convention, we maintain the definition for bandwidth in the field of computer networking.

ready been done on the subject. The pioneering work on QoS started with ATM. The ATM Forum's PNNI standard defines a routing protocol for distributing topology and load information throughout the network and a signaling protocol for processing and forwarding connection requests from the source. ATM is a connection-oriented technology. ATM allows a user to specify, when setting up a call, QoS constraints that an ATM network must be able to guarantee for that call. Call establishment consists of two operations: (1) the selection of a path based on multiple constraints, and (2) the setup of the connection state at each point along that path. Path selection is done in such a way that the path chosen appears to be capable of supporting the QoS constraints requested, based on currently available information. The processing of the call setup at each node along the path confirms that the resources requested are in fact available. If they are not, then crankback occurs, which causes a new path to be computed if possible. Thus the final outcome is either the establishment of a path satisfying the constraints, or refusal of the call. The concepts of ATM aided in introducing QoS in IP. One of the first QoS architectures for IP was the Integrated Services (IntServ) architecture [22]. IntServ distinguishes between three categories of services: Guaranteed Service [149], Controlled Load [175] and best-effort. An application can request a reservation for a flow (typically via the Resource reSerVation Protocol (RSVP) [180]) for a guaranteed or controlled load QoS, with a traffic specification (TSpec) that defines the exact amount of service required. Guaranteed Service in IntServ can provide firm (mathematically provable) upper bounds on the queueing delay through the network, which allows it to make guarantees on bandwidth, delay and queueing losses (there are none). In order to accomplish this, packet classifiers, packet schedulers, and admission control are used. The controlled load service in IntServ cannot give specific upper bounds on the queueing delay. Nevertheless, the service ensures that a very high percentage of the packets do not experience excessive delays. The controlled load service provides the flow of packets with QoS closely approximating the QoS that the same flow would receive from best-effort service under unloaded network conditions. This is achieved through admission control. The main drawback of IntServ (and ATM) is that they require per-flow state and per-flow processing, which is not scalable in large networks (such as the Internet). To cope with these scalability problems, aggregation of flows is needed, which led to the proposal of the Differentiated Services (DiffServ) architecture [20]. The principle of DiffServ is simply to classify packets into several classes, which are treated differently according to different packet scheduling and policing rules, or more poetically "*all packets are equal, but some packets are more equal than others.*" Compared to IntServ, DiffServ improves scalability at the cost of less predictable service to flows.

In the context of IP QoS architectures we also want to mention MultiProtocol Label Switching (MPLS) [140]. MPLS uses labels to expedite forwarding compared to conventional IP routing. A label distribution protocol is used to inform the MPLS-capable routers how to forward packets with a specific label. Since the labels are shorter than

IP addresses, the packets can be forwarded at a faster rate. The use of labels also creates other advantages, like the support of explicit routing. This gives network/service providers a great deal of flexibility to divert and route traffic around link failures, congestion and bottlenecks, and to provide QoS routing. Nowadays, MPLS is often used to build virtual private networks that can span different Internet domains.

In addition to IntServ, DiffServ and MPLS other QoS architectures were proposed, such as combinations of the aforementioned architectures or the Nimrod architecture [25]. However, to fully utilize the potential of these QoS architectures, the way of path selection should also be QoS-aware. For example, in the context of ATM (PNNI), QoS routing is performed by source nodes to determine suitable paths for connection requests. These connection requests specify QoS constraints that the path must obey. Since ATM is a connection-oriented technology, a path selected by PNNI will remain in use for a potentially long period of time. It is therefore important to choose a path with care. The IntServ/RSVP framework is also able to guarantee some specific QoS constraints. However, this framework relies on the underlying IP routing table to reserve its resources. As long as this routing table is not QoS-aware, paths may be assigned that cannot guarantee the constraints, which will result in blocking. In MPLS a source node selects a path, possibly subject to QoS constraints, and uses a signaling protocol (e.g., RSVP or CR-LDP) to reserve resources along that path. In the case of DiffServ, QoS-based routes can be requested, for example, by network administrators for traffic engineering purposes. Such routes can be used to ensure a certain service level agreement [176]. Even the high-capacity optical networks (SONET/SDH/WDM) require the use of constraint-based path selection algorithms to cope with the various transmission impairments (e.g., attenuation, crosstalk, dispersion, non-linearities) along the optical path. Different paths are likely to show different performance in terms of transmission quality. If electronic regeneration is used in optical networks, the various transmission impairments can be combatted, but different sets of limitations are imposed (e.g., additional delay, reduced reliability and increased operational cost). Therefore, to ensure QoS, multi-constrained routing algorithms are needed. These examples all indicate the importance of constraint-based routing algorithms, both in ATM and IP.

To enable QoS routing, it is necessary to implement state-dependent, QoS-aware networking protocols. Examples of such protocols are PNNI [158] of the ATM Forum and the QoS-enhanced OSPF protocol [7]. For the first task in routing (i.e., the representation and dissemination of network-state information), both OSPF and PNNI use link state routing, in which every node tries to acquire a "map" of the underlying network topology and its available resources via flooding. Despite its simplicity and reliability, flooding involves unnecessary communications and causes inefficient use of resources, particularly in the context of QoS routing that requires frequent distribution of multiple, dynamic parameters, e.g., using triggered updates [6]. Designing efficient QoS routing protocols is still an open issue. The focus of this thesis is on QoS algorithms and their complexity. For this study we assume that the network-state information is

temporarily static and has been distributed throughout the network and is accurately maintained at each node using QoS link state routing protocols. Once a node possesses the network-state information, it performs the second task in QoS routing, namely computing paths based on multiple QoS constraints. Before giving the formal definition of the *multi-constrained path* problem, first the notation is established.

1.3 Notation

A network is represented as a graph $G = (V, E)$ consisting of a set V of $N = |V|$ nodes and a set E of $M = |E|$ links. Nodes (in the literature also referred to as vertices or points) represent the routers or switches in a network, while the links (also referred to as edges, arcs or lines) represent the communication links (e.g., optical fiber, wireless channel, ...). We only consider connected graphs without self-loops and at most one link between a pair of nodes. A specific link in the set E between nodes u and v is denoted by (u, v). Each link $(u, v) \in E$ from node u to node v is characterized by an m-dimensional link weight vector $\vec{w}(u, v) = [w_1(u, v), w_2(u, v), \cdots, w_m(u, v)]$, where $w_i(u, v) > 0 \ \forall (u, v) \in E$ and the m components refer to QoS measures such as delay, jitter, loss, available bandwidth, cost, etc. A path in G is denoted by P or more specifically $P_{s \to t}$ if the path goes from a source node s to a destination node t. A QoS routing algorithm has the task to compute the path P that obeys multiple QoS constraints. The values L_i are the user requested quality of service desires and $\vec{L}$ is called the constraints vector. The QoS measures belong to two different classes: (1) additive[2] and (2) min-max QoS measures. For additive QoS measures, the value (further called the weight) of the QoS measure along a path is the sum of the QoS weights on the links defining that path. Examples of additive QoS measures are the delay, the hop count and the cost. For min-max QoS measures, the path weight of the QoS measure is the minimum (or maximum) of the QoS weights of the links that constitute that path. Typical examples of min-max measures are the minimum needed bandwidth and

[2] For multiplicative measures, the value of the QoS measure along a path is the product of the QoS values of the constituent links of the path. By taking the (sometimes negative sign of the) logarithm of the multiplicative measures on each link, they are transformed into *positive,* additive measures. An important example is the packet loss, or more precisely 1 minus the probability of packet loss. Indeed, if at a node the average incoming traffic [number of packets/s] is λ and if p denotes the probability of packet loss, then the average outgoing traffic equals $(1-p)\lambda$. The next hop assuring a packet loss q has incoming traffic $(1-p)\lambda$ and outgoing $(1-p)(1-q)\lambda$. Implicitly independence has been assumed. Hence, along a path with h hops the end-to-end probability of packet loss is $1 - \prod_{k=1}^{h} (1 - p_k)$. The end-to-end packet arrival probability $\prod_{k=1}^{h} (1 - p_k)$ is maximized by minimizing $-\sum_{k=1}^{h} \log(1 - p_k)$, where $-\log(1 - p_k)$ are positive, additive measures. This explains why only two different classes need to be considered.

(policy related) transit flags. Routing with (link) constraints on min-max QoS measures consists of omitting all links (and possibly disconnected nodes) from the topology that do not satisfy one of the constraints. We call this topology filtering. In contrast, (path) constraints on additive QoS measures cause more difficulties. Hence, without loss of generality and if not stated otherwise, all QoS measures are assumed to be additive.

1.4 Problem statement

For additive QoS measures the weight of a path $P = n_1 \rightarrow n_2 \rightarrow \cdots \rightarrow n_{h+1}$ consisting of h hops (links) equals the vector-sum of the weights of its constituent links

$$\vec{w}(P) = \sum_{j=1}^{h} \vec{w}(n_j, n_{j+1}) \tag{1.1}$$

The problem of finding a path that satisfies multiple QoS constraints is known as the multi-constrained path problem and is formally defined as follows:

Definition 1 Multi-Constrained Path (MCP) problem: *Consider a network $G = (V, E)$. Each link $(u, v) \in E$ is specified by a link weight vector with as components m additive QoS link weights $w_i(u, v) \geq 0$ for all $1 \leq i \leq m$. Given m constraints L_i, where $1 \leq i \leq m$, the problem is to find a path P from a source node s to a destination node t such that*

$$w_i(P) \stackrel{def}{=} \sum_{(u,v)\in P} w_i(u, v) \leq L_i \tag{1.2}$$

for all $1 \leq i \leq m$.

A path that satisfies all m constraints is referred to as a feasible path. There may be many different paths in the graph G that satisfy the constraints. According to definition 1, any of these paths is a solution to the MCP problem. However, it might be desirable to retrieve the path with smallest length $l(P)$ from the set of feasible paths. The precise definition of length $l(.)$ is important and will be discussed in Section 4.1. The problem that additionally optimizes some length function $l(.)$ is called the *multi-constrained optimal path* problem and is formally defined as follows,

Definition 2 Multi-Constrained Optimal Path (MCOP) problem: *Consider a network $G = (V, E)$. Each link $(u, v) \in E$ is specified by a link weight vector with as components m additive QoS link weights $w_i(u, v) \geq 0$ for all $1 \leq i \leq m$. Given m constraints L_i, where $1 \leq i \leq m$, the problem is to find a path P from a source node s to a destination node t satisfying (1.2) and, in addition, minimizing some length criterion such that $l(P) \leq l(P')$, for all paths P', P between s and t that satisfy (1.2).*

Both the MCP and MCOP problems are instances of QoS routing. The MCOP problem is considered to be more difficult than the MCP problem, because a solution to the MCOP problem is also a solution to the MCP problem, but not necessarily vice versa.

One of the most investigated problems in the context of QoS routing is the Restricted Shortest Path (RSP) problem. The RSP problem is a subproblem of MCOP, in which the goal is to find a path with minimal cost (i.e., length) that obeys one constraint (typically) on the delay.

The main goal of this thesis is to find an exact algorithm for the MC(O)P problem and to evaluate its complexity. The MC(O)P problem is generally considered to be a hard problem for which heuristics should be proposed. The view we uphold is different from the mainstream and may therefore be found controversial. However, our complexity study will strengthen our claim that exact QoS routing is possible in practice.

1.5 Outline

The outline of this thesis is schematically depicted in Figure 1.1. The main body of

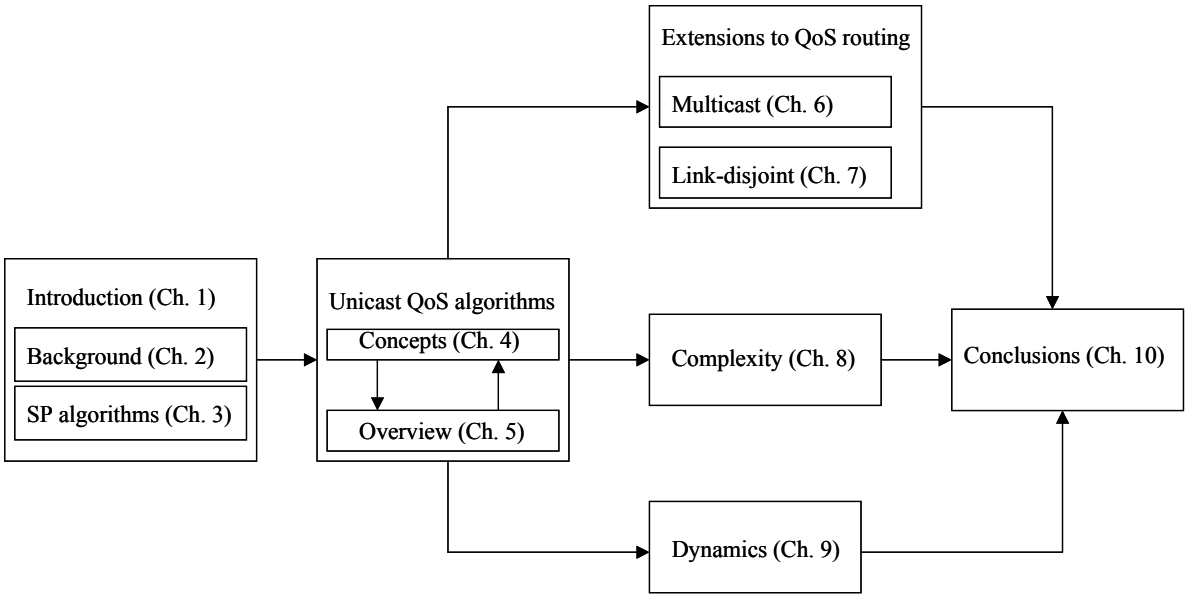


Figure 1.1: Schematic overview of the thesis outline.

the thesis consists of 10 chapters divided over 6 pillars: Introduction, unicast QoS routing algorithm(s), extensions to QoS routing, the complexity of QoS routing, dynamic QoS routing, and conclusions. The direct (horizontal) path from Introduction to Conclusions signifies the main focus of this thesis, while the side-steps to QoS extensions

and dynamic QoS routing should be considered as (important) extensions to the main theory.

A more detailed description of the content of each chapter follows. In addition to this Introductory chapter, Chapters 2 and 3 are also classified under Introduction. Chapter 2 presents the minimal background knowledge that is required to fully understand this thesis. It covers graph theory, the definition of an algorithm, and complexity theory. To understand the "more sophisticated" QoS algorithms one must be familiar with the simple yet elegant shortest path algorithms. Chapter 3 can be scrutinized for an explanation of the classical shortest path algorithms and their underlying concepts. Some of these concepts may also be used for QoS routing. Chapter 4 immediately plunges into the heart of matter and discusses the concepts inherent to exact QoS routing. Equipped with these concepts, the advantages and disadvantages of different QoS algorithms are better understood. Chapter 5 presents a detailed overview of the lion's share of proposed QoS algorithms. Moreover, a thorough evaluation of these algorithms is presented based on simulations. Such an extensive comparison study has never been undertaken before. Chapters 3 to 5 focus on unicast QoS routing in which the goal is to find a QoS-compliant path between a single source and a single destination. In Chapter 6 the extension from unicast to multicast QoS routing is examined, where the goal is to find QoS-compliant paths from a source to multiple destinations. Chapter 6 points out the problems in multicast QoS routing and proposes a multicast QoS routing algorithm. QoS routing also relates to security, reliability and robustness. In addition to a single path between source and destination, it may be desirable to find a backup path that does not share any links with the primary path. Chapter 7 therefore looks at link-disjoint QoS routing. Again problems and solutions are identified. The extensions to multicast and link-disjoint QoS routing in Chapters 6 and 7 were for the first time examined by us. The main focus of the thesis continues in Chapter 8, which explores the complexity of QoS routing. In the past, the problem of finding a feasible path for QoS routing was shown to be difficult in the worst case, although the precise conditions that constitute this worst case where never identified. Chapter 8 presents pioneering work in this field, resulting in promising conclusions for exact QoS routing in practice. QoS routing does not only consist of appropriate path selection, but also consists of acquiring information on the current state of the network and its link weights. These link weights are typically dynamic in nature, which is the topic of Chapter 9. Chapter 9 presents some preliminary steps on the difficult path towards dynamic QoS routing. Finally Chapter 10 presents the conclusions.

Chapter 2

Graphs, algorithms and complexity

In this chapter basic theory on graphs, algorithms and complexity is provided. This background material is necessary to understand the following chapters.

2.1 Graph theory

The theory of graphs is a large and complex research area. In this section some basic graph theory, graph definitions, and ways to represent a graph are explained.

2.1.1 Graph definitions

There are many definitions in graph theory. The book of Harary [69] is considered a classical reference in the field and many definitions follow his notation.

Definition 3 *Adjacency: Node v is* adjacent *to node u in the graph G if $(u, v) \in E$.*

Definition 4 *Complete graph: In the* complete graph *(also referred to as full mesh) $(u, v) \in E$, $\forall u, v \in V$. K_N denotes the complete graph with N nodes.*

Definition 5 *Connected: A graph G is* connected *if each pair of nodes is connected by a path, otherwise the graph is disconnected. A graph is k-connected if there exist k node-disjoint paths between each pair of nonadjacent nodes.*

Definition 6 *Cycle: A* cycle *is a walk for which all nodes except the first and last are distinct. If there are no cycles in a graph it is called acyclic.*

Definition 7 *Degree: The* degree *of a node u gives the number of adjacent nodes to u. The* degree sequence *of a graph gives for each node the corresponding degree.*

Definition 8 *Path: A* path *is a walk whose vertices are distinct.*

Definition 9 *Planar: A graph is* planar *if it can be embedded in a plane without crossing any links.*

Definition 10 *Regular: A graph is* k-regular *if all nodes have degree* k*.*

Definition 11 *Simple: A graph is* simple *if it does not have any self-loops or parallel links.*

Definition 12 *Tree: A* tree *is a connected acyclic simple graph.*

Definition 13 *Walk: A* walk *in a graph* G *is an alternating sequence* $v_0, e_1, v_1, \ldots, e_k, v_k$ *of nodes* v_i *and links* e_i*, where* e_i *is a link connecting* v_{i-1} *and* v_i*.*

2.1.2 Graph representation

A graph is completely determined by either adjacencies or incidences, which both can be represented in matrix-form.

A link $(u, v) \in E$ is said to be incident to nodes u and v, and vice versa. If the links are numbered from $j = 1$ to M, then the incidence matrix $I[G] = i_{uj}$ of an undirected graph G is obtained as follows:

$$\begin{aligned} i_{uj} &= 1, \text{ if node } u \text{ is incident to link} \\ &= 0, \text{ otherwise} \end{aligned}$$

for all nodes $u \in V$ and all links $j \in E$. If the graph G is a directed graph, the directed link (u, v) from node u to v is said to be incident from u and incident to node v. The incidence matrix $I[G]$ follows as:

$$\begin{aligned} i_{uj} &= +1, \text{ if link } j \text{ is incident to node } u \\ &= -1, \text{ if link } j \text{ is incident from node } u \\ &= 0, \text{ otherwise} \end{aligned}$$

for all nodes $u \in V$ and all links $j \in E$.

If $(u, v) \in E$ then nodes u and v are said to be adjacent. The adjacency matrix $A[G] = a_{uv}$ corresponding to the undirected graph G is defined as:

$$\begin{aligned} a_{uv} &= 1, \text{ if } (u, v) \in E \\ &= 0, \text{ otherwise} \end{aligned}$$

If the graph G is directed, the directed link (u, v) from u to v is in E, then node u is said to be adjacent to node v and node v is adjacent from u. The definition for the adjacency matrix remains the same.

If the graph is dense, then the adjacency matrix is a memory efficient way of representing that graph. If the graph is not dense, then it is more efficient to use linked-lists to identify the adjacencies. This is called the adjacency-list representation. The adjacency-list contains for each node $u \in V$ a list $adj[u]$ with pointers to all nodes that are adjacent to u.

Both the adjacency matrix and adjacency-list are easily adapted to represent weighted graphs. Figure 2.2 exemplifies the adjacency representations for the weighted graph in Figure 2.1.

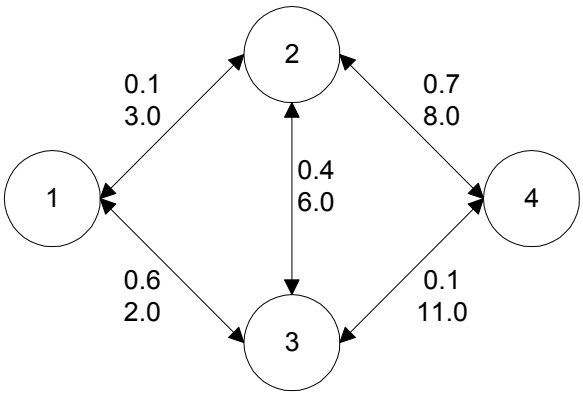


Figure 2.1: An example graph with two weights per link.

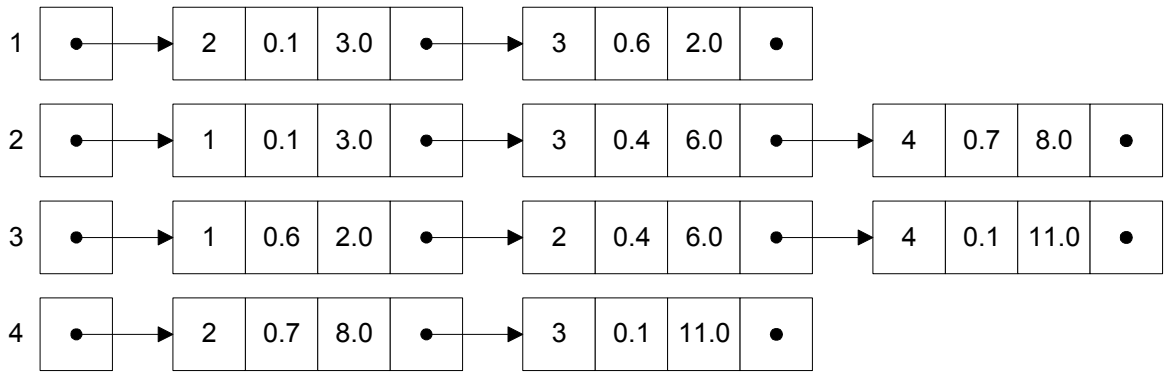


Figure 2.2: Adjacency-list representation of the graph in Figure 2.1.

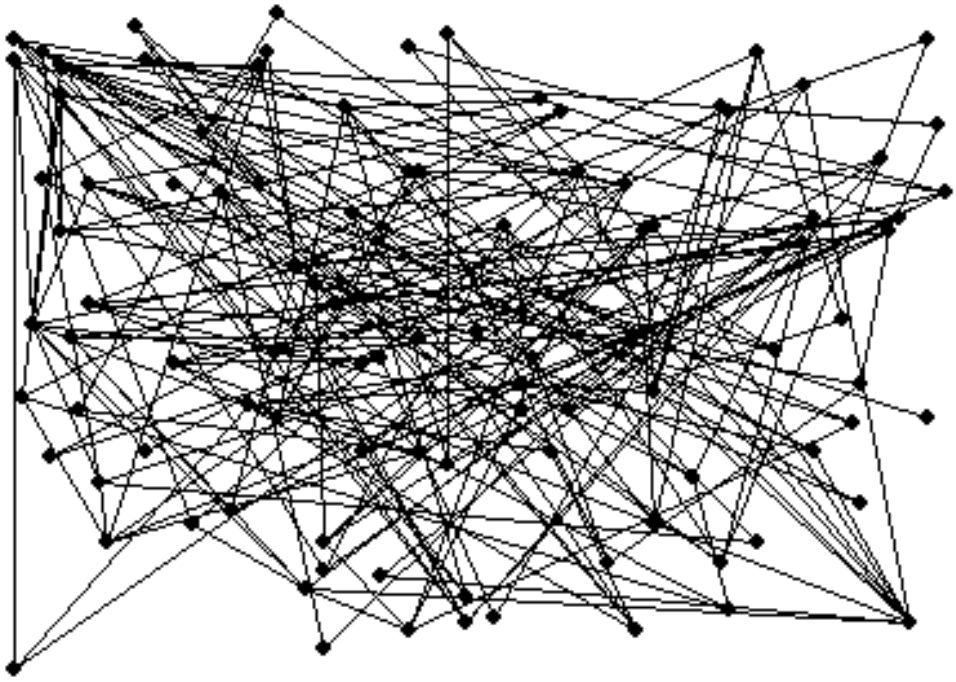


Figure 2.3: An example of a random graph in the class $G_{0.04}(100)$.

2.2 Classes of graphs

In this section four classes of graphs that are relevant to this thesis are discussed: the class of random graphs, Waxman graphs, power-law graphs, and lattices.

2.2.1 Random graph

The classic article on Random graphs is that of Erdös and Rényi [43]. However their random graph model was discovered eight years earlier by Solomonoff and Rapoport [152], but the paper of Erdös and Rényi [43] provides a more in-depth analysis and is therefore best known. The book of Bollobas [21] is a classical reference in the field of random graphs. The simplest model investigated by Erdös an Rényi was the random graph $G_p(N)$ consisting of N nodes. The probability that two nodes in the graph $G_p(N)$ are connected equals p. On average $G_p(N)$ therefore contains $p\frac{N(N-1)}{2}$ links. If $p = 1$ we have the complete graph with the maximum number of links $\frac{N(N-1)}{2}$. The probability of having i adjacent nodes, i.e. the degree distribution, equals the binomial $\binom{N-1}{i}p^i(1-p)^{N-1-i}$, with average degree $d_a = p(N-1)$. For large N the binomial takes the Poisson form $\frac{e^{-d_a}d_a^i}{i!}$. Figure 2.3 gives an example of a random graph.

Erdös and Rényi [44] also identified a phase transition (see Chapter 8.3) in random graphs. The probability that almost every graph $G_p(N)$ is connected is restricted from below by the critical threshold $p_c \sim \frac{\ln N}{N}$ for N large. Thus if $p > p_c$ then almost all graphs $G_p(N)$ are connected, else almost all graphs are disconnected.

Let X_h denote the random variable of the number of paths with h hops between the source node s and the destination node t in $G_p(N)$. Van Mieghem [162] has shown that

$$\mathbb{E}[X_h] = \frac{(N-2)!}{(N-h-1)} p^h, \ 1 \leq h \leq N-1$$

The total number of paths in $G_p(N)$ is obtained by summing over all possible hop counts $\sum_{h=1}^{N-1} X_h$. The maximum number of paths in any graph is upper bounded by the number of paths in the complete graph, which equals $\lfloor e(N-2)! \rfloor$ [161].

2.2.2 Waxman graph

The class of Waxman graphs belongs to the class of random graphs, where the probability of existence of a link between two nodes decays exponentially with the geographic distance between those two nodes. Such graphs are often chosen because of their resemblance to actual network topologies. More formally, the Waxman graphs belong to the class $G_{p_{ij}}(N)$ with $p_{ij} = f(\vec{r}_i - \vec{r}_j)$, where the vector $\vec{r}_i$ represents the position of a node i and all nodes are uniformly distributed in a hyper-cube of size z in the m-dimensional space. The dependence on distance is reflected by $f(\vec{r})$, which is a positive real function of the m coordinates of the vector $\vec{r}$. For example, for the Waxman graph, the distance function is $f(\vec{r}) = e^{-\alpha|\vec{r}|}$, where $|\vec{r}|$ is a norm denoting a distance from the origin. The idea of relating the probability of a link between node i and node j to some function of the distance between those nodes stems from the correspondence with realistic telecommunication networks. The farther two nodes lie separated, the smaller the need for a direct link between them. Figure 2.4 gives an example of a Waxman graph.

2.2.3 Power-law graph

Modelling the Internet topology is an important but difficult problem [50]. At present an accurate model is still missing. However, many topological properties of the Internet seem fairly well captured by power laws [48]. Albert and Barabasi [3] demonstrated via empirical results that also many other complex networks follow power laws. This clearly motivates our interest in power-law graphs as representing realistic network topologies. In power-law graphs the nodal degree distribution is $\Pr[d = i] = ci^{-\tau}$, where c is a constant such that $\sum_{i=1}^{N-1} ci^{-\tau} = 1$. Measurements in the Internet [48] suggest that $\tau \approx 2.4$. Figure 2.5 gives a 100-node example graph drawn from the class of power-law graphs.

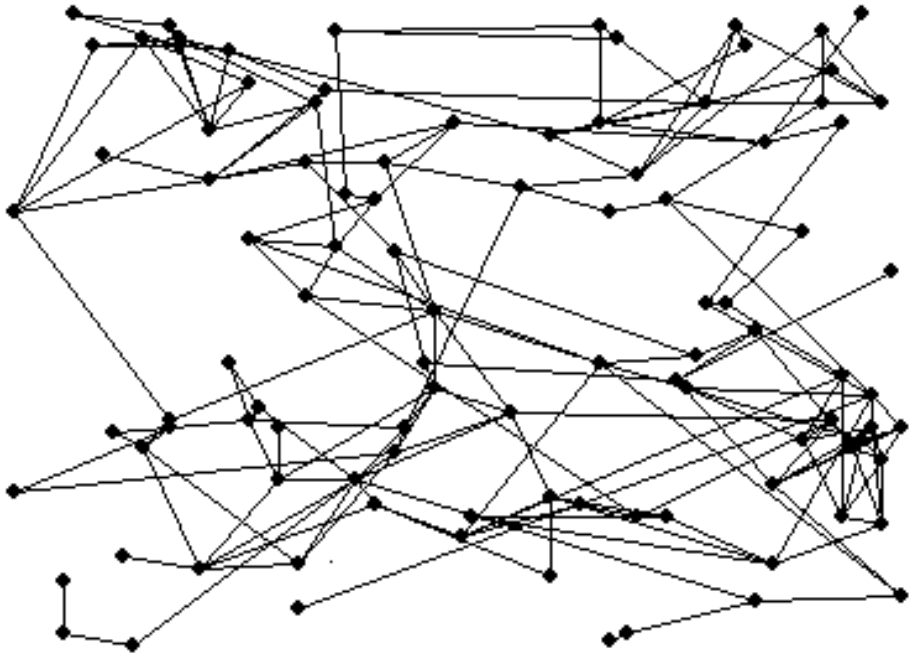


Figure 2.4: An example of a Waxman graph with $N = 100$ nodes.

There are two possible ways of creating power-law graphs. The first is growing a connected power-law graph following some rules of preferential attachment [12]. By growing a graph, often only a sub-class of the class of power-law graphs can be constructed. The second way of generating power-law graphs is by generating a degree sequence from the power-law degree distribution and then creating a graph from this prescribed degree sequence. Asano [8] provided an algorithm to generate a connected graph from a given degree sequence, provided it exists. We have used this approach to generate connected power-law graphs. Unfortunately by removing disconnected graphs, the degree distribution of the connected graphs may be slightly different from the expected distribution, as they favour sequences with high degrees. For our simulation studies this discrepancy can be tolerated.

2.2.4 Lattice

We only consider a subclass of the class of lattices, namely rectangular two-dimensional lattices with size z_1 and z_2 and $N = (z_1 + 1)(z_2 + 1)$.

The class of lattices is extremely regular. In a sense, if we imagine a spectrum of graphs, then the class of random graphs is at one extreme of this spectrum while the

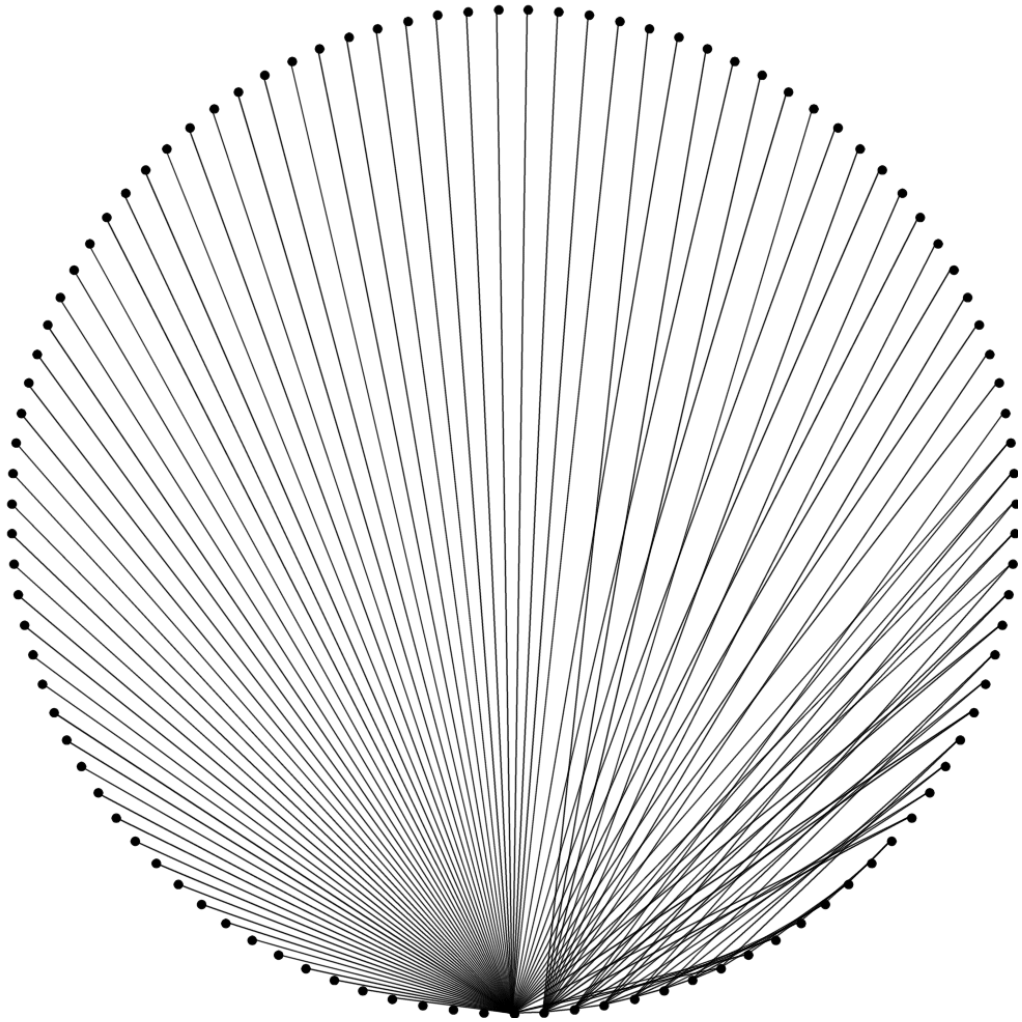


Figure 2.5: An example of a power-law graph with $N = 100$ nodes.

class of lattices is at the other. Figure 2.6 gives an example of a square lattice with $N = 100$ nodes. All interior nodes have degree 4. The shortest-hop path between two diagonal corner points in the rectangular two-dimensional lattice has $h = z_1 + z_2$ hops. Any path in a rectangular two-dimensional lattice can be represented by a sequence of r(ight), l(eft), u(p) and d(own). A shortest-hop path between two diagonal corners consists of z_1 r's (or l's) and z_2 d's (or u's). The total number of such shortest-hop paths equals $\binom{z_1+z_2}{z_1}$.

2.3 Algorithmic complexity

Before explaining the complexity of an algorithm, first the definition of an algorithm is provided. The word "algorithm" originates from the Persian author Abu Ja'far Muhammad ibn Musa Al-Khwarizmi, who wrote a book (around 825 A.D.) on Hindu-Arabic numerals. Unfortunately the original book is lost, but a Latin translation with the title "Algoritmi de numero Indorum" survived, which gave birth to the word "algorithm." The word algebra is also likely to come from the work of Al-Khwarizmi. Several definitions of an algorithm exist:

- Cormen *et al.* [34]: *An algorithm is any well-defined computational procedure that takes some value, or set of values, as input and produces some value, or set of values, as output. An algorithm is thus a sequence of computational steps that transform the input into the output. We can also view an algorithm as a tool for solving a well-specified computational problem. The statement of the problem specifies in general terms the desired input/output relationship. The algorithm describes a specific computational procedure for achieving that input/output relationship.*

- Schrijver [145]: *An algorithm can be seen as a finite set of instructions that perform operations on certain data. The input of the algorithm will give the initial data. When the algorithm stops, the output will be found in prescribed locations of the data set.*

- Merriam-Webster Dictionary: *An algorithm is a procedure for solving a mathematical problem in a finite number of steps that frequently involves repetition of an operation.*

We prefer the description of Cormen *et al.*, as it best captures the algorithms studied in this thesis. In the definition of Cormen *et al.* an abstract problem Q is defined to be a binary relation on a set I of problem instances and a set S of problem solutions. For a computer program to be able to solve an abstract problem, problem instances must be represented in a form that the program understands. Two popular

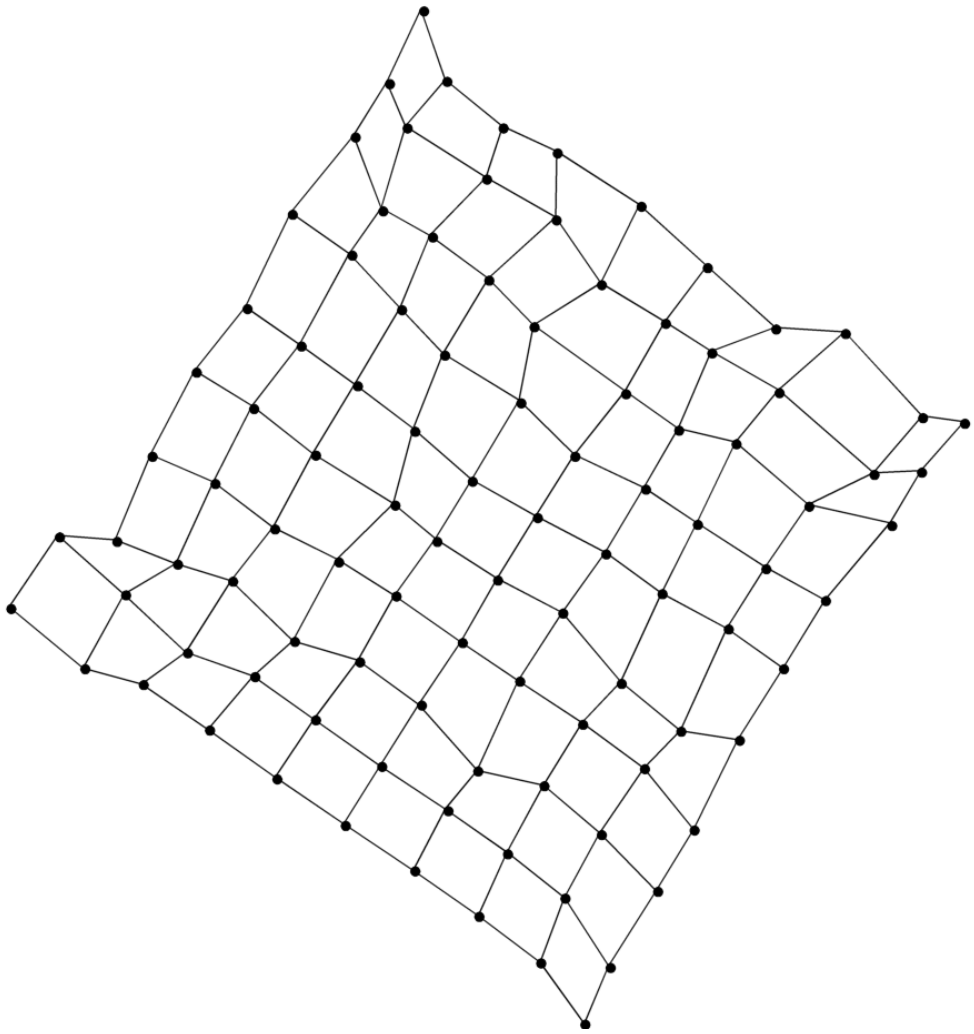


Figure 2.6: An example of a square lattice with $N = 100$ nodes.

computer representations are the Turing machine [160] and the random access machine (RAM). Since current computers, contrary to quantum computers, use binary strings to represent problem instances, this means (among others) that numbers only increase logarithmically with the size of the input (i.e., the length of the binary strings defines the input).

Definition 14 *Polynomial-time algorithm: An algorithm is called a polynomial-time algorithm if it terminates after a number of computational steps bounded by a polynomial in the input size.*

Polynomial-time algorithms are often called efficient algorithms. Note that if the number of steps increases polynomial with some numeric values (e.g., link weights) instead of logarithmically as the input size, this is not a polynomial-time algorithm. Garey and Johnson [57] have named such algorithms "pseudo-polynomial-time algorithms."

The complexity of an algorithm is an important criterion for evaluating algorithms. Formally, *complexity refers to the intrinsic minimum amount of resources needed to solve a problem or execute an algorithm.* Complexity can refer to time-complexity (e.g., polynomial running time) or space-complexity (memory usage). If not specifically stated otherwise, the term complexity refers to time-complexity. Complexity can also be subdivided into average-case complexity, amortized complexity and worst-case complexity.

The worst-case complexity gives an upper bound on the number of computational steps (running time) as a function of the input. The average-case complexity gives the expected running time as a function of the (average) input. The amortized complexity guarantees the average-case complexity in the worst-case. Often the complexity is denoted in the asymptotically most relevant input parameters. The following asymptotic notations are used:

Definition 15 *Θ-notation: $f(x) \in \Theta(g(x))$ as $x \to x_0$ if positive constants c_1 and c_2 exist such that $c_1 g(x) \leq f(x) \leq c_2 g(x)$ for all x sufficiently close to x_0.*

Definition 16 *O-notation: $f(x) \in O(g(x))$ as $x \to x_0$ if a positive constant c exists such that $|f(x)| \leq c|g(x)|$ for all x sufficiently close to x_0.*

Definition 17 *Ω-notation: $f(x) \in \Omega(g(x))$ as $x \to x_0$ if a positive constant c exists such that $cg(x) \leq f(x)$ for all x sufficiently close to x_0.*

Definition 18 *o-notation: $f(x) \in o(g(x))$ as $x \to x_0$, given any $\mu > 0$, we have that $|f(x)| < \mu|g(x)|$ for all x sufficiently close to x_0.*

Definition 19 *ω-notation: $f(x) \in \omega(g(x))$ as $x \to x_0$, given any $\mu > 0$, we have that $0 \leq \mu g(x) < f(x)$ for all x sufficiently close to x_0.*

The Θ-notation refers to upper and lower bounds, while the O-notation and o-notation only refer to upper bounds, and the Ω-notation and ω-notation only refer to lower bounds.

2.4 NP-completeness

In this section we discuss informally the classes P, NP and co-NP. These classes only contain decision problems, of which the solution is either a "yes" or a "no." Many abstract problems are not decision problems, but optimization problems (e.g., the shortest path problem). Luckily, such optimization problems can often be rephrased in polynomial time to a decision problem. This holds for all problems considered in this thesis.

In the previous section a polynomial-time algorithm was defined. *Informally, we can define the class P as the class of decision problems, which are solvable by a polynomial-time algorithm.*

The class NP stands for Nondeterministic Polynomial-time solvable decision problems. The term "nondeterministic" is a heritage from the early days when NP was defined in terms of nondeterministic machines [57]. Nowadays an equivalent, but more simple definition is used:

The class NP is the class of decision problems, whose solutions can be checked/verified by a polynomial-time algorithm.

Finally *the class co-NP is defined as the class of decision problems ∈ NP, for which the complementary problem also belongs to NP.*

The relationship between the classes P and NP is fundamental for the theory of NP-completeness, but due to its complexity this relation is still not fully understood. One obvious relationship that can be deduced from the definitions of P and NP is that P $\subseteq$ NP. The question that still remains unsolved is whether P $\neq$ NP? In [34] four possible scenarios are provided (see Figure 2.7), of which the last scenario (P$\subset$NP$\cap$co-NP) is widely believed to be the most likely. If indeed P $\neq$ NP, then problems in NP\P cannot

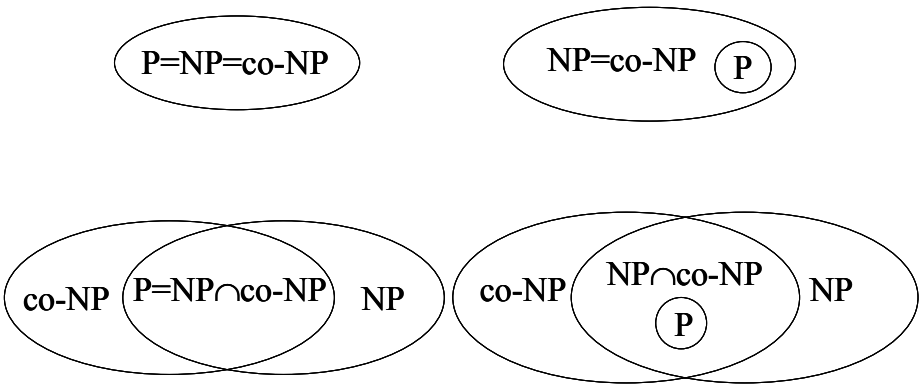


Figure 2.7: Four possible scenarios for the relation between P and NP.

be solved by polynomial-time algorithms and are therefore considered intractable. The class of NP-complete problems is believed to be contained in NP\P (see Figure 2.8).

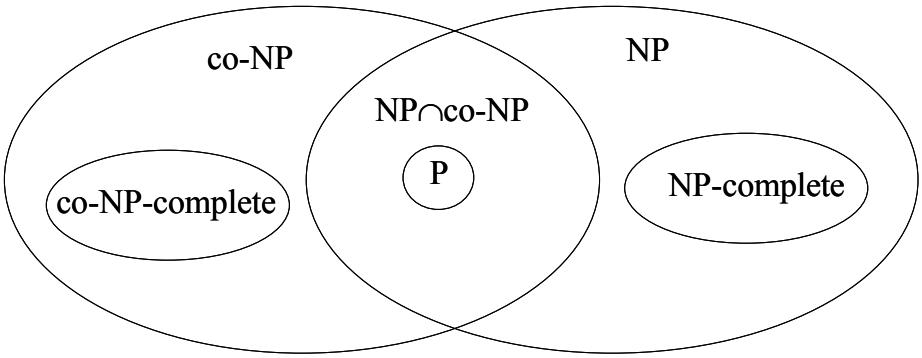


Figure 2.8: A possible scenario for the class of NP-complete problems.

Cook [33] in 1971 introduced the concept of NP-completeness and formulated the first NP-complete problem, referred to as the satisfiability (SAT) problem. A problem Π is defined to be NP-complete if

1. $\Pi \in NP$, and
2. $\Pi' \leq_p \Pi$ for every $\Pi' \in NP$

where $\Pi' \leq_p \Pi$ means that problem Π can be reduced in polynomial-time to problem Π' and therefore by solving Π' we can retrieve the solution to Π in polynomial time. NP-complete problems are the hardest problems in NP and consequently, if any NP-complete problem could be solved in polynomial time, then all NP-complete problems could be solved in polynomial time and P=NP. Conversely, if any problem in NP is not solvable in polynomial time, then all NP-complete problems cannot be solved in polynomial time. Garey and Johnson [57] have discussed several techniques for proving NP-completeness. The simplest technique is called proof by restriction. These proofs consist of showing that problem Π contains as a special case a known NP-complete problem Π'. If this is the case, then problem Π is also NP-complete. This also shows that the theory of NP-completeness is based on a worst-case analysis. In fact, a problem Π could be NP-complete, while a subproblem Π' of Π could be in P.

The main problems considered in this thesis (MCP and MCOP, see Section 1.4) have all been proven to be NP-complete.

Chapter 3

Shortest path algorithms

In this chapter we overview classical (one-dimensional) shortest path algorithms. A selection is made based on the impact the algorithms have had and their relevance to this thesis. In addition to the original papers, three excellent books, [34], [2], and [145], were regularly consulted. These books provide an in-depth coverage of many algorithms. In this chapter we confine to explaining breadth-first search, depth-first search, the Bellman-Ford algorithm, the Dijkstra algorithm, bi-directional search, the A* algorithm, and mathematical programming. The methods of mathematical programming can be generally applied and entire books have been devoted to the subject. Only their applicability as a shortest path algorithm is relevant for this chapter.

The notation used in this chapter is presented in Section 1.3. For our meta-codes we have used the same convention as in [34].

Before the shortest path algorithms are discussed, the shortest path problem is first formally defined:

Definition 20 *Shortest Path (SP) problem: Given a graph $G = (V, E)$, a source node s and destination node t. Each link $(u, v) \in E$ between nodes u and v $(u, v \in V)$ is specified by a single weight $w(u, v) \geq 0$. Find a path P^* from s to t for which $w(P^*) = \sum_{(u,v)\in P^*} w(u, v)$ is minimum, i.e. $w(P^*) \leq w(P)$, $\forall P$.*

In the definition of the shortest path problem we assume the weights to be non-negative and additive, since it is highly unlikely that negative weights will be used in the Internet. Some shortest path algorithms, e.g. Bellman-Ford, can also handle negative link weights provided that there are no negative cycles present. These instances of the SP problem can be solved in polynomial time, however, in general the SP problem (with possibly negative cycles) is an NP-complete problem [2]. If there are negative cycles, walks could traverse these cycles infinitely. It is not simple to prohibit the revisiting of nodes (which is not allowed for a path) and this makes the problem NP-complete. Note that the detection of negative cycles is not an NP-complete problem. The SP

problem is a nice example of a problem that is NP-complete in the worst-case, but which has instances that are solvable in polynomial time. Fortunately these instances $\in P$ are also most relevant in practice. Next, a brief explanation of two elementary graph algorithms, namely breadth-first search and depth-first search is provided.

3.1 Elementary graph algorithms

The meta-code of a basic algorithm for searching a tree in a graph $G = (V, E)$ is displayed in Figure 3.1 and denoted as the search algorithm.

```
SEARCH(G, s)
1. for each node u ∈ V
2.     do color[u] ← WHITE
3.        π[u] ← NIL
4. color[s] ← GREY
5. Q ← {s}
6. while Q ≠ ∅
7.     do u ← extract a node from Q
8.        for each v ∈ Adj[u]
9.            do if color[v] = WHITE
10.               then color[v] ← GREY
11.                   π[v] ← u
12.                   add v to Q
```

Figure 3.1: A search algorithm.

The search algorithm colors the nodes WHITE, GREY or BLACK. All nodes are initially WHITE and become GREY when the search algorithm discovers that node and consequently stores it in the set Q. Q contains all discovered GREY nodes. The search algorithm proceeds by extracting, according to some rule, a GREY node to discover its neighboring nodes that were not yet discovered. Newly discovered nodes are colored GREY and are added to the set Q. The extracted node is colored BLACK and shall not be examined anymore. The search algorithm continues extracting and discovering nodes until the set Q is empty and all nodes have been examined. The predecessor of a node u is stored in the vector $\pi[u]$. This vector can be used to construct the tree rooted at s in the graph G.

The search algorithm in Figure 3.1 does not specify how to extract a node from Q. Different rules lead to different algorithms. The two fundamental strategies are breadth-first search and depth-first search.

3.1.1 Breadth-first search

Breadth-first search uses a first-in first-out (FIFO) rule to extract nodes. Nodes are extracted from the head of the queue Q and stored in the tail of Q. By searching in this breadth-first way, first the nodes with a one-hop distance are colored, then the nodes at a two-hop distance, and continuing up to the nodes at distance H hops, where H gives the maximum hop count between s and any other node in G. The tree returned by breadth-first search is therefore a shortest path tree in terms of hop count. The complexity of breadth-first search is $O(N + M)$ [34].

3.1.2 Depth-first search

Contrary to breadth-first search, depth-first search, as indicated by the name, searches into the depth of a graph. Depth-first search uses a last-in first-out (LIFO) rule, where nodes are inserted at and extracted from the head of the queue Q. Depth-first search therefore explores paths as far as possible and then "backtracks" to initiate a new search until all links have been explored. Depth-first search returns a depth-first forest (possibly) consisting of several depth-first trees. Depth-first search can therefore also easily be used on disconnected graphs and is often used to obtain information on the structure of a graph. The complexity of depth-first search is $\Theta(N + M)$ [34].

3.2 Classical shortest path algorithms

In this section the classical algorithms Bellman-Ford and Dijkstra are described. Both algorithms can return the shortest path tree rooted at a source. A description of these classical algorithms and their implementations can also be found in [54] and for a performance evaluation we refer to [30]. Many variations of these algorithms have been proposed in the literature, mainly based on different proposals for a priority queue [31]. Before exploring the different algorithms, we first give an important property of one-dimensional shortest paths and describe the technique of relaxation that is used by many shortest path algorithms.

Property 21 *Subpaths of shortest paths in one dimension are also shortest paths.*

Proof. We will give a proof by contradiction. Assume that P is the shortest path from s to t, i.e. $P = s, ..., u, ..., v, ..., t$. Let Q be a subpath of P from node $u \in P$ to $v \in P$, i.e. $Q = u, ..., v$. If Q' instead of Q is the shortest path from u to v, we can find the path $P' = s, ..., Q', ...t$ that is shorter than path $P = s, ..., Q, ..., t$ with length $l(P) = \sum_{(u,v)\in P} w(u, v) = w(P_{s\to u})+w(Q)+w(P_{vt}) > w(P_{s\to u})+w(Q')+w(P_{vt}) = l(P')$, which is a contradiction since P is the shortest path from s to t. ■

Property 21 is important, because as will be explained in Chapter 4, the absence of this property in multi-dimensional shortest paths induces many complications.

Property 21 is used in the technique of relaxation to obtain the shortest path length in a monotonically decreasing fashion. Each node $u \in V$ maintains an estimate $d[u]$ of the shortest path distance from the source node s to node u. Based on property 21 we know that subpaths of shortest paths must also be shortest. Therefore, if $d[v] > d[u]+w(u,v)$ we can improve the "shortest" path to v found sofar by going via the node u to node v, using link (u,v). This process of checking whether we can improve the distance estimate of a path to a node v by going via a different path to a neighboring node u and taking the link (u,v), is called relaxing the node v. Initially all estimates $d[u]$ $\forall u \in V$ are set to infinity. In Figures 3.2 and 3.3 the meta-code for the initialization and the relaxation are given. Both Bellman-Ford and Dijkstra use these functions.

```
INITIALIZE(G, s, d, π)
1. for all nodes v ∈ N
2.     d[v] ← ∞
3.     π[v] ← NIL
4. d[s] ← 0
```

Figure 3.2: Initialization.

```
RELAX(u, v, w, d, π)
1. if d[v] > d[u] + w(u, v)
2.     d[v] ← d[u] + w(u, v)
3.     π[v] ← u
```

Figure 3.3: Relaxation.

Lines 1-3 of the INITIALIZE routine in Figure 3.2 set for all the nodes the estimates to infinity and the predecessors to NIL. Only the estimate $d[s]$ of the source node is set to 0 in line 4, since the search is started from the source itself. Line 1 of the procedure RELAX checks whether the distance $d[v]$ can be improved by going via the node u and link (u,v) to node v. If this is the case then the estimate and predecessor of node v are updated in lines 2 and 3.

Figure 3.4 gives a small example to illustrate the routines INITIALIZE and RELAX. Figure 3.4(a) illustrates the initialization. Since s is the source node it also is the starting point of the Bellman-Ford and Dijkstra algorithms. There are two neighboring nodes to s and therefore also two links to relax. Both links pass the relaxation test and therefore their estimates and predecessors are updated (see Figure 3.4(b)). In the

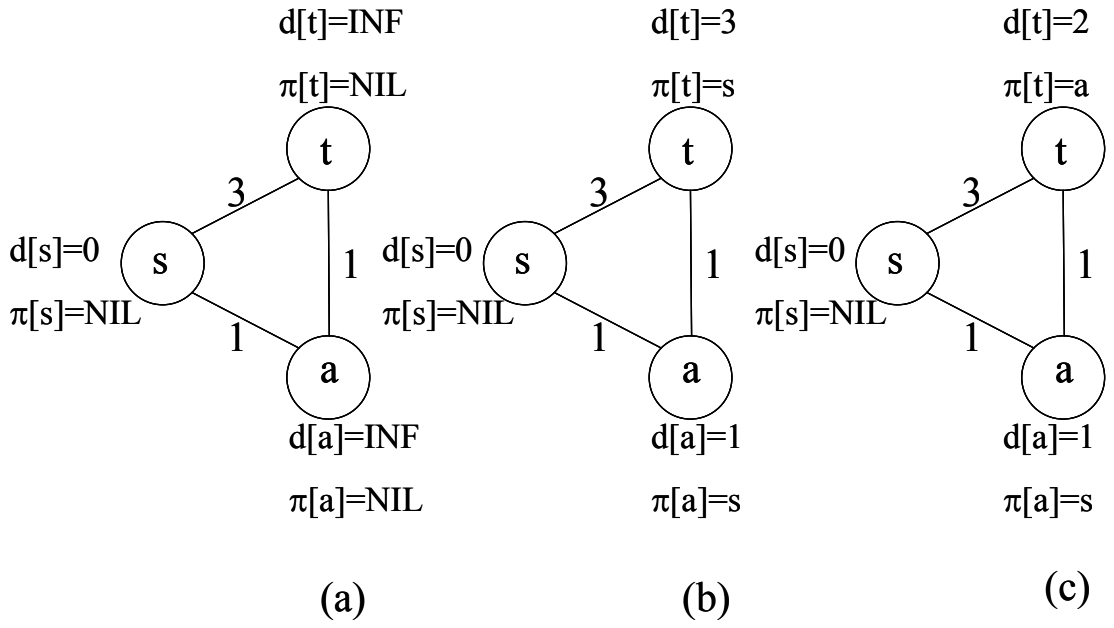


Figure 3.4: Example operation of routines INITIALIZE and RELAX.

next step (Figure 3.4(c)), the link (a, t) can be relaxed and hence the estimate $d[t]$ and concurrently $\pi[t]$ must be updated. The choice of which nodes to examine first is different for Bellman-Ford and Dijkstra and will be discussed below.

3.2.1 Bellman-Ford algorithm

The Bellman-Ford algorithm [17], [89], [117] is based on the Bellman equations (set-up for the complete graph):

$$d^{(0)}[i] = w(i, N),\ i = 1, ..., N$$

and

$$\left.\begin{array}{c} d^{(k+1)}[i] = \min_{j \neq i} \left(w(i,j) + d^{(k)}[j] \right),\ i = 1, ..., N-1 \\ d^{(k+1)}[N] = 0 \end{array}\right\} \text{ for } k = 0, 1, 2, ...$$

where $d^{(i)}[u]$ is the estimate of the shortest path distance from s to u found at the i-th iteration.

Just as in breadth-first search, the Bellman-Ford algorithm traverses the graph by first examining 1-hop paths, then 2-hop paths up to paths with $H \leq N-1$ hops, where H denotes the maximum minimum hop count between any two nodes in the graph. At an iteration h, the algorithm examines whether it can relax a link, obtaining the

shortest paths with at most h hops. The meta-code of the Bellman-Ford algorithm is given in Figure 3.5.

```
BellmanFord(G, w, s)
1. Initialize(G, s, d^(0), π)
2. for h ← 1 to N − 1
3.     do for each link (u, v) ∈ E
4.         do Relax(u, v, w, d^(h), π)
5. for each link (u, v) ∈ E
6.     do if d^(N−1)[v] > d^(N−1)[u] + w(u, v)
7.         then return FALSE
8. return TRUE
```

Figure 3.5: The Bellman-Ford algorithm.

Line 1 initializes the estimates and predecessors of the nodes as indicated in Figure 3.5. Line 2 gives the expansion of the hop count up to the maximum $N-1$. Lines 3 and 4 relax the edges in G. Lines 5 to 8 check if there are no negative-length cycles present. When nonnegative weights are assumed as in definition 20, these lines can be omitted.

We will give a small example on the execution of Bellman-Ford. However, contrary to many books we will not depict the search with graphs, but with an activity table. This gives a better understanding of how an algorithm works once it is programmed in some computer language.

Consider the topology in Figure 3.6. The goal is to find the shortest path from source node $s = 1$ to destination node $t = 8$. The activity table is given in Table 3.1.

The notation x, y in Table 3.1 refers to $d^{(h)}[u], \pi[u]$. The edges are scanned in lexicographic order as follows: $(1,2)$, $(1,3)$, $(2,3)$, $(2,4)$, $(3,4)$, $(3,5)$, $(4,5)$, $(4,6)$, $(4,7)$, $(5,6)$, $(6,7)$, $(6,8)$, $(7,8)$. h in the activity table refers to line 2 in the meta-code (Figure 3.5). At each iteration h all edges $(u,v) \in E$ are relaxed. It may occur that the distance $d[v]$ of a node v is updated multiple times during an iteration. To illustrate this an iteration line is split where appropriate.

The Bellman-Ford algorithm has some drawbacks, for instance by relaxing all nodes at each iteration. In fact, the first h iterations are only relevant for the links E' on the paths that are at most h hops distanced from the source. If h is small it is likely that $E' \subseteq E$ is only a small subset of E and hence many links $E \backslash E'$ are needlessly relaxed. This is why Bellman-Ford best works on sparse graphs.

An other inefficiency stems from the Bellman-Ford equations. They only allow iteration h to use the information of previous $h-1$ iterations. However it may occur that $d^{(h)}[v]$ is decreased by relaxing the edge (u,v) and that this new information is

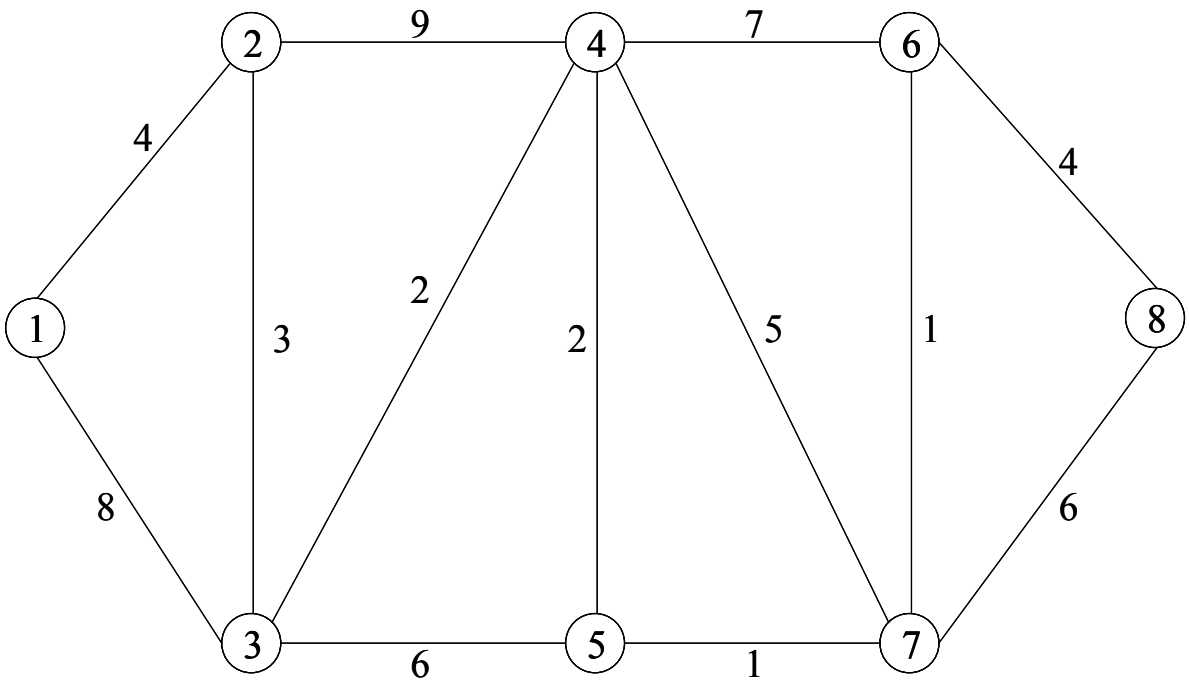


Figure 3.6: Example topology.

not used when relaxing the edge (v, w), which is lexicographically larger than (u, v). By using that new information, some steps in the Bellman-Ford execution could be skipped, thereby increasing performance. However, this destroys the property that at each iteration h we have at most h-hop count paths.

A big advantage of Bellman-Ford is that it can be used as a distributed algorithm, as is done in RIP [75], [115], the first protocol used in the Internet. This distributed capability is easily seen from the Bellman equations.

The Bellman-Ford algorithm has a worst-case complexity of $O(NM)$, since it takes at most $N - 1$ iterations and for each iteration M relaxations.

3.2.2 Dijkstra algorithm

Edsger Wybe Dijkstra is widely considered to be one of the pioneers in shortest path routing. His algorithm [40], simply referred to as the Dijkstra algorithm, has had an enormous impact in many fields.

The Dijkstra algorithm uses the principle of relaxation and the fact that subsections of shortest paths are also shortest paths. For each node $v \in V$ in the graph the Dijkstra algorithm maintains the attribute $d[v]$, which reflects the shortest weight sofar from the

Table 3.1: Activity table

h	1	2	3	4	5	6	7	8
0	0	$\infty, \varnothing$	$\infty, \varnothing$	$\infty, \varnothing$	$\infty, \varnothing$	$\infty, \varnothing$	$\infty, \varnothing$	$\infty, \varnothing$
1	|	4,1	8,1	$\infty, \varnothing$	$\infty, \varnothing$	$\infty, \varnothing$	$\infty, \varnothing$	$\infty, \varnothing$
2	|	|	7,2	13,2	$\infty, \varnothing$	$\infty, \varnothing$	$\infty, \varnothing$	$\infty, \varnothing$
2	|	|	|	10,3	14,3	$\infty, \varnothing$	$\infty, \varnothing$	$\infty, \varnothing$
3	|	|	|	9,3	13,3	$\infty, \varnothing$	$\infty, \varnothing$	$\infty, \varnothing$
3	|	|	|	|	12,4	15,4	17,4	$\infty, \varnothing$
4	|	|	|	|	11,4	14,4	16,4	$\infty, \varnothing$
4	|	|	|	|	|	13,5	16,4	21,6
5	|	|	|	|	|	12,5	14,6	19,6
6	|	|	|	|	|	|	13,6	18,6
7	|	|	|	|	|	|	|	17,7
8	|	|	|	|	|	|	|	|

source node to v. The Dijkstra algorithm also keeps track of the predecessor $\pi[v]$ that is either another node or NIL.

```
DIJKSTRA(G, s, t)
1. INITIALIZE(G, s, d, π)
2. queue Q ← V
3. while Q ≠ ∅
4.     EXTRACT-MIN(Q)→ u
5.     if u = t
6.         return path
7.     else
8.         for each v ∈ adj[u] /*for each neighbor of u*/
9.             RELAX(u, v, w, d, π)
```

Figure 3.7: The Dijkstra algorithm.

Figure 3.7 gives the meta-code of the Dijkstra algorithm. Line 1 of the meta-code initializes all nodes. Line 2 inserts all nodes in the queue Q. The main algorithm starts at line 3. Line 4 extracts the node u from the queue that has the shortest weight (i.e., $d[u] \leq d[v]\ \forall v \neq u \in Q$). Node u can be regarded as the new scanning node towards destination t. Lines 5 and 6 are optional, if the goal is only to retrieve the shortest path between a source and a single destination. The original Dijkstra algorithm does

not have lines 5 and 6 and returns the shortest path tree rooted at source node s. In Figure 3.7, line 5 checks whether the extracted node is different from the destination node t, else the algorithm is finished and the shortest path can be returned via the predecessor list π, running backward from t to s (line 6). Lines 7 to 9 perform the relaxation procedure for each adjacent node v of u.

The worst-case complexity of the Dijkstra algorithm, when using Fibonacci heaps [53], equals $O(N \log N + M)$. The proof that the Dijkstra algorithm is exact, can be found in [34]. If the Dijkstra algorithm is executed on a graph with negative weights, it may occur that the shortest path is not found. By slightly modifying the Dijkstra algorithm to examine/extract nodes multiple times, the shortest path in a graph with negative weights (but no negative cycles) is guaranteed to be found. However, contrary to the Bellman-Ford algorithm, the Dijkstra algorithm may require an exponential running time [88].

3.2.3 Bi-directional search

Nearly all shortest path algorithms proposed, are designed to find a shortest paths tree, rooted at the source to all other nodes. In practice, these algorithms (after a small modification) are often only used to find a path between a single source-destination pair. For instance, a car-navigation system only needs to find the optimal route between the current car position and the intended destination. This particular use of the classical shortest path algorithms may be somewhat inefficient. A potential improvement in computational efficiency stems from the idea to search for the best path alternately from the source s and destination t. We refer to this algorithm as the bi-directional search algorithm. It was first proposed by Dantzig [35] in 1960 and later corrected by Nicholson [122]. The efficiency gain of the bi-directional search algorithm can be significant for some classes of graphs as presented and explained below.

The concept of bi-directional search originated after observing that the Dijkstra algorithm examines a number of "unnecessary" nodes, especially when the shortest (sub)path grows towards the destination. To reduce the number of unnecessary scans, it is better to start scanning from the source node s as well as from the destination node t. Figure 3.8 presents a graph, which was deemed a difficult topology in [30], because the Dijkstra algorithm needs to evaluate all nodes before reaching the destination t from the source s. This situation is circumvented by alternating between scanning from the source node and scanning from the destination node[1]. In that case, a large part of the topology will not be scanned, clearly resulting in a higher efficiency.

Alternating between two directions and meeting in the middle is not always enough to find the shortest path, as illustrated in Figure 3.9. Keeping track of the minimum

[1] In case of a directed graph, the scan-procedure from destination t towards source s should proceed in the reversed direction of the links.

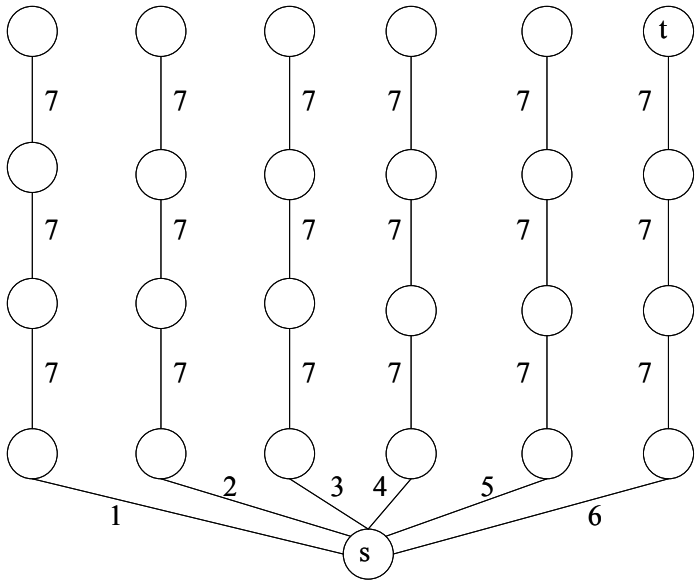


Figure 3.8: Example of a difficult topology for the Dijkstra algorithm.

shortest path length found so far is necessary. Since the Dijkstra algorithm is executed from two sides, two queues Q_s and Q_t, two attributes $d_s[]$ and $d_t[]$, and two predecessor lists π_s and π_t are needed. The bi-directional search algorithm extracts a node u by alternating between Q_s and Q_t. If a node u has been extracted from Q_s and from Q_t and if the end-to-end path length is smaller than or equal to the shortest discovered (but not extracted) path so far, then we have found the shortest path by concatenating the two sub-paths from s to u and u to t.

The meta-code of the bi-directional search (BDS) algorithm is given in Figure 3.10.

Lines 1-4 give the initialization for the two queues. The main algorithm starts at line 7. Lines 8-13 alternate between extracting a node u from Q_s and from Q_t. If a node has been extracted from Q_s and Q_t (i.e., it has been extracted twice in total) and if the end-to-end path length is smaller or equal to the shortest (but not extracted) path sofar (represented by *minlength*), then the shortest path is found and can be returned by concatenating the two subpaths from s to u and u to t. Else, the algorithm proceeds with its alternating search. Depending on whether node u was extracted from Q_s or Q_t, lines 17-21 initiate the relaxation procedure. When necessary, lines 22 and 23 update *minlength*.

The worst-case complexity of this algorithm is identical to that of the Dijkstra algorithm, namely $O(N \log N + M)$.

Property 22 *The bi-directional search algorithm is exact.*

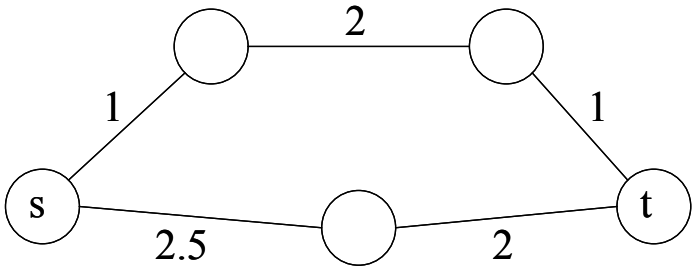


Figure 3.9: Because the number of hops of the shortest (upper) path is unequal, the middle link is counted twice before a node on this path is extracted twice. The end-to-end length of the upper path ($l(P) = 4$) is discovered before the node on the lower path is extracted twice. The length of the lower path ($l(P) = 4.5$) is larger than the length of the upper path and this lower path should therefore not be returned.

Proof. Since the Dijkstra algorithm is exact, we have two shortest path trees, one rooted at the source and the other at the destination. Since subpaths of shortest paths are shortest paths, whenever these two trees touch each other, we will have found the shortest path between source and destination. This shortest path length is stored in *minlength* (line 23). When a node is extracted twice *and* the shortest path has an even number of hops, this means we will have retrieved the shortest path. However, when the shortest path has an odd number of hops, then the middle link will be counted twice, which may result in another (even hop count) path to be extracted (twice) first. However, since the actual shortest path is shorter, *minlength* will also be shorter and hence will not allow the larger path to be returned. ■

The remainder of this subsection is devoted to the performance evaluation of the bi-directional search algorithm. The expected efficiency gain of the bi-directional search algorithm versus the Dijkstra algorithm is compared. The performance measure is based on the average number of extracted nodes of both algorithms,

$$EXN = \frac{1}{T} \sum_{i=1}^{T} \frac{n_{bidirectional}(i)}{n_{Dijkstra}(i)}$$

where T refers to the total number of examined topologies in a particular class of graphs and $n_{bidirectional}(i)$ and $n_{Dijkstra}(i)$ refer to the number of extracted nodes by the bi-directional search algorithm and the Dijkstra algorithm, respectively. We have considered two classes of graphs that possess a different law for the hop count, namely the random graph $G_{0.2}(N)$ and the square lattice.

For the link weights we have used uniformly distributed random variables in the range [0,1). In each class of graphs, a connected graph was generated, in which the

```
BDS(G, s, t)
1. INITIALIZE(G, s, d_s, π_s)
2. INITIALIZE(G, t, d_t, π_t)
3. Q_s ← V
4. Q_t ← V
5. minlength ← ∞
6. alternate ← 1
7. while Q_s ≠ ∅ and Q_t ≠ ∅
8.       if alternate = 1
9.             EXTRACT-MIN(Q_s) → u
10.            alternate ← 0
11.      else
12.            EXTRACT-MIN(Q_t) → u
13.            alternate ← 1
14.      if u ∉ {Q_s, Q_t} and d_s[u] + d_t[u] ≤ minlength
15.            return path
16.      else
17.            for each v ∈ Adj[u]
18.                  if alternate = 0
19.                        RELAX(u, v, w, d_s, π_s)
20.                  else
21.                        RELAX(u, v, w, d_t, π_t)
22.                  if d_s[v] + d_t[v] < minlength
23.                        minlength ← d_s[v] + d_t[v]
```

Figure 3.10: Bi-directional search algorithm.

shortest path between two randomly chosen (different) nodes was calculated with both algorithms. At each calculation, the number of extracted nodes was stored. This procedure was repeated $T = 10^4$ times. The results for both classes of graphs, for different sizes of N, are plotted in Figure 3.11.

Figure 3.11 shows that for very small graphs (i.e., $N \leq 25$) the Dijkstra algorithm is more efficient than the bi-directional search algorithm. However, for larger graphs the bi-directional search algorithm displays a higher efficiency. For uniform link weights, this efficiency gain is smallest for the class of two-dimensional lattices, where it seems to saturate at 62% of Dijkstra's extracted nodes. This gain is already substantial. However, the efficiency gain in the class of random graphs is much larger and continues to decrease with N in our simulated range. The reason most likely lies in the random structure of the graph, which makes it more probable for the Dijkstra algorithm to scan

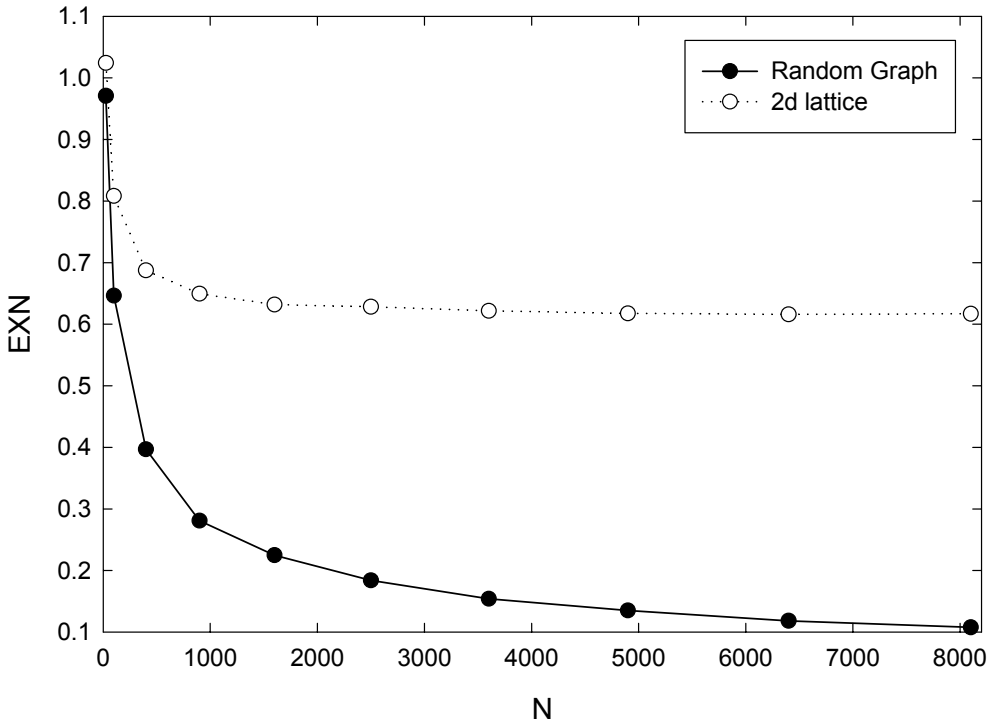


Figure 3.11: EXN as a function of N for the class random graphs and the two-dimensional lattices.

nodes that are outside of "the scope" of the shortest path.

We present an order estimate for the gain plotted in Figure 3.11. The shortest path tree in the class of random graphs $G_p(N)$ with independent exponential or uniformly distributed link weights is a uniform recursive tree (URT) [164]. A URT grows by attaching a new node uniformly to any of the already existent nodes in the URT. In the bi-directional search algorithm, two separate URTs are grown, URT_s and URT_t rooted at source s and destination t, respectively. The scanning processes create two URTs of about equal size. Let v denote the typical size of the URTs when they meet. When the two URTs meet, any node in URT_s can be potentially attached to any node in URT_t, corresponding roughly to v^2 possibilities. This implies that both URTs are connected if the number of interconnection possibilities includes about all nodes, hence, $v^2 = O(N)$ from which $v = O\left(\sqrt{N}\right)$. This estimate explains that the performance measure EXN decreases roughly as $O(N^{-0.5})$, where simulations (up to $N = 8000$) give $EXN = O(N^{-0.45})$.

The argument why $EXN \rightarrow 0.6$ in the two-dimensional lattice is as follows. Since the link weights are uniformly distributed, we may expect that the shortest path tree grows as an isotropic diffusion process with radius r around s and t, respectively. The number of nodes in each circle is about αr^2, where α is a constant. When these two circles touch each other, the shortest path is found which happens if $2r$ is about the distance R between s and t. In case of the Dijkstra algorithm, only the circle from s finds the shortest path if node t is enclosed, which needs about αR^2 nodes to be discovered. Hence, in the bi-directional search algorithm about $2\alpha r^2$ nodes need to be discovered, while $\alpha R^2 = 4\alpha r^2$ in the Dijkstra process, which leads to a gain factor of 0.5. The actually simulated value is somewhat less, a gain of about 0.4, which is mainly due to neglecting the effect of the finite boundaries in the square lattice. In summary, by taking into account the underlying graph structure, it is possible to estimate roughly the performance measure EXN or efficiency gain of the bi-directional search algorithm over the Dijkstra algorithm.

3.3 Best-first search

Best-first search can be seen as an important technique that lies somewhere between breadth-first search and depth-first search. Best-first search predicts the quality of the set of (sub)paths. It first explores the "best" subpaths, which seem most likely to lead to the shortest path. The estimations are computed via an evaluation length function $l()$, which may depend on the current node u, the destination, the information gathered by the search up to node u and on any extra knowledge about the problem domain [131]. The estimates are heuristic in nature and may therefore be misleading. The different types of best-first search algorithms differ in the evaluation function they employ. Two well known best-first search algorithms are the Dijkstra algorithm (see Section 3.2.2), which uses the weights of the subpaths as estimates for the best end-to-end path length and the A* algorithm, which will be described below.

3.3.1 A* algorithm

The A* algorithm was described formally for the first time by Hart *et al.* [71], [72]. The length function $l()$ that A* uses is composed out of two parts, namely the weight $d[u]$ of a subpath from s to some node u, plus a lower-bound function $b(u)$ that bounds the weight of the shortest path from u to t. The values of $d[u]$ (with $d[s] = 0$) can be overestimated, but will decrease monotonically towards the weight $w(P^*_{s\rightarrow u})$ of the shortest path $P^*_{s\rightarrow u}$ from s to u. The closer the lower bounds $b(u)$ are to $w(P^*_{u\rightarrow t})$, the better informed the A* algorithm is. If $b(u) = 0$, then the A* algorithm reduces to the Dijkstra algorithm, also referred to as blind uniform cost algorithm. Assume we have two A* algorithms: A^*_1 and A^*_2, with lower-bound functions $b_1()$ and $b_2()$, respectively.

If $b_1(u) > b_2(u)\ \forall u \in V \backslash \{t\}$, then A_1^* is better informed and hence more efficient than A_2^*. Of course, the difficulty lies in obtaining lower bounds that are as accurate as possible while still easily computable. For a good discussion of the properties of the A* algorithm we refer to [131].

As a possible application of the A* algorithm, let us consider an electronic route-planner in a car that needs to find the shortest route between a starting point (e.g., Delft) and a destination (e.g., The Hague). The route-planner is equipped with a detailed road map of The Netherlands. We could use an A* algorithm with $b(u) = 0$ $\forall u \in V$ (i.e. the Dijkstra algorithm), but in this case better lower bounds can be obtained by using the geographical distances between points in the road map and the destination. Only if there exists a straight road between a point u and the destination will the geographical distance $b(u)$ equal $w(P^*_{u \to t})$, else $b(u) < w(P^*_{u \to t})$.

Since $b(u) \geq 0\ \forall u \in V$ this approach outperforms the Dijkstra approach by sooner directing the search in the correct direction.

Like the bi-directional search algorithm (see Section 3.2.3), it is also possible to make a bi-directional variant of the A* algorithm [135], which can be even more efficient in finding the shortest path between two nodes in a graph.

3.4 Mathematical programming

A mathematical programming problem is a problem with the objective to minimize (or maximize) a real-valued function of real or integer variables, often subject to constraints on the variables. In mathematical form:

Find $\vec{X} = \{x_1, ..., x_n\}$, which minimizes (maximizes) $f(\vec{X})$ subject to constraints (if any)

$$\begin{aligned} g_j(\vec{X}) &\leq 0,\ j = 1, 2, ..., q \\ h_j(\vec{X}) &\leq 0,\ j = 1, 2, ..., p \end{aligned}$$

where $\vec{X}$ is an n-dimensional vector, $f()$ the objective function, $g()$ the inequality constraints and $h()$ the equality constraints. The term "programming" therefore does not refer to computer programming, which it predates, but to optimization.

Some of the most used techniques that are classified under mathematical programming are linear programming, integer programming, non-linear programming and dynamic programming. Linear and dynamic programming are briefly discoursed upon in light of the shortest path problem. More mathematical programming techniques are described in [137].

3.4.1 Linear programming

By formulating the general linear programming (LP) problem and developing the simplex method, Dantzig [36] laid the cornerstones for a wide applicability of linear programming.

As the name implies, in linear programming the objective function and the constraints appear as linear functions over a polyhedron. A subset R of $\mathbb{R}^p$ is called a polyhedron if there exists a $p \times p$ matrix A and a vector $b \in \mathbb{R}^q$ (for some $q \geq 0$), such that $R = \{x|\ Ax \leq b\}$. There are several possible ways in which a linear programming problem can be formulated, but the standard form is as follows:

Minimize

$$\sum_{j=1}^{p} c_j x_j$$

subject to the constraints

$$\begin{aligned} \sum_{j=1}^{p} a_{ij} x_j &= b_i,\ i = 1, 2, ..., q \\ x_j &\geq 0,\ j = 1, 2, ..., p \end{aligned}$$

or in matrix form:

Minimize

$$c^T_{(p\times 1)} x_{(p\times 1)}$$

subject to the constraints

$$\begin{aligned} A_{(q\times p)} x_{(p\times 1)} &= b_{(q\times 1)} \\ x_{(p\times 1)} &\geq 0_{(p\times 1)} \end{aligned}$$

Any linear programming problem can be written in the standard form by using appropriate transformations [137]. We can transform "maximize f" into "minimize $-f$," we can introduce slack variables y_i such that $\sum_{j=1}^{p} a_{ij} x_j \leq b_i \Rightarrow \sum_{j=1}^{p} a_{ij} x_j + y_i = b_i$ and if x_i is unrestricted in sign, it can be written as $x_i = x_i^+ - x_i^-$, where $x_i^+ \geq 0,\ x_i^- \geq 0$.

The following maximization example is used to illustrate the search in linear programming:

Maximize $x_1 + x_2$ subject to

$$\begin{aligned} x_1 + 2x_2 &\leq 5 \\ x_1 &\leq 3 \\ x_2 &\leq 2 \\ x_1, x_2 &\geq 0 \end{aligned}$$

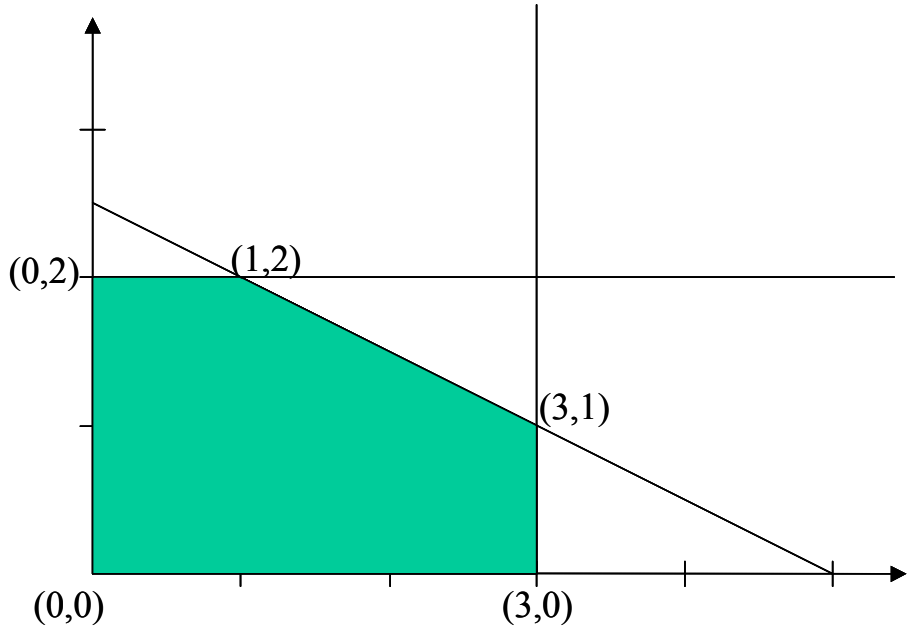


Figure 3.12: Feasible polyhedron

The feasible region is colored grey in Figure 3.12. This region is called a polyhedron and has five extreme points, namely $(0,0), (3,0), (3,1), (1,2), (0,2)$.

To find the point which maximizes $x_1 + x_2$ in the polyhedron, we slide the line $x_1 + x_2$ outwards from the origin $(0,0)$ as far as possible until we encounter the extreme point $(3,1)$. This is the solution, because sliding the line any further means leaving the polyhedron. It is not a coincidence that the solution is an extreme point, because a linear programming problem always has an extreme point as (one of) its optimal solution(s). In fact, this property is extensively used by the simplex method. Non-extreme points cannot be returned by the simplex method, but only by interior point methods. Before explaining how to solve the SP problem through LP, we want to mention one last property of linear programming, namely that of duality. For more properties/theorems we refer to the literature, e.g. [36], [145], [137], [139].

The basic idea of duality is that every "primal" LP problem has associated with it another "dual" problem, which are closely interrelated. Given that the primal problem is stated in the following form:

Minimize

$$\sum_{j=1}^{p} c_j x_j$$

subject to

$$\begin{aligned} \sum_{j=1}^{p} a_{ij}x_j &\geq b_i,\ i = 1, 2, ..., q \\ x_j &\geq 0,\ j = 1, 2, ..., p \end{aligned}$$

Then the dual problem is obtained as follows:
Maximize

$$\sum_{i=1}^{q} b_i y_i$$

subject to

$$\begin{aligned} \sum_{i=1}^{q} a_{ij}y_i &\leq c_i,\ j = 1, 2, ..., p \\ y_i &\geq 0,\ i = 1, 2, ..., q \end{aligned}$$

From the above equations it is easily seen that the dual of a dual problem equals the primal problem. Two fundamental theorems that describe the relation between the primal and dual problems are referred to as the weak duality theorem and the strong duality theorem. These theorems are provided here, but for the proofs we refer to the literature (for instance [2]).

Theorem 23 *Weak duality theorem: If x is any feasible solution of the primal problem and y is any feasible solution of the dual problem, then $\sum_{i=1}^{q} b_i y_i \leq \sum_{j=1}^{p} c_j x_j$.*

Theorem 24 *Strong duality theorem: If any one of the pair of primal and dual problems has a finite optimal solution, so does the other one and both have the same objective function values.*

After having discussed the linear programming fundamentals, we will now apply the LP technique to solve the shortest path (SP) problem. Actually, we will only write the SP problem as an LP problem, after which a linear programming solver (such as simplex) can be used to solve the problem.

To each directed link (i, j), we associate an N-dimensional (row)vector $\vec{v}_{ij}$ consisting of zeros except for -1 at the i-th component and $+1$ at the j-th component, thus

$$\vec{v}_{ij} = (0, ..., 0, -1, 0, ..., 0, 1, 0, ..., 0)$$

The total of all link vectors $\vec{v}_{ij}$ forms an incidence matrix $A_{M\times N}$. An important property of A is that each row sum equals zero.

We denote the link weight (resistance) vector by $\vec{c}$ or $c_{M\times 1}$where

$$\vec{c} = (w(1,2), ..., w(1,j), ..., w(1,N), w(2,1), w(2,3), ..., w(N-1,N))$$

Consider first the flow problem specified by a demand vector $d_{N\times 1}$, where the component $d_j = -1$ if unit traffic is injected at node j into the network, $d_j = 0$ if neither traffic enters nor leaves the network and $d_j = 1$ if unit traffic leaves the network at node j. Assume for simplicity first that

$$d_{N\times 1} = (-1, 0, ..., 0, 1, 0, 0, ..., 0)$$

which represents the situation where only in the sender (node 1) traffic is injected and at a receiver (node j) the traffic leaves the network. Let the vector $\vec{x}$ (or in different notation $x_{M\times 1}$) denote the traffic flow vector in each link with $x_j = 1$ if a flow passes through link j, else $x_j = 0$. At each node j, the sum of input flows and output flows must be equal to d_j. In matrix form,

$$A^T x = d$$

The interest lies in finding the traffic flow vector that minimizes the total weight $c^T x = \sum_{(i,j)\in P} w(i,j)x_{ij}$. This problem is called the *primal formulation* of the shortest path problem in the linear programming framework,

$$\min_{\vec{x}} c^T x \quad \text{subject to} \quad A^T x = d \quad \vec{x} \geq 0$$

If the objective function and/or the constraints cannot be expressed in linear functions of the decision variables, we have to resort to a different optimization technique (non-linear programming).

When all/some variables are constrained to take integer values, we refer to such optimization problems as all/mixed integer programming problems.

3.4.2 Dynamic programming (Floyd-Warshall algorithm)

Richard Bellman [16] has provided us with the foundations of dynamic programming. Dynamic programming was designed as a tool for optimally solving a specific class of decision problems. These dynamic programming problems are characterized by their hierarchical structure. Similarly to divide-and-conquer algorithms [34], dynamic programming subdivides (sub)problems into manageable pieces, which are solved and then used to retrieve the final solution to the overall problem. Contrary to divide-and-conquer, the "manageable" pieces can be used by multiple parent (sub)problems.

Dynamic programming relies on the principle of optimality [16]:

An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.

This means that an optimal solution to a problem must be based on optimal solutions to the subproblems of this problem. This means that dynamic programming can be regarded as a memoryless process, because solutions to subproblems only rely on solutions to subsubproblems, but not to earlier solutions. The best way to explain dynamic programming is most likely through examples. Here we confine to describing the Floyd-Warshall algorithm, which is a dynamic programming algorithm that finds the shortest paths between all pairs of nodes in the graph.

The Floyd-Warshall algorithm was proposed by Floyd [49], who based it on a theorem of Warshall [173]. The Floyd-Warshall algorithm considers subsets $V(k) = \{1, 2, ..., k\}$ of the set $V = V(N) = \{1, 2, ..., N\}$ of all nodes. $d^{(k)}[u, v]$ represents the length of the shortest path from u to v, given that the intermediate nodes are taken from the set $V(k)$. $d^{(0)}[u, v]$ therefore refers to the weight of the direct link between u and v, if it exists. If the intermediate nodes of the shortest path between u and v are present in the set $V(k-1)$, they will also be present in the set $V(k)$. Consequently, if all N nodes are used: $V(N) = V$, and then all shortest paths between all pairs of nodes are found. The dynamic programming requirement of a hierarchical structure is therefore present and can be represented in a recursive formula:

$$d^{(k)}[u, v] = \min\left(d^{(k-1)}[u, v], d^{(k)}[u, k] + d^{(k)}[k, v]\right)$$

The two terms in the min operator respectively signify that either node k is not an intermediate node on the shortest path from u to v or node k is an intermediate node on the shortest path from u to v. The predecessor $\pi^{(k)}[u, v]$ of node v on the shortest path from u to v contains the intermediate nodes from the set $V(k)$. The meta-code is given in Figures 3.13 and 3.14. $D^{(i)}$ denotes the matrix of the components $d^{(i)}[u, v]$, $\Pi^{(i)}$ denotes the matrix with components $\pi^{(i)}[u, v]$, $INF[N \times N]$ is the $N \times N$-matrix with as components all ∞, and $NIL[N \times N]$ is the matrix with NIL as components.

The Floyd-Warshall algorithm has a complexity of $\Theta(N^3)$. More efficient algorithms for the all-pairs shortest paths problem exist, however only Floyd-Warshall was explained to illustrate a dynamic programming application.

```
INITIALIZEFW(G, D^(0), Π^(0))
1. for u ← 1 to N
2.     for v ← 1 to N
3.         if (u, v) ∈ E
4.             then d^(0)[u, v] ← w(u, v)
5.                  π^(0)[u, v] ← u
6. do d^(0)[u, u] ← 0
7.    π^(0)[u, v] ← NIL
8. return D^(0), Π^(0)
```

Figure 3.13: Initialization for the Floyd-Warshall algorithm.

```
FLOYD-WARSHALL(G)
1. D^(0) ← INF[N × N]
2. Π^(0) ← NIL[N × N]
3. INITIALIZEFW(G, D^(0), Π^(0))
4. for k ← 1 to N
5.     do for u ← 1 to N
6.         do for v ← 1 to N
7.             d^(k)[u, v] ← min (d^(k−1)[u, v], d^(k)[u, k] + d^(k)[k, v])
8. return D^(N), Π^(N)
```

Figure 3.14: The Floyd-Warshall algorithm.

Chapter 4

Concepts of exact MCP algorithms

Chapters 2 and 3 have provided an introduction into the field of algorithms and complexity. This basic theory is the first step in accomplishing the main goal of this thesis, which is to find an algorithm that can solve the MCP problem (finding a path subject to multiple constraints) efficiently in practice. See Section 1.4 for a formal definition of the MCP problem.

Although usually first the state-of-the-art in algorithms is reviewed, we take a less conventional approach. This chapter starts by discussing some fundamental concepts of exact MCP algorithms. This discussion embodies some of our key research results and will lead to a better understanding of the state-of-the-art in MCP algorithms, which are evaluated in Chapter 5. Note that the list of concepts presented here is not exhaustive and more concepts may exist that might improve MCP algorithms. The first concept discussed is that of the path length function.

4.1 Definition of the path length $l(P)$

The *weight* vector of a path as defined in (1.1) is a vector-sum. As in linear algebra, the *length* of a (path) vector requires a vector norm to be defined. The presented framework applies for *any* definition of length $l(.)$ that obeys the vector norm criteria: (a) $l(\vec{p}) > 0$ for all non-zero vectors $\vec{p}$ and $l(\vec{p}) = 0$ only if $\vec{p} = 0$, (b) for all vectors $\vec{p}$ and $\vec{u}$ holds the triangle inequality $l(\vec{p} + \vec{u}) \leq l(\vec{p}) + l(\vec{u})$. If $\vec{p}$ and $\vec{u}$ are non-negative vectors (i.e., all vector components are non-negative), then $l(\vec{p} + \vec{u}) \geq l(\vec{p})$, because the length of a non-negative vector cannot decrease if a non-negative vector is added. The definition of the path length $l(P)$ is needed to be able to compare paths, since the link weight components all reflect different QoS measures with specific units.

First we review the straightforward choice of a linear path length as proposed by Jaffe [85],

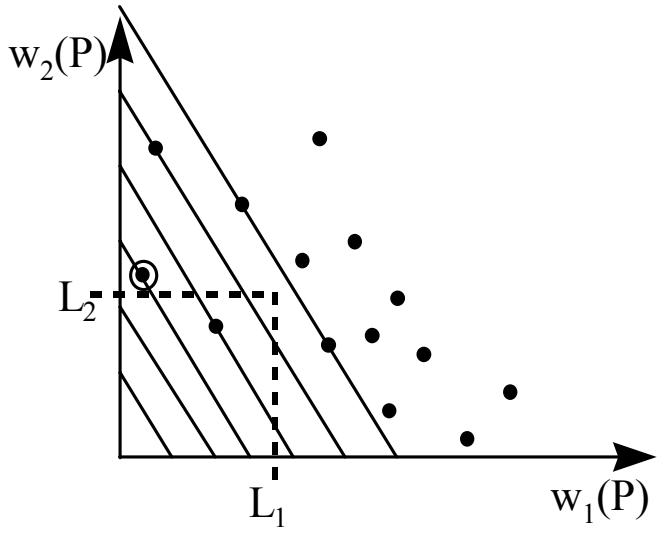


Figure 4.1: In $m = 2$ dimensions, each path P between the source node and the destination node has a point representation in the $(w_1(P), w_2(P))$-plane. The parallel lines shown are equilength lines $d_1 w_1(P) + d_2 w_1(P) = l$, which contain solutions with equal length l. Clearly, all solutions lying above a certain line have a length larger than the ones below or on the line. The shortest path returned by Dijkstra's algorithm applied to the reduced graph, is the first solution (encircled) intersected by a set of parallel lines with slope $-\frac{d_1}{d_2}$. In this example, the shortest path (encircled) lies outside the constraints area.

$$l(P) = \sum_{i=1}^{m} d_i w_i(P) = \vec{d}.\vec{w}(P) \tag{4.1}$$

where d_i are positive real numbers. By replacing the link weight vector $\vec{w}\,(u, v)$ of each link (u, v) in the graph G by the single metric $\vec{d}.\vec{w}(u, v)$ according to (4.1), the m-parameter problem is transformed to a single parameter problem enabling the use of Dijkstra's shortest path algorithm. The Dijkstra algorithm applied to the reduced graph will return a path P that minimizes $l(P)$ defined by (4.1).

When scanning the solution space with a straight equilength line $l(P) = l$ as in Figure 4.1, the area scanned outside the constraint region is minimized if the slope of the straight equilength lines satisfies $\frac{d_1}{d_2} = \frac{L_2}{L_1}$. In m-dimensions, the largest possible volume of the solution space that can be scanned subject to $w_i(P) \leq L_i$ is reached for the plane which passes through the maximum allowed segments L_i on each axis. The equation of that plane is $\sum_{i=1}^{m} \frac{w_i(P)}{L_i} = 1$. Hence, the best choice in (4.1) is $d_i = \frac{1}{L_i}$, for all $1 \leq i \leq m$. In that case, half of the constraints volume is scanned before a solution

outside that volume with $l(P) > 1$ can possibly be selected. In addition, this optimum choice also normalizes each component $w_i(P)$ by L_i (in a specific unit). In spite of the advantage that the simple Dijkstra shortest path algorithm can be used, the drawback of (4.1) is that the shortest path does not necessarily satisfy all constraints.

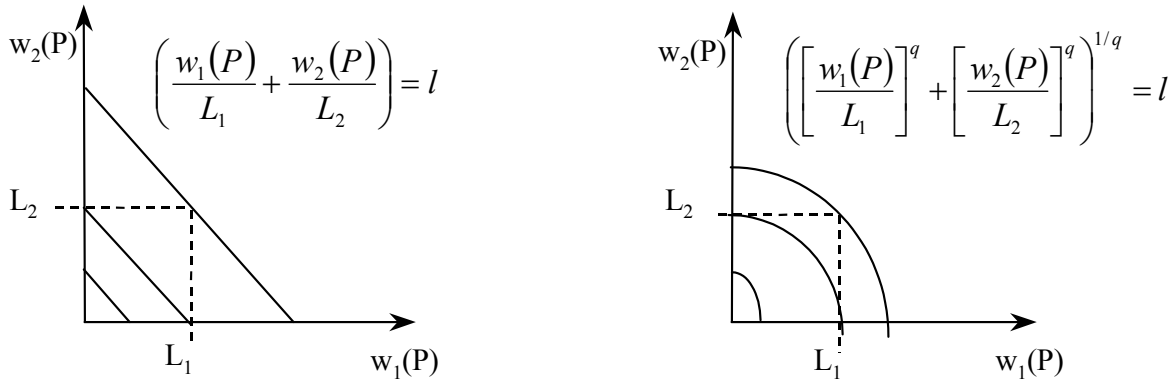


Figure 4.2: Illustration of curved equilength lines.

As illustrated in Figure 4.2, curved equilength lines match the constraint boundaries much better. The non-linear definition,

$$l_q(P) = \left(\sum_{i=1}^{m} \left[\frac{w_i(P)}{L_i} \right]^q \right)^{\frac{1}{q}} \tag{4.2}$$

is well-known as Holder's q-vector norm [60] and is fundamental in the theory of classical Banach spaces (see Royden [142, Chapter 6]). Obviously, the best match is obtained in the limit when $q \to \infty$, since then the equilength lines are rectangles precisely conform to the constraint boundaries (see Figure 4.3). In that case, the definition (4.2) reduces to the maximum vector component divided by the corresponding constraint,

$$l_\infty(P) = \max_{1 \le i \le m} \left[\frac{w_i(P)}{L_i} \right] \tag{4.3}$$

If the shortest path, computed with length definition (4.3), has length larger than 1 and, hence, violates at least one of the constraints, no other path will satisfy the constraints. Thus, finding the shortest path with the definition (4.3) of path length solves the multi-constrained path (MCP) problem. However, the shortest path is not guaranteed to be found with Dijkstra's algorithm, which relies on property 21 of a linear path length definition that *subsections of shortest paths are also shortest paths.*

Theorem 25 *In multiple dimensions and using a non-linear definition of the path length, the subsections of shortest paths are not necessarily shortest paths.*

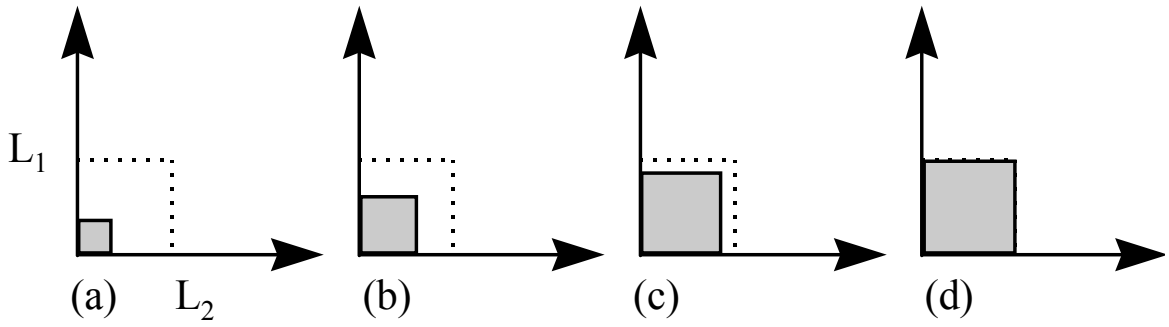


Figure 4.3: Equilength lines are rectangles conform to the constraint boundaries.

Proof. The proof relies on the inequality $l(P+Q) \leq l(P)+l(Q)$. Consider two paths P_1 and P_2, for which $l(P_1) < l(P_2)$ and assume that, by adding a same link a to both paths, the paths P_3 and P_4 can be constructed. Let us first focus on the case where the equality sign holds, typically if $q = 1$ in (4.2). By construction, we have $l(P_3) = l(P_1 + a)$ and by the equality sign, $l(P_1+a) = l(P_1)+l(a)$ and analogously, $l(P_4) = l(P_2+a) = l(P_2)+l(a)$. Since $l(P_1) < l(P_2)$, there holds that $l(P_3) < l(P_4)$ or, the subpaths of shortest paths with linear definition of path length are again shortest paths, leading to the well-known and intuitive result. When the inequality sign holds in $l(P+Q) \leq l(P)+l(Q)$, typically if $q > 1$ as readily verified from (4.2), we arrive in a similar fashion to the set of inequalities $l(P_4) = l(P_2 + a) < l(P_2) + l(a)$; $l(P_3) = l(P_1 + a) < l(P_1) + l(a)$ and $l(P_3) < l(P_4)$. However, from this set, it cannot be concluded whether $l(P_3) \leq l(P_4)$ or $l(P_3) > l(P_4)$. It suffices to show that the latter situation may exist in order to prove the theorem. This can be illustrated via the example graph of Figure 4.4. For the constraints $(14, 11, 22)$, the shortest path from node a to node i runs over node c, e and f. According to the definition (4.3), the path length of $P(i - f - e - c - a)$ equals 0.95, which means that it satisfies all constraints. However, as illustrated in Figure 4.4, the shortest path from node a to node e does not traverse node c but node b. ■

4.1.1 Different (non-linear) length functions

Depending on the constrained optimization problem, we can use different length functions, provided they obey the criteria for length. Two types of length functions, namely the linear length function (4.1) and the non-linear length function (4.3), have already been examined. Here a third semi-linear length function is introduced:

$$l(P) = \begin{cases} \sum_{i=1}^{m} d_i w_i(P), & \text{if } w_i(P) \leq L_i \text{ for } i = 1, ..., m \\ \infty, & \text{else} \end{cases} \tag{4.4}$$

This semi-linear length function can be considered a combination of the functions

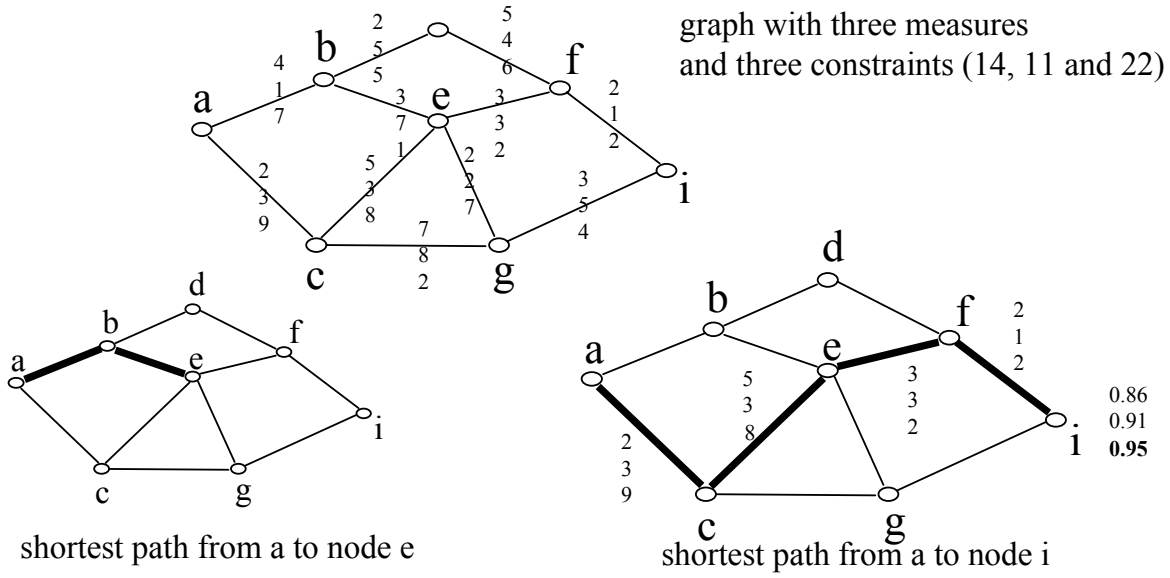


Figure 4.4: Illustration of the property that, when using a non-linear path length definition, subsections of shortest paths are not necessarily shortest paths. Indeed, the length $l(a-b-e) = \max\left(\frac{4+3}{14}, \frac{1+7}{11}, \frac{7+1}{22}\right) \simeq 0.727$ is smaller than $l(a-c-e) = \max\left(\frac{2+5}{14}, \frac{3+3}{11}, \frac{9+8}{22}\right) \simeq 0.772$, although $a-c-e$ is a subsection of the shortest path.

(4.1) and (4.3). Strictly speaking, length function (4.4) is non-linear, because it uses a linear function only inside the constraints surface. These constraints are confining and make length function (4.4) non-linear and thus theorem 25 applies. Nonetheless, the distinction between semi-linear and non-linear is intentionally maintained, because in some instances (see Section 3.2.3) they can lead to different properties.

Two possible length functions (besides function (4.3)) for some specific problems encountered in QoS routing are also presented. The two problems considered are the Restricted Shortest Path (RSP) problem and the Hop-Constrained Maximum Bandwidth (HCMB) problem.

RSP length function:

Given a graph $G = (V, E)$, where each link is characterized by a delay and (monetary) cost, a source node s and a destination node t. Given a delay-constraint D, the problem is to find a path P within the delay-constraint for which the cost is minimum.

A suitable length function for this problem is:

$$l(P) = \begin{cases} c(P), & \text{if } d(P) \leq D \\ \infty, & \text{else} \end{cases}$$

where $c(P)$ is the cost of path P and $d(P)$ is the delay of P. The length function only optimizes for one measure, the cost. This is an example of a semi-linear length function. Therefore, again, the subsections of shortest paths are not necessarily shortest paths. Guo and Matta [66] have used a similar approach to solve the RSP problem.

Hop-Constrained Maximum-Bandwidth (HCMB) problem:

Given a graph $G = (V, E)$, where each link has a specified capacity (bandwidth). Given a source s and a destination t, find a path P with no more than H hops that has maximum capacity.

A possible length function for this problem is:

$$l(P) = \begin{cases} \frac{1}{BW(P)}, & \text{if } h(P) \leq H \\ \infty, & \text{else} \end{cases}$$

where $BW(P)$ is the minimum available bandwidth of path P and $h(P)$ is the number of hops taken by P. The length function is very similar to that for RSP, but is given to illustrate that min/max parameters can also be incorporated. A discussion of the HCMB problem can be found in ([7], [63]).

4.1.2 Visualization of the search space

Figure 4.1 has clarified the linear length search in a two-dimensional space. Each dot in that space represented a path and the weights of that path. The set of paths (dots), from which the solution must be retrieved, is called the search space. The set of dots in Figure 4.1 has been manually constructed. In this section the real search space in

two different classes of graphs is visualized for the ($m = 2$) two-dimensional case. The class of random graphs and the class of square lattices are examined, both with $N = 49$ nodes and uniformly distributed link weights. Figures 4.5 and 4.6 give the search spaces as they change with the level of correlation (ρ) between the two weights per link. Only the first 100 shortest paths according to path length (4.1), with $d_i = 1$ for $i = 1, ..., m$ are plotted.

Because length (4.1) is used, the first 100 paths of the search space do not extend beyond a certain linear equilength line and therefore seem flattened. If all paths in the search space would have been plotted, this would have been avoided, but this is computationally not tractable. All search spaces are depicted on the same scales. Immediate from the plots is that the weights of paths in random graphs are shorter than in the lattices. This is caused by a higher expected hop count in the class of square lattices. Also the shortest paths in the class of two-dimensional lattices are likely to be minimum-hop paths, contrary to the class of random graphs. This is observed for $\rho = -1$, where all the 100 paths lie on the equilength line with length $w_1(P) + w_2(P) = w_1(P) + h - w_1(P) = h$. For the random graphs, the 100 paths are scattered over different hop count paths, as illustrated by the different lines. Besides the two above-mentioned differences, the "clouds" of dots from the random graphs and lattices are quite similar, indicating that the link weight distribution is the most formative factor of the search space.

4.2 The k-shortest path algorithm

The previous section has argued that a linear length cannot guarantee exactness in multiple dimensions. We must therefore resort to non-linear length functions. A corollary of theorem 25 is that the use of non-linear functions requires not only shortest subpaths to be stored, but also non-shortest subpaths. This can be accomplished via a k-shortest path algorithm (Chong *et al.*, [32]). A k-shortest path algorithm is similar to Dijkstra's algorithm. Instead of storing at each intermediate node only the previous hop and the length of the shortest path from the source to that intermediate node, we can store the shortest, the second shortest, the third shortest, ..., up to the k-shortest path together with the corresponding length. It is possible to store less than k paths at a node, but not more. There exist many k-shortest path algorithms. The paper of Eppstein [42] lists many of the references to these algorithms.

In case the value of k is not restricted, the k-shortest path algorithm returns all possible paths ordered in length between source and destination (the search space in the previous section was obtained in this way). The value of k can be limited, in which case there always is a possibility that the multi-constrained path cannot be found. If k is not restricted, exactness is guaranteed.

The fact that k is not restricted in an exact algorithm, implying that all possible

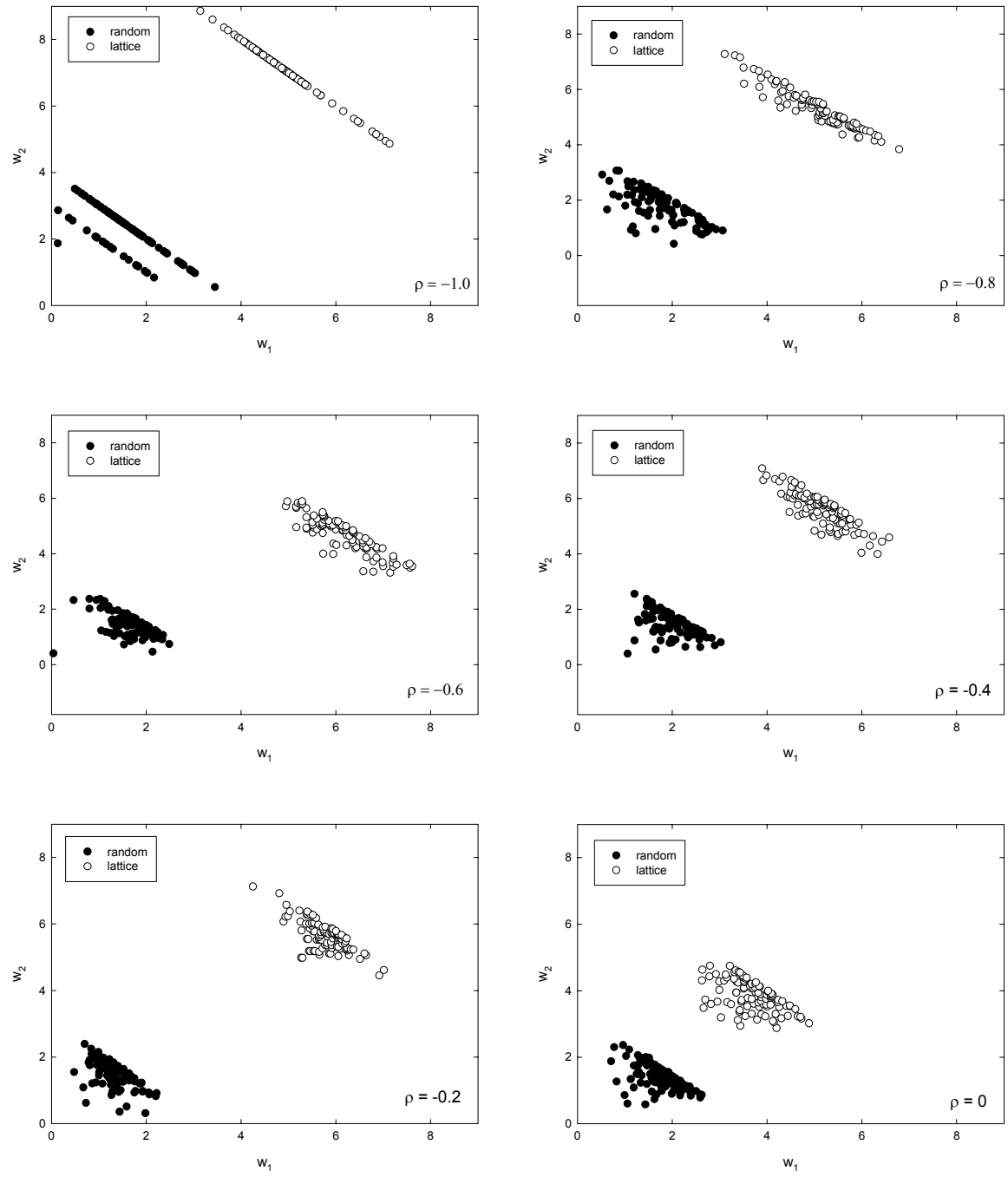


Figure 4.5: Search spaces for the class of random graphs and the class of lattices, for correlation coefficients -1, -0.8, -0.6, -0.4, -0.2 and 0.

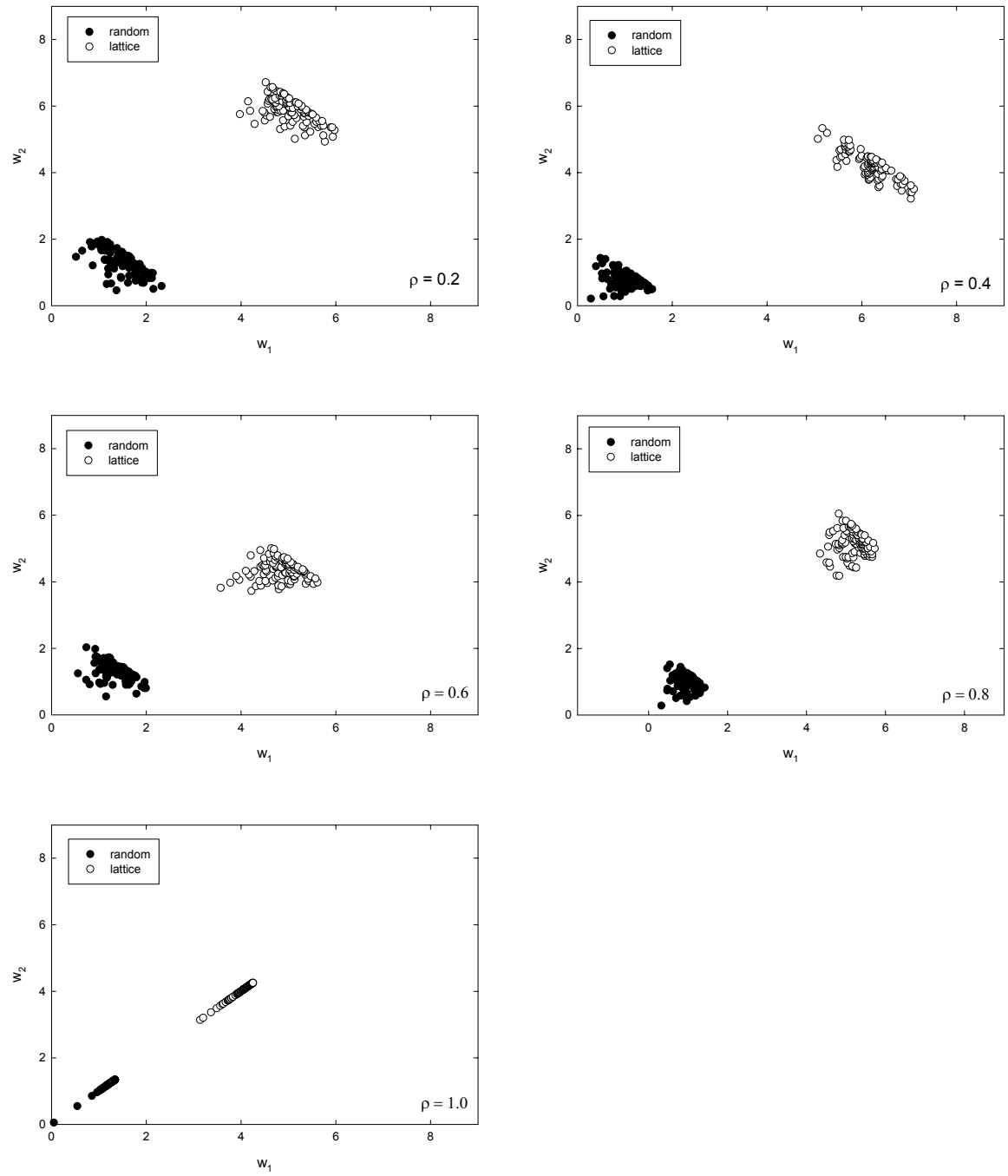


Figure 4.6: Search spaces for the class of random graphs and the class of lattices, for correlation coefficients 0.2, 0.4, 0.6, 0.8 and 1.

paths between source and destination may need to be computed, gives rise to the alluded NP-complete character of the MCP problem. In [162], Van Mieghem has shown that the number of all possible paths between source and destination is less than or equal to $\lfloor e(N-2)! \rfloor$, where $e \simeq 2.718$. This bound scales in the number of nodes N in a non-polynomial fashion. Thus, the maximum value $k_{\max} \leq \lfloor e(N-2)! \rfloor$, for any graph. Bounds for the minimum value $k_{\min}$ needed to find the exact path are difficult to obtain a priori.

4.3 Dominated paths

If the value of k in a k-shortest path based algorithm is unrestricted, then it is necessary to reduce the search space to increase the computational efficiency. One such search-space-reducing technique is that of non-dominance.

4.3.1 Definition of non-dominance

Confining to $m = 2$ dimensions, consider two paths P_1 and P_2 from a source to some intermediate node, each with path weight vector $(w_1(P_1), w_2(P_1)) = (x_1, y_1)$ and $(w_1(P_2), w_2(P_2)) = (x_2, y_2)$, respectively. Figure 4.7 represents two possible scenarios for these two paths.

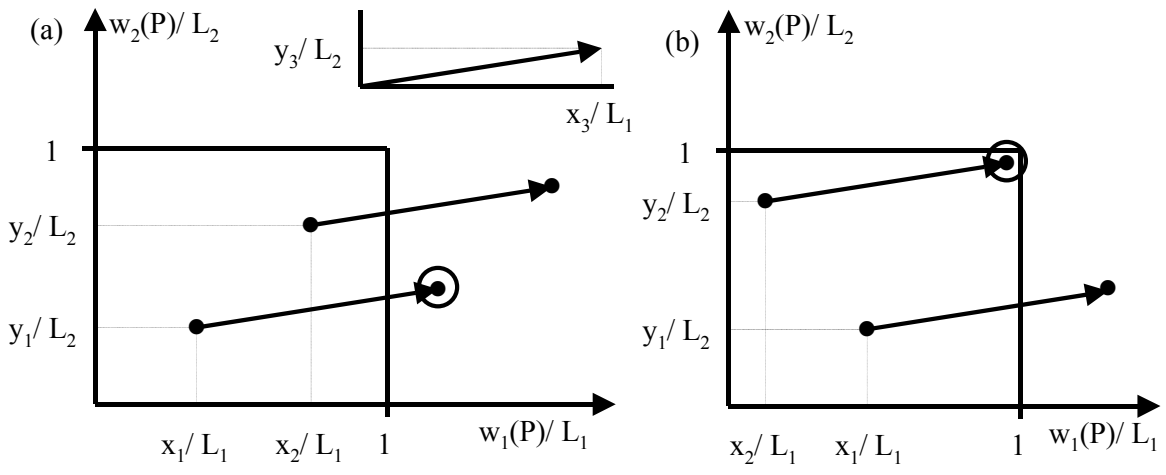


Figure 4.7: Dominated paths: in scenario (a), P_1 dominates P_2, but in scenario (b) neither P_1 nor P_2 is dominant. The shortest path is encircled.

In scenario (a), P_1 is shorter than P_2 and $w_i(P_1) < w_i(P_2)$ for all $1 \leq i \leq m$ components. In that case, any path from the source to the final destination node that

uses P_1 will be shorter than any other path from this source to that destination that makes use of P_2. Indeed, if, for all i, $w_i(P_1) \leq w_i(P_2)$, then $w_i(P_1) + u_i \leq w_i(P_2) + u_i$ for any u_i. For all definitions of length $l(.)$ satisfying the vector norm criteria (such as (4.3)) then holds: $l(\vec{w}(P_1) + \vec{u}) \leq l(\vec{w}(P_2) + \vec{u})$, for any vector $\vec{u}$. Hence, certainly P_2 will never be a subpath of a shortest path and therefore P_2 should not be examined further. Using the terminology of Henig [77], P_2 is said to be dominated by P_1 if, for all i, $w_i(P_1) \leq w_i(P_2)$.

In scenario (b), both paths have crossing abscissa and ordinate points: $w_i(P_1) < w_i(P_2)$ for some indices i, but $w_j(P_1) > w_j(P_2)$ for at least one index j. In such scenario, the shortest path (P_1 in Figure 4.7 (b) with definition (4.3)) between the source and some intermediate node is not necessarily part of the shortest path from source to destination. This is demonstrated in Figure 4.7 (b) by adding the path vector (x_3, y_3), which completes the path towards the destination. It illustrates that P_2 (and not the shortest subpath P_1) lies on that shortest path. Hence, if two subpaths have crossing abscissa-ordinate values, all m components of both paths must be stored in the queue. Alternatively, two paths are non-dominated if $\vec{w}(P_1) - \vec{w}(P_2)$ is not a suitable link weight vector (or path vector), because at least one of its components is negative. Formally defined:

Definition 26 *A path P is called* non-dominated *if there*[1] *does not exist a path P', for which $w_i(P') \leq w_i(P)$ for all link weight components i except for at least one j for which $w_j(P') < w_j(P)$.*

Theorem 27 *If for all m QoS measures, there is no negative cycle in the graph G, then a walk containing a loop is always dominated by the same walk (path) without the loop.*

Proof. If no negative cycles appear in G, then traversing a cycle (loop) Q will never decrease any weights and therefore walk P will always dominate walk $P + Q$, since $w_i(P) \leq w_i(P + Q) = w_i(P) + w_i(Q)$, for all $i = 1, ..., m$. ■

Definition 28 *The Pareto set is defined as the set of all non-dominated paths between s and t.*

The non-dominated weight vectors can be further classified into two groups, namely supported and unsupported. Unsupported non-dominated vectors are dominated by a convex combination of other non-dominated vectors (see Figure 4.8). Although the

[1] If there are two or more different paths between the same pair of nodes that have an identical weight vector, only one of these paths suffices. In the sequel we will therefore assume one path out of the set of equal weight vector paths as being non-dominated and regard the others as dominated paths.

distinction between supported and unsupported vectors may be very useful to many multiple criteria decision making problems, it cannot be applied to constraint-based routing without scarifying exactness, because omitting an unsupported path from the search space may result in omitting the only feasible path.

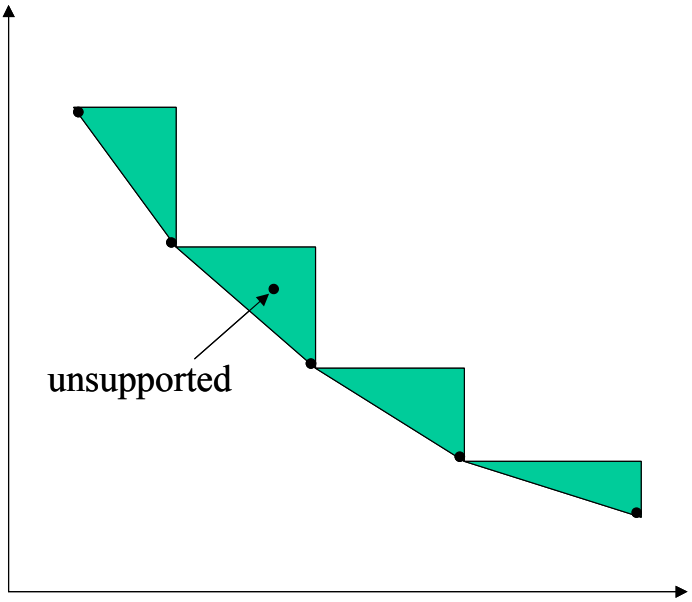


Figure 4.8: An unsupported node/path.

By making the non-dominance property more strict, it is possible to device ϵ-approximation schemes. We call this restricted version of dominance ε-dominance.

Definition 29 *A set of non-dominated paths is ε-non-dominated if the "distance" between the non-dominated paths is $\geq \varepsilon$, where the distance between two paths P_1 and P_2 is defined as* $\max_{i=1,\ldots,m}(|w_i(P_1) - w_i(P_2)|)$.

Therefore, if a path Q is dominated by a path P, then Q is also ε-dominated by P, but not necessarily vice versa. Figure 4.9 illustrates the concept of ε-dominance, where the squares around the points (paths) have size 2ε.

4.3.2 An attainable bound for k_{max}

The worst-case amount of non-dominated paths is determined by the granularity of the constraints. In reality most protocols will only allocate a fixed, positive number of bits per measure. In that case, the constraints L_i can be expressed as an integer number of a basic measure unit. For example, the delay component can be expressed in units of

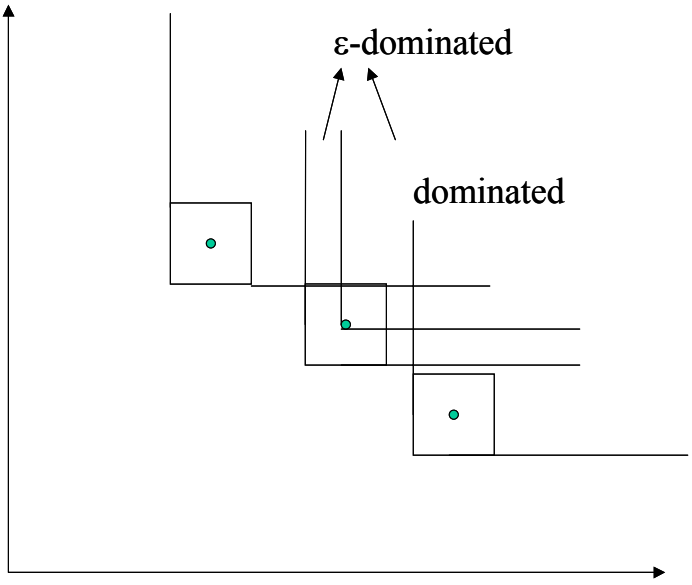


Figure 4.9: Visualization of ε-dominance.

ms. The worst-case number of partial paths that have to be maintained in parallel in each node is $\min(L_1, L_2)$ for $m = 2$ as shown in Figure 4.10.

Since the concept of path dominance reduces the m-dimensional search space, the worst-case number of partial paths is

$$k_{\max} = \min\left[\frac{\prod_{i=1}^{m} L_i}{\max_{1\leq i\leq m} L_i}, \lfloor e(N-2)! \rfloor\right] \tag{4.5}$$

where the second argument of the min-operator denotes the maximum number of paths that exists between two nodes in any graph (see [162]). This argument applies in case the granularity is infinitely small or, equivalently, for real values of w_i. The first argument applies in case of a finite granularity, as shown below:

Theorem 30 *If all weight components have a finite granularity, the number of non-dominated paths within the constraints cannot exceed* $\frac{\prod_{i=1}^{m} L_i}{\max_{1\leq i\leq m} L_i}$.

Proof. Without loss of generality, assume that $L_1 \leq L_2 \leq ... \leq L_m$, such that $\frac{\prod_{i=1}^{m} L_i}{\max_{1\leq i\leq m} L_i}$ reduces to $\prod_{i=1}^{m-1} L_i$. First, if $m = 1$, there is only one shortest and non-dominated path possible within the constraint L_1. This case reduces to single parameter shortest path routing. For $m \geq 2$, the maximum number of distinct paths[2] is $\prod_{i=1}^{m} L_i$.

[2]Two paths are called distinct if their path weight vectors are not identical.

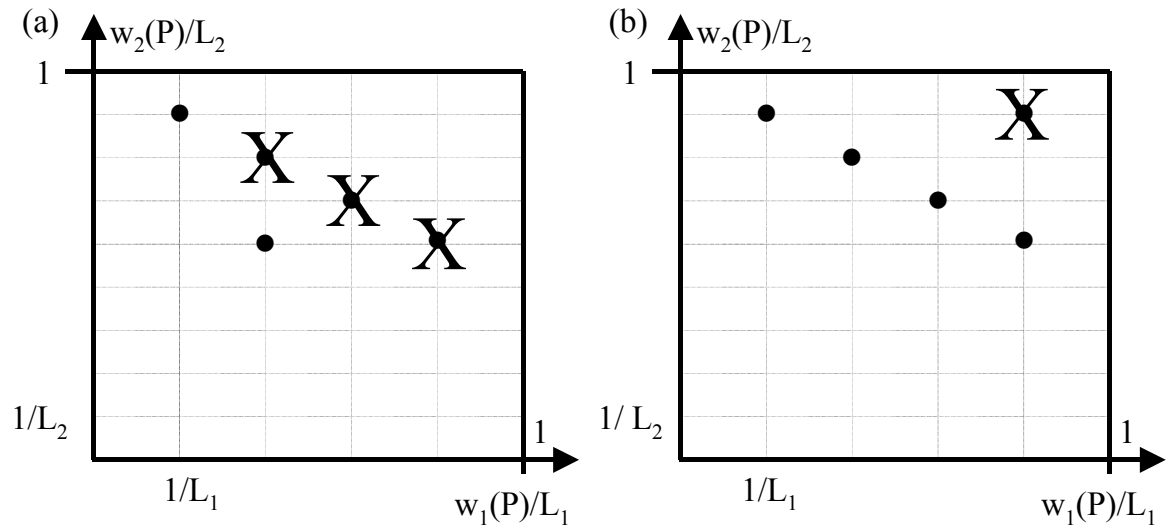


Figure 4.10: (a) only two partial paths should be maintained in parallel; (b) any path dominated by all should be discarded. The 'X' refers to dominated paths.

Two paths P_1 and P_2 do not dominate each other if, for at least two different link weight components $1 \leq a \neq b \leq m$ holds that $w_a(P_1) < w_a(P_2)$ and $w_b(P_1) > w_b(P_2)$. This definition implies that, for any couple of non-dominated paths P_1 and P_2, at least two components of the m-dimensional vector $\vec{w}(P_1)$ must be different from $\vec{w}(P_2)$. Equivalently, if we consider an $(m-1)$-dimensional subvector $\vec{v}$ by discarding the j-th component in $\vec{w}$, at least one component of $\vec{v}(P_1)$ must differ from $\vec{v}(P_2)$. The maximum number of different subvectors $\vec{v}$ equals $\prod_{i=1;i\neq j}^{m} L_i$. If $j \neq m$ such that $L_j < L_m$, within the $\prod_{i=1;i\neq j}^{m} L_i$ possibilities, there are paths for which only the j-th and/or m-th component differ, while all the other components are equal. In order for these paths not to dominate each other, the j-th and m-th component must satisfy the condition that if $w_m(P_1) > w_m(P_2)$, then $w_j(P_1) < w_j(P_2)$ or vice versa. For the m-th component, there are L_m different paths for which $w_m(P_1) = L_m > w_m(P_2) = L_m - 1 > ... > w_m(P_{L_m}) = 1$. Since $L_j < L_m$, there are only L_j paths for which $w_j(P_1) = 1 < w_j(P_2) = 2 < ... < w_j(P_{L_j}) = L_j$. Therefore, there are paths P_1 and P_2 for which $w_m(P_1) > w_m(P_2)$, but $w_j(P_1) = w_j(P_2)$. Hence, only L_j instead of L_m non-dominated paths are possible, leading to a total of $\frac{L_j}{L_m}\prod_{i=1;i\neq j}^{m} L_i = \prod_{i=1}^{m-1} L_i$ non-dominated paths. This proofs the upper bound $k_{\max} = \prod_{i=1}^{m-1} L_i$. ■

Corollary 31 *The first bound* $\frac{\prod_{i=1}^{m} L_i}{\max_{1\leq i\leq m} L_i}$ *in (4.5) on the number of non-dominated paths within the constraints can be attained.*

Proof. Without loss of generality assume that $L_1 \leq L_2 \leq ... \leq L_m$. We will show that there exist sequences $\{L_1, L_2, ..., L_m\}$ for which the bound $\frac{\prod_{i=1}^{m} L_i}{\max_{1\leq i\leq m} L_i}$ is achieved. If for each pair of paths P_i, P_j the m-th link weight component obeys

$$w_m(P_i) \geq w_m(P_j) + \sum_{k=1}^{m-1} (w_k(P_j) - w_k(P_i)) \tag{4.6}$$

then $\frac{\prod_{i=1}^{m} L_i}{\max_{1\leq i\leq m} L_i}$ is a strict, attainable bound. Formula (4.6) is found by recursively applying the following prerequisite, recalling that the smallest difference between two weight components is one unit: if for two paths P_1, P_2 applies that $w_j(P_1) - w_j(P_2) = 1$ (in units of the j-th weight component) for only one j of the first $m-1$ measures and $w_i(P_1) - w_i(P_2) = 0$ for the other $1 \leq i \neq j \leq m-1$, then for non-dominance to apply, the m-th weight components must satisfy $w_m(P_1) - w_m(P_2) \leq -1$. If (4.6) is not obeyed, then $w_m(P_1) > w_m(P_2) - 1$, i.e. $w_i(P_1) \geq w_i(P_2)$ for $i = 1, ..., m$ and according to the definition of non-dominance P_1 is then dominated by P_2. The largest possible difference between two path vectors provides us with a lower bound on L_m,

$$L_m \geq 1 + \sum_{i=1}^{m-1} (L_i - 1)$$

When this bound is not satisfied, then the number of non-dominated paths within the constraints is smaller than $\frac{\prod_{i=1}^{m} L_i}{\max_{1\leq i\leq m} L_i}$. ■

For example, in $m = 5$ dimensions, with $L_1 = 1,\ L_2 = 2,\ L_3 = 3,\ L_4 = 3,\ L_5 = 6$ ($\geq 1 + 1 + 2 + 2$), all $\frac{\prod_{i=1}^{m} L_i}{\max_{1\leq i\leq m} L_i} = 18$ non-dominated vectors are

$$\left|\begin{array}{c||c||c||c||c||c||c||c||c||c||c||c||c||c||c||c||c||c}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\
1 & 1 & 1 & 2 & 2 & 2 & 3 & 3 & 3 & 1 & 1 & 1 & 2 & 2 & 2 & 3 & 3 & 3 \\
1 & 2 & 3 & 1 & 2 & 3 & 1 & 2 & 3 & 1 & 2 & 3 & 1 & 2 & 3 & 1 & 2 & 3 \\
6 & 5 & 4 & 5 & 4 & 3 & 4 & 3 & 2 & 5 & 4 & 3 & 4 & 3 & 2 & 3 & 2 & 1
\end{array}\right|$$

Although there exist many problems that are NP-complete, the average-case complexity might not be intractable, suggesting that such an algorithm could have a good performance in practice. The theory of average-case complexity was first advocated by Levin [106]. Below, a lemma is given that suggests that *the average and even amortized* [3] *complexity of solving the MCP problem is polynomial in time* for fixed m and all weights w_i independent random variables.

[3] Amortized analysis differs from average-case analysis in that probability is not involved; an amortized analysis guarantees the average performance of each operation in the worst-case [34].

Lemma 32 The expected number of non-dominated vectors in a set of T i.i.d. vectors in m dimensions is upper bounded by $(\ln T)^{m-1}$.

A proof of Lemma 32 can be found by adopting a similar approach as presented by Barndorff-Nielsen and Sobel [13], or by Bentley *et al.* [18].

To gain some insight into the number of non-dominated paths in a graph, assume that the path-vectors are i.i.d. vectors. Then, in the worst-case, there exist $T = \lfloor e(N-2)! \rfloor$ paths, leading us to an expected number of non-dominated paths between the source and destination in the worst-case of

$$(\ln T)^{m-1} = (\ln(\lfloor e(N-2)! \rfloor))^{m-1} \leq (1 + (N-2)\ln(N-2))^{m-1}$$

Hence, the amortized complexity is polynomial in N for fixed m.

In the limit $m \to \infty$ and for w_j independent random variables, all paths in $G = (V, E)$ are non-dominated, which leads to the following lemma.

Lemma 33 If the m components of the link weight vectors are independent random variables and the constraints L_j are such that $0 \leq \frac{w_j}{L_j} \leq 1$, then any path with H hops has precisely a length (as defined by (4.3)) equal to H in the limit $m \to \infty$.

Proof. Consider a path P from source to destination with H hops. Then $\Pr[w_j(P)/L_j > H] = 0$. Moreover, with length function (4.3), $\Pr[l(P) > H] = 0$. Since it is assumed that the link weights are independent, we have $\Pr[l(P) \leq H - \varepsilon] = \prod_{j=1}^{m} \Pr[w_j(P)/L_j \leq H - \varepsilon]$. But, for any real $\varepsilon > 0$ and each j, $\Pr[w_j(P)/L_j \leq H - \varepsilon] < 1$, $\Pr[l(P) \leq H - \varepsilon] = 0$ for $m \to \infty$. Hence, in that limit, each path with H hops has length precisely equal to H. ■

This means that for $m \to \infty$ it suffices to calculate the minimum-hop path, irrespective of the link weight distribution of the m independent components. Since the minimum-hop problem is an instance of a single measure shortest path problem, it has polynomial complexity. Of course, this limit case $m \to \infty$ mainly has theoretically value. In realistic QoS routing, only a few link weights are expected to occur. The number m of QoS link weights is a design choice for the QoS routing protocol.

4.4 Look-ahead

4.4.1 The look-ahead concept

Besides path dominance, the look-ahead concept can be viewed as an additional[4] mechanism to reduce the search space of possible paths. The idea, first introduced in the

[4]There may exist more search-space-reducting methods. The use or even existence of other search-space-reducting methods may rely on the specifics of the topology and link weight structure.

field of Artificial Intelligence and named A* (see Chapter 3), is to further limit the set of possible paths by using information of the remaining subpath towards the destination. The look-ahead concept proposes to compute the shortest path tree rooted at the destination to each node $n \in V$ in the graph G for each of the m link weights separately. Hence, for each link weight component $1 \leq i \leq m$, the lowest value from the destination to a node $n \in V$ is stored in the queue of that node n. In total, a one-dimensional shortest path algorithm is executed m times resulting in $N-1$ vectors with shortest values for each link weight component from a node n to the destination t. The basic importance of look-ahead is to provide each node n with an exact, attainable lower bound of $w_i(P_{n\to t})$, for each individual link weight component i. We denote by $P^*_{n\to t;i}$ the shortest path in the link weight component i from node n to the destination t. Since a one-dimensional shortest path algorithm is executed m times for each link weight separately, the shortest path $P^*_{n\to t;i}$ is likely different from $P^*_{n\to t;j}$ for different link weight components $i \neq j$. Let us denote the vector with these lower bounds by $\vec{b}(n)$, with $b_i(n) = w_i\left(P^*_{n\to t;i}\right)$.

Why are these exact, attainable lower bounds $b_i(n)$ so useful? First, at any intermediate node n and for each suitable path $P_{s\to n}$, the inequality

$$w_i(P_{s\to n}) + b_i(n) \leq L_i \tag{4.7}$$

should be satisfied for all measures $i = 1, ..., m$. Indeed, if the sum of the link weight component, of a subpath $P_{s\to n}$ from the source s to the intermediate node n, and the lowest possible value $b_i(n) = w_i\left(P^*_{n\to t;i}\right)$, of the shortest remaining subpath $P^*_{n\to t;i}$ from that intermediate node n to the destination t, exceeds the constraint L_i, then subpath $P_{s\to n}$ can never be complemented with a path $P_{n\to t}$ to satisfy the constraint L_i. Hence, the subpath $P_{s\to n}$ that violated one of the inequalities in (4.7) should not be considered further as a possible candidate of the multi-constrained routing problem. The check of compliance to the inequalities (4.7) can reduce the number of paths in the search space of possible paths.

A second improvement is based on the knowledge of the m one-dimensional shortest paths $P^*_{t\to s;i}$. If the length $l\left(P^*_{t\to s;i}\right) = l\left(P^*_{s\to t;i}\right) < 1$, this means that the inverse path $P^*_{s\to t;i}$ that minimizes the sum of the i-th link weight component possesses an m-dimensional length smaller than 1 and thus meets all the constraints. This implies that there already exists a path from source s to destination t with length shorter than 1 and that the overall m-dimensional shortest path must at least be smaller or equal in length than this path $P^*_{s\to t;i}$.

A third possible improvement of look-ahead is to store in each queue $l(P_{s\to n} + P^*_{n\to t;i})$ instead of $l(P_{s\to n})$. Thus, the length of the sum of the vector $\vec{w}(P_{s\to n})$ and the lower-bound vector $\vec{b}(n)$ is the comparison metric stored in the queue at each node. This change favors the paths with lowest "predicted" end-to-end length rather than the path with the so far lowest length. Observe that in the end at the destination queue,

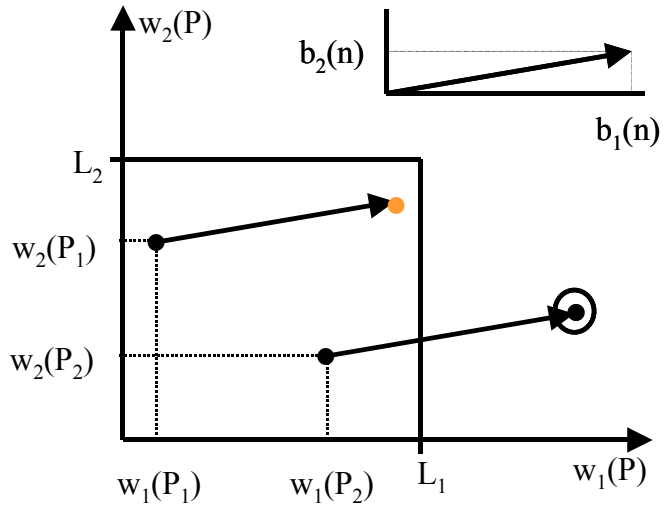


Figure 4.11: The look-ahead constraints check in two dimensions $m = 2$: the addition of the intermediate path's link weight vector $\vec{w}(\mathcal{P})$ and the lowest possible remaining link weight vector $\vec{b}(n)$ must lie within the constrained region.

the stored "predicted" lengths are the same as the actual lengths. Simulations in [167] show that the look-ahead technique indeed reduces the complexity of solving MCP.

4.4.2 Complexity of look-ahead

The complexity of the three look-ahead improvements consists of the sum of (a) m times the complexity of Dijkstra's (or another shortest path algorithm) $O(mN \log N + mM)$ and (b) m times the computation of the length of a path, which is at most $O(m^2 N)$. Hence, only for a sufficiently large number of nodes N, the look-ahead concept is expected to improve the performance, mainly by limiting the search space of possible paths. Since the search space of possible paths can grow as a factorial, i.e. $O((N-2)!)$ for large N, this suggests that the improvements will pay off the small increase in complexity.

4.4.3 Other look-ahead applications

Instead of employing a one-dimensional shortest path algorithm per individual link weight component, other polynomial-time multiple-parameter routing algorithms (e.g., TAMCRA [38] with small k, or Jaffe's linear length algorithm [85]) could be used to determine the end-to-end predictions. In that case, again the paths from the destination t to all other nodes are computed. It is possible to store the m weights of these

paths at their corresponding node, or each node n in the graph only receives one path length $l(P_{t\to n})$ corresponding to the length used in the main multiple-parameter routing algorithm.

For any non-linear length holds that $l(P_{s\to t}) \leq l(P_{s\to n}) + l(P_{n\to t})$. For length (4.3), the constraints require that $l(P_{s\to t}) \leq 1$, such that the look-ahead tests (4.7) could be replaced by the possibly too stringent[5] test $l(P_{s\to n}) \leq 1 - l(P_{n\to t})$. Since polynomial-time heuristics are used, the lengths $l(P_{t\to n})$ are not necessarily the smallest possible. Hence, SAMCRA, for example equipped with TAMCRA as look-ahead, cannot exactly solve the MCOP problem: it is only exact for the less restrictive MCP, because the end-to-end path length $l(P_{s\to t})$ is exact and it is verified to be feasible. However, since non-linear length algorithms are likely to outperform linear length algorithms, TAMCRA may lead SAMCRA's search sooner into the correct direction. This is indeed observed in [102].

In case of a linear length as in Jaffe's algorithm with length (4.1), the lengths $l(P_{B\to n}) = l(P^*_{n\to B})$ are shortest and, hence, they can serve as single lower bounds $b(n)$, provided the MCP algorithm uses the same (semi) linear length within the constraints, e.g. length function (4.4). Clearly, the advantage of a routing algorithm with a linear length is that an attainable lower bound can be obtained.

4.5 Bi-directional search in multiple dimensions

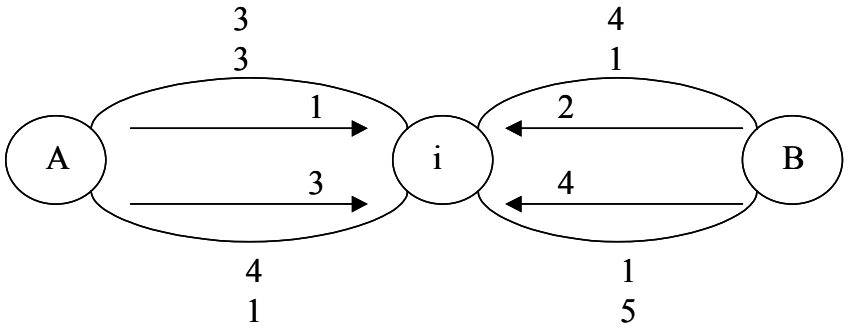


Figure 4.12: Example of bi-directional search in multiple dimensions.

In Section 3.2.3, the concept of bi-directional search in one dimension was described. This section analyzes whether the concept of bi-directional search can be extended from $m = 1$ to $m > 1$ dimensions. The complicating factor is the non-linear length (4.3),

[5] If bounds larger than the lower bounds $\vec{b}$ are used, the search space reduction may be too stringent, which may exclude finding a feasible solution.

which causes that subsections of shortest paths in $m > 1$ dimensions are not necessarily shortest paths themselves. If two shortest paths (one originating at the source node and the other at the destination node) in $m > 1$ dimensions meet at an intermediate node, the resulting complete path is not necessarily the shortest path from source to destination. We must keep track of the shortest length of the complete paths found so far. Even if a new complete path exceeds the length of a previously found complete path, that new path cannot be discarded as illustrated in Figure 4.12. In this figure, the links represent paths, with their corresponding path weight vector. The arrows indicate the order of arrival of these subpaths at node i. Once the first two subpaths have arrived at node i, the first complete path is obtained with weight vector (7,4). If the constraints are (10,10), then the length of this path equals 0.7. Once the third path arrives at node i, it makes a complete path with the second path, with total length 0.8. However, this subpath cannot be removed, because combined with path 4, it forms the shortest path with link weight vector (5,6) and length 0.6. This example also illustrates that, at some intermediate nodes, multiple paths have to connect with each other.

The problems in multiple dimensions complicate the recognition of the true shortest path. In other words, an efficient stop criterion is absent and therefore the search for paths must be continued until the queue is empty. The bi-directional search in $m > 1$ dimensions has more potential for the MCP problem, where the routing algorithm can stop as soon as a complete path obeys the constraints. Bi-directional search has also more potential for QoS algorithms that use a (semi) linear length function. If the semi-linear length (4.4) is used, then if two paths P_1 (originating from the source) and P_2 (originating from the destination) meet at a node, the same approach as in Section 3.2.3 can be adopted, provided that the complete path obeys the constraints. In other words, the first complete path that is extracted and that obeys *minlength* (and the constraints) is also the shortest path according to (4.4) within the constraints. For bi-directional search in $m > 1$ dimensions, a semi-linear length function is more convenient to handle than a fully non-linear length function. In [102], we have proposed HAMCRA, a bi-directional variant of SAMCRA, which solves the MCP problem exact by using SAMCRA in one direction and TAMCRA in the other.

4.6 The SAMCRA algorithm

The previous sections have listed four concepts for an exact and efficient QoS routing algorithm. All four concepts are present in SAMCRA. This section is devoted to the detailed presentation of the SAMCRA algorithm. In the meta-code, some functions (INSERT, EXTRACT-MIN, DECREASE-KEY) are borrowed from Cormen *et al.* [34].

```
INITIALIZE(G, m, s, t)
1.  for each v ∈ V
2.      counter[v] ← 0
3.  maxlength ← 1.0
4.  for i = 1, ..., m
5.      DIJKSTRA(G, s, t, i) → b_i(n), P*_{s→t;i}
6.      if l(P*_{s→t;i}) < maxlength
7.          maxlength ← l(P*_{s→t;i})
8.  queue Q ← ∅
9.  counter[s] ← counter[s] +1
10. INSERT(Q, s, counter[s], NIL, l(b(s)))
```

Figure 4.13: Meta-code Initialization phase.

4.6.1 Meta-code SAMCRA

The subroutine INITIALIZE (see Figure 4.13) initializes the necessary parameters for the main algorithm and computes the look-ahead information. Lines 1 and 2 set the number of stored paths (counter[]) at each node to zero. *maxlength* refers to the maximum length that a (sub)path may have. Paths with length $l(P) > maxlength$ can be discarded, because they either violate the constraints or are larger than an already found end-to-end path. *maxlength* is initially set to 1.0 in line 3, corresponding to the constraint values. The look-ahead lower bounds $\vec{b}$ are calculated in line 5 with the function DIJKSTRA(G, s, t, i). This function finds for each individual QoS measure i the lower bounds $b_i(n)$ from any node $n \in V$ to the destination node t. An efficient way to accomplish this is to compute, for each measure i, a shortest path tree with the Dijkstra algorithm from the destination t to all other nodes. Also, for each measure i, the shortest paths from s to t are stored. For each of these m shortest paths, line 6 computes the length (4.3) and checks whether one of them has a lower length than *maxlength*. If this happens, in line 7, *maxlength* is updated with the new lower value, because if we already have a path with length < 1.0, it is pointless to evaluate paths with larger length. SAMCRA starts with the source node s, which is inserted into the queue (line 10).

The subroutine FEASIBILITY (see Figure 4.14) checks whether paths dominate each other or violate the *maxlength* value. A (sub)path $u[i]$ refers to the i-th path that is stored at node u. The vector $\vec{d}(u[i])$ represent a subpath weight vector $\vec{w}\,(P_{s\to u})$. FEASIBILITY extends the i-th path at node u towards the neighboring node v, where already counter[v] nodes are stored. The weight vector of this extended path equals $\vec{d}[u[i]] + \vec{w}(u, v)$. For each of the counter[v] subpaths $v[j]$ stored at node v (lines 2-3),

```
FEASIBILITY(G, u, i, v,counter,d, w,maxlength)
1. dominated ← 0
2. for j = 1, ...,counter[v]
3.     if (d⃗[u[i]] + w⃗(u, v)) − d⃗[v[j]] ≥ 0⃗
           OR l(d⃗[v[j]]) > maxlength
4.         v[j] ← BLACK
5.     else if d⃗[v[j]] − (d⃗[u[i]] + w⃗(u, v)) ≥ 0⃗
6.         dominated ← 1
7. return dominated
```

Figure 4.14: Meta-code Feasibility.

the path weight vector $\vec{d}[v[j]]$ is subtracted from $\vec{d}[u[i]] + \vec{w}(u, v)$ to verify whether the resulting vector consists of only non-negative components. If all components of the difference vector are non-negative, then the subpath $v[j]$ is dominated by the extended path (see Section 4.3). Line 3 also checks whether the subpath $v[j]$ violates the *maxlength* value. If the subpath $v[j]$ is either dominated or exceeds *maxlength*, it need not be considered anymore and is marked black in line 4. A path marked black has become obsolete and may be replaced by a new path. Line 5 checks whether the extended path itself is dominated by a subpath $v[j]$. If so, it is labelled "dominated." Our final subroutine is called UPDATEQUEUE (see Figure 4.15).

UPDATEQUEUE has the task of updating the queue Q with a new path, namely the extended path from $u[i]$ to node v. Lines 1-2 check if any black paths exist with larger *predicted_length* than the new extended path. If so, it replaces the black path $v[j]$ with the extended path in lines 3-5. Line 3 decreases the *predicted_length* of subpath $v[j]$ with the smaller *predicted_length* from the extended path and updates the path weight vector (line 4) and predecessor list (line 5). If the queue Q is updated through DECREASE-KEY in lines 3-5, the subroutine UPDATEQUEUE stops in line 6 and returns to the main algorithm. However, if lines 1-6 fail and no black paths can be replaced, then the extended path is inserted in the queue (lines 7-10). In this case, not the real length of this subpath is stored, but its *predicted_length*. However, SAMCRA can be used with different length function as discussed in Section 4.1.

The main algorithm (see Figure 4.16) starts with the execution of the subroutine INITIALIZE (line 1). Provided the queue Q is not empty (otherwise no feasible path is present), the EXTRACT-MIN function in line 3 selects the minimum path length in the queue Q and returns $u[i]$, the i-th path $P_{s\to u}$ stored in the queue at node u. With these numbers and the predecessor list π, the entire path can be reconstructed via backtracing. The extracted path is marked grey in line 4. If the node u, corresponding

```
UPDATEQUEUE(Q, u, i, v, j, d, w, π, counter[v],
predicted_length)
1.  for j ← 1 to counter[v]
2.      if v[j] = BLACK AND
            l(d⃗[v[j]] + b⃗[v]) > predicted_length
3.          DECREASE-KEY(Q, v, j,predicted_length)
4.          d⃗[v[j]] ← d⃗[u[i]] + w⃗(u, v)
5.          π[v[j]] ← u[i]
6.          return
7.  counter[v] ← counter[v] +1
8.  INSERT(Q, v,counter[v], predicted_length)
9.  d⃗[v[counter[v]]] ← d⃗[u[i]] + w⃗(u, v)
10. π[v[counter[v]]] ← u[i]
```

Figure 4.15: Meta-code Updatequeue.

to the extracted path $u[i]$, equals the destination t, the shortest path satisfying the constraints is returned. If $u \neq t$, the scanning procedure is initiated in line 8. Line 8 describes how the i-th path up to node u is extended towards its neighboring node v, except for the previous node where it came from. The previous node on the path $u[i]$ is stored in the predecessor list π. Returning to this previous node induces a loop, which must be avoided. Since the link weights are non-negative, paths that have a loop are always dominated by paths without loops. This property relieves us from the task of storing/backtracing the entire path $u[i]$ to avoid loops. Line 9 invokes the FEASIBILITY subroutine to check whether all stored paths at node v are non-dominated and obey *maxlength*. FEASIBILITY also checks whether the new extended path is not dominated by previously stored paths at node v. In line 10, the length of the predicted end-to-end path weight vector (composed of the real subpath weight vector from s to v plus the lower-bound vector from v to t) is calculated. Line 11 tests if the new extended path is non-dominated and has a *predicted_length* $\leq$ *maxlength*. If this is the case it can be stored and the queue must be updated (line 12). Removing paths for which *predicted_length* $>$ *maxlength* is the search space reduction of the look-ahead concept. Finally, *maxlength* can be updated in lines 13-14.

4.6.2 Complexity of SAMCRA

The calculation of the worst-case complexity of SAMCRA as presented above will be computed. First, the worst-case complexity of the subroutines is determined, after which the total worst-case complexity of SAMCRA is constructed.

```
SAMCRA(G, m, s, t, L)
1. INITIALIZE(G, m, s, t) → b⃗
2. while Q ≠ ∅
3. EXTRACT-MIN(Q) → u[i]
4. u[i] ← GREY
5. if u = t
6.     return path
7. else
8.     for each v ∈ adj[u]\{π[u[i]], s}
9.         FEASIBILITY(G, u, i, v, counter, d, w, maxlength)
           → dominated
10.        predicted_length ← l(d⃗[u[i]] + w⃗(u, v) + b⃗[v])
11.        if predicted_length < maxlength
               AND dominated ≠ 1
12.            UPDATEQUEUE(Q, u, i, v, j, d, w, π, counter[v],
               predicted_length)
13.            if v = B AND
                   predicted_length < maxlength
14.                maxlength ← predicted_length
```

Figure 4.16: Meta-code SAMCRA.

The initialization phase has a polynomial-time complexity. Initializing counter takes $mO(N)$ times. Executing heap-optimized Dijkstra (lines 4-5) leads to $mO(N \log N + M)$ and m times computing a length of a path (line 6) leads to $mO(mN)$. The other operations take $O(1)$, leading to a total worst-case complexity of $O(N + mN \log N + mM + m^2N + 1) = O(mN \log N + mM + m^2N)$.

The complexity of the FEASIBILITY subroutine depends on the calculation of length and verification of dominance. Calculating the length (4.3) of a weight vector takes $O(m)$, while verifying path dominance between two paths takes $O(m)$ at most. Since there can be at most $k_{\max}$ paths at a node, the FEASIBILITY subroutine takes at most $O(k_{\max}m)$.

The complexity of the subroutine UPDATEQUEUE depends on the specifics of the heap structure (e.g., Fibonacci or Relaxed heaps). Lines 1 and 2 take at most $O(k_{\max}m)$. The heap functions DECREASE-KEY (line 3) and INSERT (line 8) can be performed in $O(1)$. Updating $\vec{d}$ (lines 4 and 9) takes at most $O(m)$. The total worst-case complexity of UPDATEQUEUE leads to $O(k_{\max}m)$.

The total worst-case complexity of SAMCRA is constructed as follows. The initialization phase adds $O(mN \log N + mM + m^2N)$. The queue Q can never contain

more than $k_{\max}N$ path lengths. When using a Fibonacci or Relaxed heap to structure the queue, selecting the minimum path length among $k_{\max}N$ different path lengths takes at most a calculation time of the order of $O(\log(k_{\max}N))$ [34]. As each node can be selected at most $k_{\max}$ times from the queue, the EXTRACT-MIN function in line 3 takes $O(k_{\max}N\log(k_{\max}N))$ at most. Returning a path in line 6 takes at most $O(N)$. The for-loop starting on line 8 is invoked at most $k_{\max}$ times from each side of each link in the graph, leading to $O(k_{\max}M)$. FEASIBILITY takes $O(k_{\max}m)$. Calculating the length in line 10 takes $O(m)$ and updating the queue takes $O(k_{\max}m)$. Combining all those contributions yields a total worst-case complexity for SAMCRA of $O(mN\log N + mM + m^2N + N + k_{\max}N\log(k_{\max}N) + k_{\max}^2mM)$ or

$$C_{SAMCRA} = O(k_{\max}N\log(kN) + k_{\max}^2mM) \tag{4.8}$$

where m is fixed. When the link weights are real numbers, the granularity is infinitely small implying that the first argument in (4.5) is infinite and, hence, $k_{\max} = O(N!) = O(\exp(N\ln N))$. But, as argued before, in practice these measures will have a finite granularity, such that the link weights w_i are integers (in units specified by the QoS qualifier). Hence, $k_{\max}$ is limited by the first, finite argument in (4.5). This means that, for a fixed number of constraints m and finite granularity in the constraints, SAMCRA has a pseudo-polynomial-time complexity.

For a single constraint ($m = 1$ and $k_{\max} = 1$), SAMCRA's complexity reduces to the complexity of the Dijkstra algorithm: $C_{Dijkstra} = O(N\log N + M)$. By restricting k at the expense of possibly loosing exactness, an optimized version of TAMCRA is obtained. It is also possible to stop SAMCRA, when a feasible path (not necessarily shortest) is found, which significantly reduces the execution time, especially for loose constraints. For the MCP problem, this option is recommended.

4.6.3 Example of the operation of SAMCRA

Consider the topology drawn in the top of Figure 4.17. We are asked to find a path from A to B subject to the constraints vector $\vec{L} = (10, 10)$.

SAMCRA returns the shortest path satisfying the $\vec{L}$-vector in 7 steps (including initialization). Whenever a path is extracted from the queue (line 3 of the meta-code), the corresponding box is colored in grey. The arrows refer to lines 8-12. The algorithm stops when the first entry of the destination node B is extracted from the queue.

In step Init of Figure 4.17, the initialization phase of SAMCRA is displayed. The lower-bound vectors $\vec{b}(n)$ are displayed in boxes beside the nodes. The initialization phase also examines the two shortest Dijkstra paths $P^*_{A\to B;1}$ and $P^*_{A\to B;2}$ from A to B, where $P^*_{A\to B;1} = ACEFB$ with $\vec{w}(P^*_{A\to B;1}) = (4, 12)$ and where $P^*_{A\to B;2} = ADEB$ with $\vec{w}(P^*_{A\to B;2}) = (8, 4)$. The path $P^*_{A\to B;2}$ lies within the constraints, $w_i(P^*_{A\to B;2}) \le L_i$ for

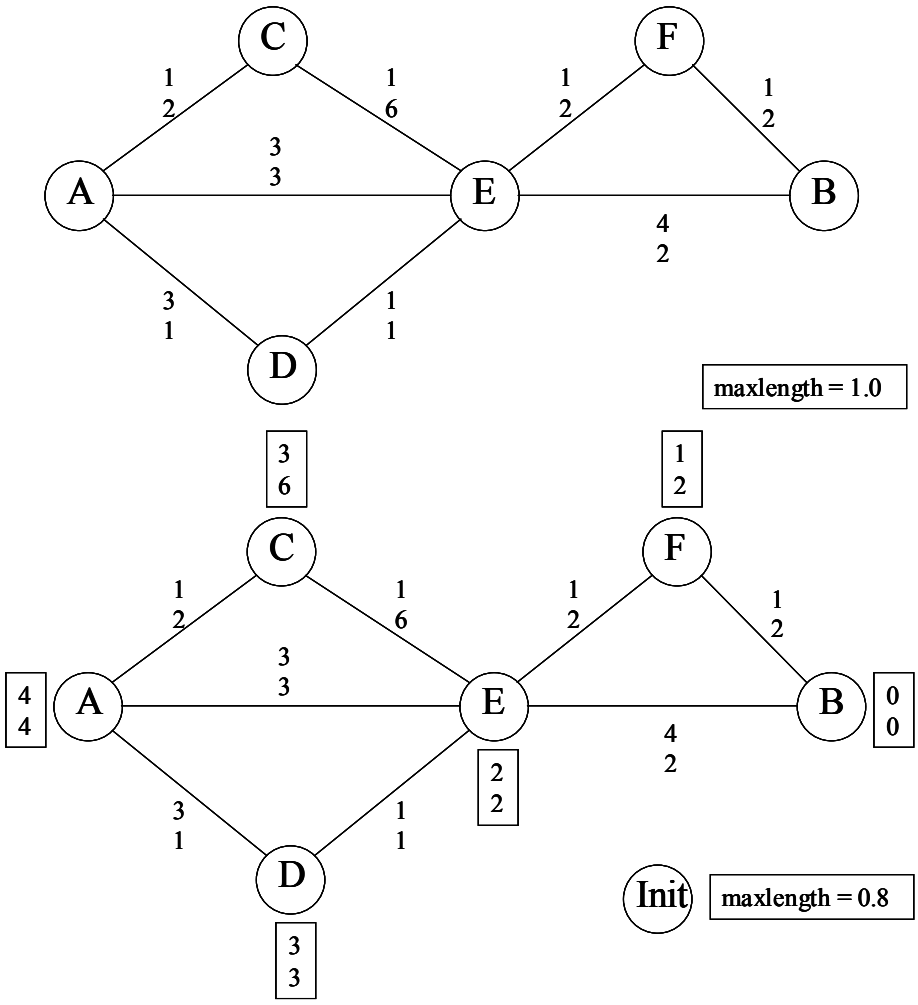


Figure 4.17: Example of the operation of SAMCRA (initialization).

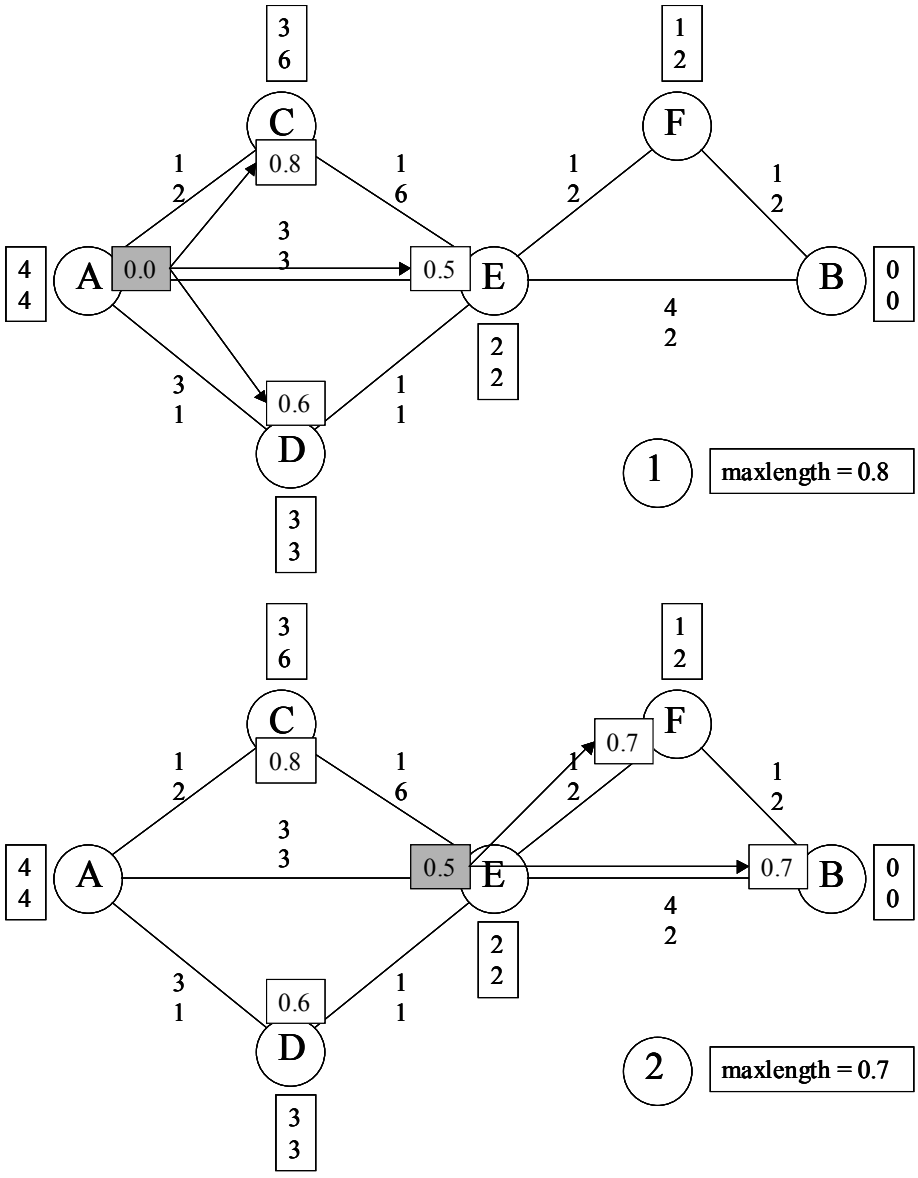


Figure 4.18: Example of the operation of SAMCRA (steps 1 and 2).

$i = 1, 2$, and $P^*_{A\to B;2}$ has length $l(P^*_{A\to B;2}) = 0.8$, which is smaller than the initialized *maxlength* $= 1$. *maxlength* therefore is lowered[6] to 0.8.

The main algorithm starts in step 1 of Figure 4.18 by scanning the neighbors of node A. In step 2, the path with the minimum predicted end-to-end length, which corresponds to node E in Figure 4.18 with $l(\vec{w}(P_{AE})+\vec{b}(E)) = 0.5$, is extracted from the queue and the scanning procedure from E is invoked. The path P_{AED}, with $\vec{w}\,(P_{AED}) = (4, 4)$, is not stored because it is dominated by the previously stored path P_{AD} stored at node D. The same holds for the path towards node C, P_{AEC}. Besides being dominated by the previously stored path P_{AC}, its length $l(\vec{w}(P_{AEC}))$ also exceeds *maxlength*. The paths toward nodes B and F are stored and *maxlength* is updated with the length of the end-to-end path P_{AEB}. In step 3, shown in Figure 4.19, the scanning procedure from node D is invoked. In step 4, two subpaths are stored at node F: P_{AEF} with $\vec{w}\,(P_{AEF}) = (4, 5)$ and predicted length $l(\vec{w}(P_{AEF}) + \vec{b}(F)) = 0.7$, and P_{ADEF} with $\vec{w}\,(P_{ADEF}) = (5, 4)$ and predicted length $l(\vec{w}\,(P_{ADEF})+\vec{b}(F)) = 0.6$. In step 5, in Figure 4.20, a new end-to-end path P_{ADEFB} is found with predicted length 0.6. *maxlength* is therefore set to 0.6. Note that this predicted length equals the real length, because we have attained a complete path from A to B. In the next and final step, the destination node B is extracted. This implies that the shortest path, minimizing the length (4.3) and within the constraints, has been found. By using the predecessor list π, this shortest path P_{ADEFB} is reconstructed in the reverse direction (as in Dijkstra's algorithm).

Since the granularity is 1 (the vector components are all integers), we observe that, although with (4.5) $k_{\max} = 10$, $k_{\min} = 2$ suffices for the exact solution, because two queue entries are needed at node F, while all the other nodes store less entries. If k was restricted to 1, no path satisfying the constraints would have been found. SAMCRA guarantees that, if there is a compliant path, this path is always found.

4.7 Conclusions

Four concepts for exact QoS routing were identified: a non-linear length function, the k-shortest paths approach, the principle of non-dominance and the look-ahead technique. The multiple QoS constraints make any length function non-linear. Motivated by this constraints surface we have proposed the non-linear length function $l(P) = \max_{i=1,\ldots,m}\left(\frac{w_i(P)}{L_i}\right)$. A problem of the inherent non-linearity is that subsections of shortest paths are not necessarily shortest themselves, which necessitates to evaluate multiple paths at a node. This is accomplished by a k-shortest paths approach. In the worst-case this k-shortest paths approach could evaluate all possible paths and therefore efficient search-space-reducing techniques are required. Two such techniques are

[6] If only a solution to the MCP problem is required, the algorithm can be stopped since a feasible path from A to B has been found.

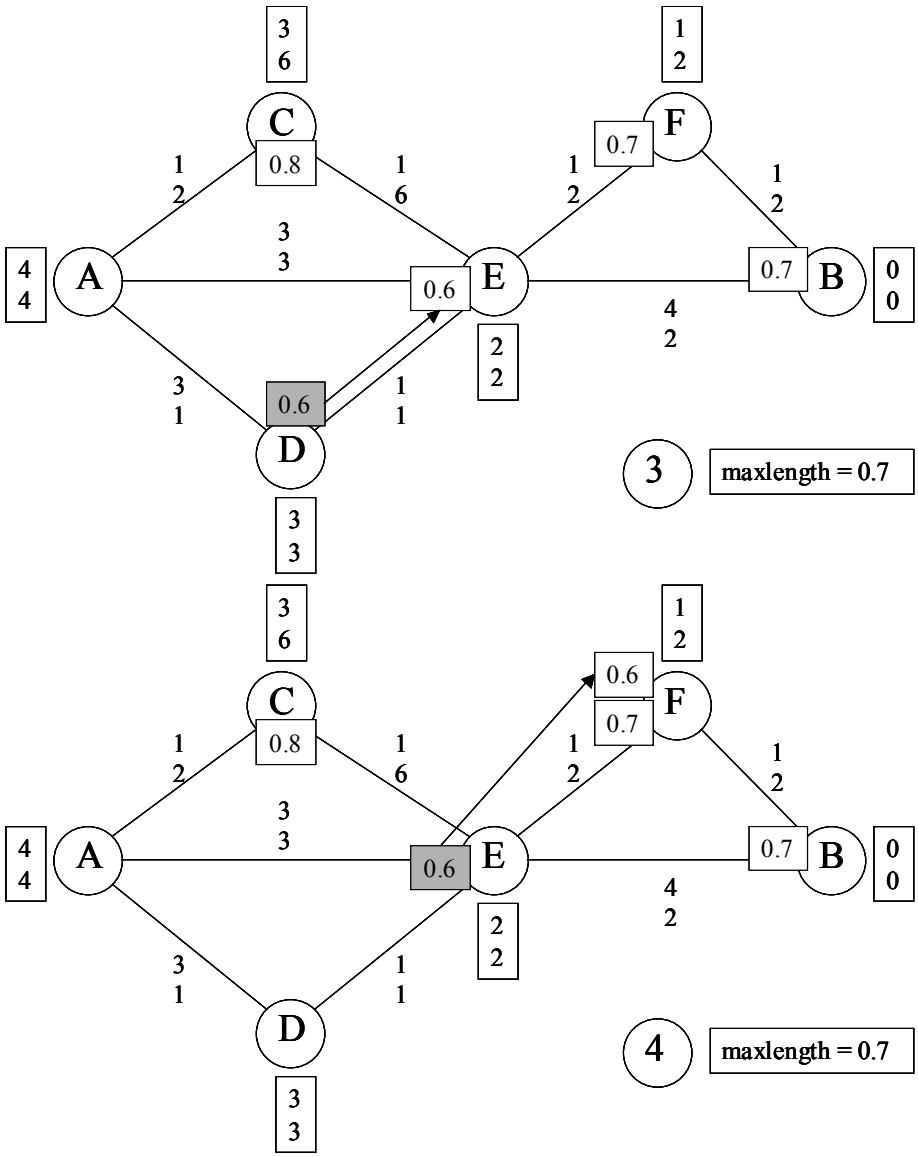


Figure 4.19: Example of the operation of SAMCRA (steps 3 and 4).

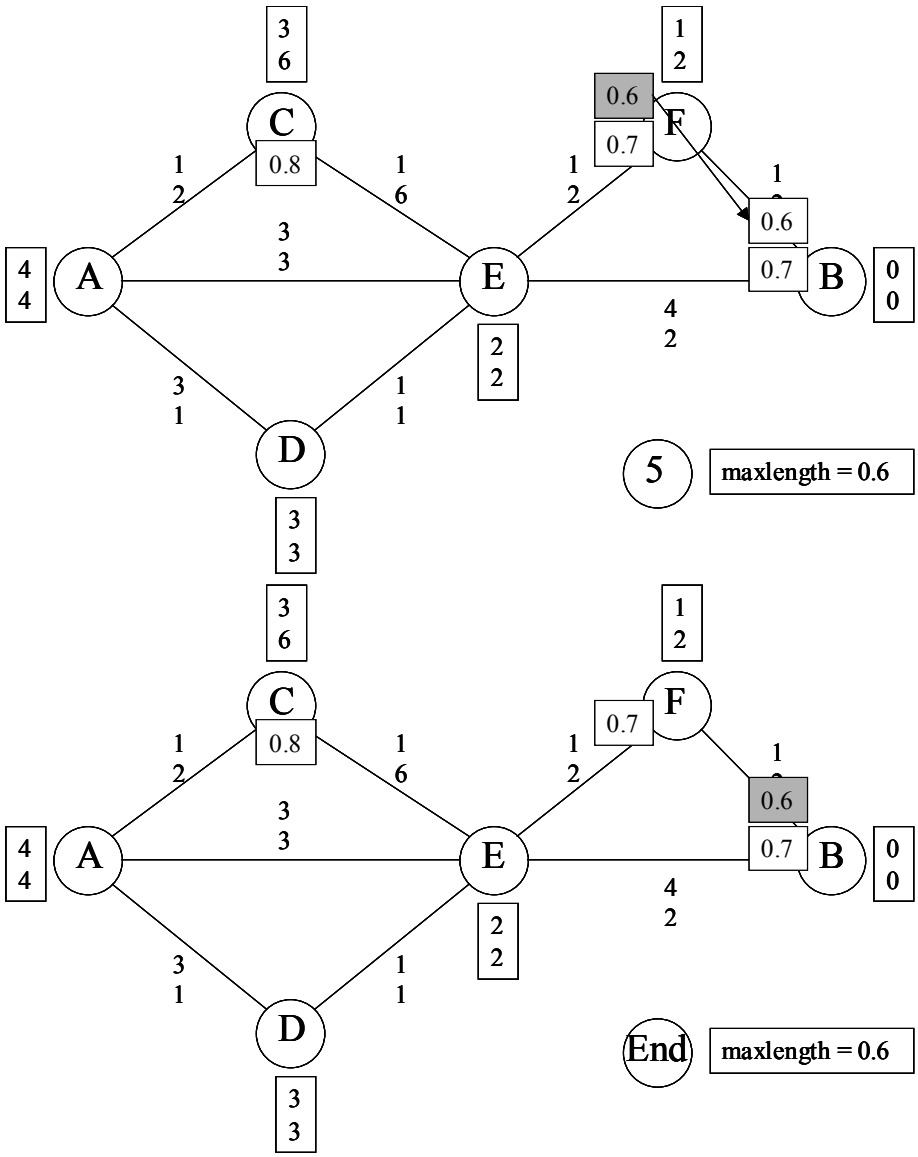


Figure 4.20: Example of the operation of SAMCRA (steps 5 and 6).

"the principle of non-dominance" that excludes paths P' for which already a path P is known that has weights $w_i(P) \leq w_i(P')$, for $i = 1, ..., m$ and "the concept of look-ahead." Look-ahead computes lower-bound vectors to assist in determining whether a path can become feasible or not. Based on these four concepts we have proposed the exact algorithm SAMCRA. Naturally, in addition to the four concepts, others may exist, e.g. a preprocessing of the graph that prunes links that cannot be on the shortest path or a bi-directional search. Based on such new concepts SAMCRA could evolve over time and maintain the top-position it has acquired today.

Chapter 5

Overview of QoS algorithms

Finding a path subject to multiple constraints is known to be an NP-complete problem. Hence, accurate constraint-based path selection algorithms with a fast running time are scarce. Numerous heuristics and a few exact algorithms have been proposed. In this chapter, we compare the lion's share of these algorithms. The main focus is on *multi-constrained path* algorithms, which are classified under heuristics, ϵ-approximation algorithms and exact algorithms. The performance evaluation of these algorithms is presented based on complexity analysis and simulation results and may shed some light on the difficult task of selecting the proper algorithm for a QoS-capable network. Also some attention is given to algorithms that do not solve the MCP problem, but which were proposed to solve specific instances of the MCP problem, like the restricted shortest path problem.

5.1 Heuristics

5.1.1 Jaffe's algorithm

Jaffe [85] presented two MCP algorithms for the $m = 2$ dimensional case. The first is an exact pseudo-polynomial-time algorithm with a worst-case complexity of $O(N^5 b \log Nb)$, where b is the largest weight in the graph. Because of this prohibitive complexity, only the second algorithm, hereafter referred to as Jaffe's algorithm, is discussed. Jaffe proposed using a shortest path algorithm on a linear combination of the two link weights:

$$w(u, v) = d_1 \cdot w_1(u, v) + d_2 \cdot w_2(u, v) \tag{5.1}$$

where d_1 and d_2 are positive multipliers.

As discussed in Section 4.1, the shortest path based on a linear combination of link weights does not necessarily reside within the constraints. Jaffe had also observed this fact and therefore evaluated a non-linear path length function of the form $f(P) =$

$\max\{w_1(P), L_1\} + \max\{w_2(P), L_2\}$, whose minimization can guarantee to find a feasible path, if such a path exists. However, because no simple shortest path algorithm can cope with this non-linear length function, Jaffe approximated the non-linear length by the linear length function (4.1). Andrew and Kusuma [4] generalized Jaffe's analysis to an arbitrary number of constraints, m, by extending the linear length function to

$$l(P) = \sum_{i=1}^{m} d_i w_i(P)$$

and the non-linear function to

$$l(P) = \sum_{i=1}^{m} \max\left(w_i(P),\, L_i\right)$$

By choosing $d_i = \frac{1}{L_i}$ we maximize the volume of the solution space that can be scanned by linear equilength lines subject to $w_i(P) \leq L_i$. Furthermore, when using Dijkstra's algorithm [40] with Fibonacci heaps, the complexity for Jaffe's algorithm becomes $O(N \log N + mM)$.

If the returned path is not feasible, then Jaffe's algorithm stops, although the search could be continued by using different values for d_i, which might result in a feasible path. This type of search is referred to as Lagrangian relaxation, in which the values d_i are altered by the algorithm itself. Unfortunately, in some cases, even if all possible combinations of d_i are exhausted, a feasible path may not be found using linear search. As shown in Chapter 4, an exact algorithm must necessarily use a non-linear length function, even though a non-linear function cannot be minimized with a simple shortest path algorithm.

5.1.2 Iwata's algorithm

Iwata *et al.* [84] proposed a polynomial-time heuristic to solve the MCP problem. The algorithm first computes one (or more) shortest path(s) based on one QoS measure and then checks if all the constraints are met. If this is not the case, the procedure is repeated with another measure until a feasible path is found or all QoS measures are examined. Like Jaffe's algorithm, Iwata's algorithm is also a special case of Lagrangian relaxation. A similar approach has been proposed by Lee *et al.* [104].

The problem with Iwata's algorithm is that there is no guarantee that any of the shortest paths, with respect to each individual measure, is close to a path within the constraints. This is illustrated in Figure 5.1, which shows twenty paths of a two-constraint problem applied to a random graph with 100 nodes. Only the second and third shortest path for measure 1, and the second and fourth shortest path for measure 2, lie within the constraints.

Via Dijkstra's algorithm and when only considering one shortest path per QoS measure, we can obtain a complexity of $O(mN \log N + mM)$.

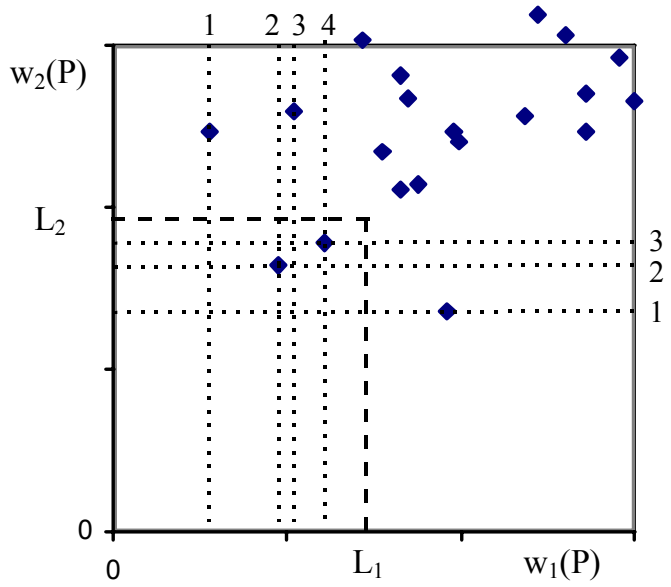


Figure 5.1: Twenty shortest paths for a two-constraint problem. Each path is represented as a dot and the coordinates of each dot are its path-weights for each measure individually.

5.1.3 TAMCRA

TAMCRA is short for a "Tunable Accuracy Multiple Constraints Routing Algorithm" [38], [37]. TAMCRA is a heuristic that is based on three concepts: (1) a non-linear measure for the path length, (2) a k-shortest path approach [32], and (3) the principle of non-dominated paths [77]. These three principles were explained in Chapter 4.

1. *non-linear path-length measure.* Motivated by the geometry of the constraints surface in m-dimensional space, the length of a path P is defined as $l(P) = \max_{1 \leq i \leq m} \left(\frac{w_i(P)}{L_i} \right)$.
   A solution to the MCP problem is a path whose weights are all within the constraints: $l(P) \leq 1$.

2. *k-shortest path algorithm.* This algorithm (e.g., as presented in [32]) is essentially Dijkstra's algorithm, with extensions to return not only the shortest path to a given destination, but also the second shortest, the third shortest, ..., up to the $k-$th shortest path. In TAMCRA the k-shortest path concept is applied to intermediate nodes i on the path from the source node s to the destination node t to keep track of multiple sub-paths from s to i. In TAMCRA the maximum queue size k allowed at a node is predetermined by the user.

3. *Principle of non-dominance.* A path Q is said to be dominated by a path P if $w_i(P) \leq w_i(Q)$ for $i = 1, .., m$, with an inequality for at least one i. This property allows TAMCRA to efficiently reduce the search space without compromising the solution.

TAMCRA has a worst-case complexity of

$$O(kN \log(kN) + k^2 mM)$$

In TAMCRA the allocated queue space is predefined via k and hence the complexity is polynomial. The accuracy of TAMCRA can be tuned via k, where better performance can be achieved with a larger k. Simulation results for different values for k can be found in [38].

5.1.4 Chen's algorithm

Chen and Nahrstedt [27] provided an approximate algorithm for the MCP problem. This algorithm first transforms the MCP problem into a simpler problem by scaling down $m-1$ (real) link weights to integer weights as follows:

$$w_i^*(u,v) = \left\lceil \frac{w_i(u,v) \cdot x_i}{L_i} \right\rceil \text{ for } i = 2, 3, \ldots, m,$$

where x_i are predefined positive integers. The simplified problem consists of finding a path P, for which $w_1(P) \leq L_1$ and $w_i^*(P) \leq x_i$, $2 \leq i \leq m$. A solution to this simplified problem is also a solution to the original MCP problem, but not necessarily vice versa (because the conditions of the simplified problem are more strict). Since the simplified problem can be solved exactly, Chen and Nahrstedt have shown that *the MCP problem can be exactly solved in polynomial time provided that at least $m-1$ QoS measures have bounded integer weights.*

To solve the simplified MCP problem, Chen and Nahrstedt proposed two algorithms based on dynamic programming: the Extended Dijkstra's Shortest Path algorithm (EDSP) and the Extended Bellman-Ford algorithm (EBF). The algorithms return a path that minimizes the first (real) weight, provided that the other $m-1$ (integer) weights are within the constraints. According to Chen and Nahrstedt, the EBF algorithm is expected to give better performance in terms of execution time when the graph is sparse and the number of nodes is relatively large.

The complexities of EDSP and EBF are $O(x_2^2 \cdots x_m^2 N^2)$ and $O(x_2 \cdots x_m NM)$, respectively. To achieve good performance, large x_i's are needed, which makes this approach rather computationally intensive for practical purposes. By adopting the concept of non-dominance, like in SAMCRA, this algorithm could[1] reduce its search space, resulting in a faster execution time.

[1] All algorithms in Section 5.5 maintained their original forms, without any possible improvements.

5.1.5 Randomized algorithm

Korkmaz and Krunz [97] proposed a randomized heuristic for the MCP problem. The concept behind randomization is to make random decisions during the execution of an algorithm [118], so that unforeseen traps can potentially be avoided when searching for a feasible path. The proposed randomized algorithm is divided into two parts: an initialization phase and a randomized search. In the initialization phase, the algorithm computes the shortest paths from every node u to the destination node t with respect to each QoS measure and with respect to the linear combination of all m measures. This look-ahead information (see Section 4.4) will provide lower bounds for the path weight vectors of the paths from u to t. Based on the information obtained in the initialization phase, the algorithm can decide whether there is a chance of finding a feasible path or not. If so, the algorithm starts from the source node s and explores the graph using a randomized breadth-first search (BFS, see Chapter 3). In contrast to conventional BFS, which systematically discovers every node that is reachable from node s, the randomized BFS discovers nodes from which there is a good chance to reach the destination t. By using the information obtained in the initialization phase, the randomized BFS can check whether this chance exists before discovering a node. If there is no chance of reaching the destination, the algorithm foresees the trap and avoids exploring such nodes any further. In the randomized algorithm, the objectives of the look-ahead property are twofold. First, the lower-bound vectors obtained in the initialization phase are used to check whether a sub-path from s to u can become a feasible path. This is a search-space-reducing technique. Second, a different preference rule for extracting nodes can be adopted based on the predicted end-to-end length, i.e. the length of the sub-path weight vector plus the lower-bound vector. The randomized BFS continues searching by randomly selecting discovered nodes until the destination node is reached. If the randomized BFS fails in the first attempt, it is possible to execute only the randomized BFS again so that the probability of finding a feasible path can be increased.

Under the same network conditions, multiple executions of the randomized algorithm may return different paths between the same source and destination pair. However, some applications might require the same path again. In such cases, path caching should be used [132].

The worst-case complexity of the randomized algorithm is $O(mN \log N + mM)$.

5.1.6 H_MCOP

Korkmaz and Krunz [96], [98] also provided a heuristic called H_MCOP. This heuristic tries to find a path within the constraints by using the non-linear path length function (4.3) of SAMCRA. In addition, H_MCOP tries to simultaneously minimize the weight of a single "cost" measure along the path. To achieve both objectives simultaneously,

H_MCOP executes two modified versions of Dijkstra's algorithm in the backward and forward directions. In the backward direction, H_MCOP uses Dijkstra's algorithm for computing the shortest paths from every node to the destination node t with respect to $w(u,v) = \sum_{i=1}^{m} \frac{w_i(u,v)}{L_i}$. Later on, these paths from every node u to the destination node t are used to estimate how suitable the remaining sub-paths are. In the forward direction, H_MCOP uses a modified version of Dijkstra's algorithm. This version starts from the source node s and discovers each node u based on a path P, where P is a heuristically determined complete s-t path that is obtained by concatenating the already traveled sub-path from s to u and the estimated remaining sub-path from u to t. Since H_MCOP considers complete paths before reaching the destination, it can foresee several infeasible paths during the search. If paths seem feasible, then the algorithm can switch to explore these feasible paths based on the minimization of the single measure. Although similar to the look-ahead property, this technique only provides a preference rule for choosing paths and cannot be used as a search-space-reducing technique.

The complexity of the H_MCOP algorithm is $O(N \log N + mM)$. If one deals only with the MCP problem, then H_MCOP could be stopped whenever a feasible path is found during the search in the backward direction, reducing the computational complexity. The performance of H_MCOP in finding feasible paths can be improved by using the k-shortest path algorithm and by eliminating dominated paths.

5.1.7 Limited path heuristic

Yuan [179] presented two heuristics for the MCP problem. The first "limited granularity" heuristic has a complexity of $O(N^m M)$, whereas the second "limited path" heuristic (LPH) has a complexity of $O(k^2 NM)$, where k corresponds to the queue size at each node. The author claims that when $k = O(N^2 \log_2 N)$, the limited path heuristic has a very high probability of finding a feasible path, provided that such a path exists. However, applying this value results in an excessive execution time.

According to Yuan, the performance of both algorithms is comparable when $m \leq 3$. For $m > 3$, LPH performs better than the limited granularity heuristic. Moreover, the limited granularity heuristic closely resembles the algorithm of Chen and Nahrstedt (discussed in Section 5.1.4).

LPH is an extended Bellman-Ford algorithm that uses two of the concepts of TAMCRA. Both use the concept of non-dominance and maintain at most k paths per node. However, TAMCRA uses a *k-shortest* path approach, while LPH stores the first (and not necessarily shortest) k paths. Furthermore LPH does not check whether a sub-path obeys the constraints, but only checks at the end for the destination node. An obvious difference is that LPH uses a Bellman-Ford approach, while TAMCRA uses a Dijkstra-like search. Simulations (not shown) reveal that Bellman-Ford-like implementations require more execution time than Dijkstra-like implementations, especially when the graphs are dense.

5.2 ϵ-approximation

An ϵ-approximation algorithm is an algorithm that is not necessarily exact, but which can provide a solution quantifiably close to the exact solution. The solution provided by an ϵ–approximation algorithm is guaranteed to be within a factor $(1+\epsilon)$ of the exact solution, where $\epsilon > 0$. Such a performance guarantee is not provided by heuristics and in this sense ϵ–approximation algorithms are considered to be better than heuristics. Unfortunately, their complexity is a function of $\frac{1}{\epsilon}$ and therefore their running time in practice is usually excessive.

Almost all proposed ϵ–approximation algorithms in the field of QoS routing focus on the RSP problem and can therefore only handle two QoS measures/constraints. Warburton [172] was the first to develop a fully polynomial-time approximation scheme (FPTAS) for the RSP problem, assuming acyclic graphs. Hassin [73] improved this algorithm and provided two ϵ-optimal approximation algorithms with the complexities of $O((\frac{MN}{\epsilon}+1)\log\log B)$ and $O(\frac{MN^2}{\epsilon}\log(\frac{N}{\epsilon}))$, where B is an upper bound on the cost $c(P)$ of a path. It is assumed that the link weights are positive integers. Hassin's first ϵ-optimal approximation algorithm initially determines an upper bound (UB) and a lower bound (LB) on the optimal cost denoted by OPT. For this, the algorithm initially starts with $LB = 1$ and $UB =$ sum of (N-1) largest link-costs, and then systematically adjusts them using a *testing* procedure. Once these bounds are found, the approximation algorithm bounds the cost of each link by rounding and scaling it according to: $c'(u,v) = \left\lfloor \frac{c(u,v)(N-1)}{\epsilon LB} \right\rfloor \ \forall\ (u,v) \in E$. Finally, it applies a pseudo-polynomial-time algorithm on these modified weights. The second approximation algorithm uses a slightly different technique called interval partitioning, in which a set of positive numbers $Q = \{p_1, p_2, \ldots, p_m\}$ is partitioned into subsets $R_1, \ldots, R_{r+1}$ such that $p_i \in R_j$ if and only if $\frac{X(j-1)}{r} < p_i \leq \frac{Xj}{r}$ for $j = 1, \ldots, r$, and $p_i \in R_{r+1}$, if and only if $p_i > X$, where X is a given positive number. Phillips [133] and Lorenz and Raz [111] provided further improvements of which the latter has the best complexity $O(MN(\log\log N + \frac{1}{\epsilon}))$.

Orda [125] and Lorenz *et al.* [112] modified ϵ-optimal approximation algorithms to scale better in hierarchical networks. Ergün *et al.* [45] proposed an ϵ-optimal approximation algorithm for a RSP-related problem, in which one link weight is a function of the other. Goel *et al.* [59] considered a related problem, in which the least-cost path from a given source to all destinations is searched, while satisfying the delay constraint Δ for each path. For this problem, Goel *et al.* provided an ϵ-approximation algorithm with the complexity of $O((M + N\log N)\frac{D}{\epsilon})$, where D can be at most $N-1$.

In comparison with the number of ϵ–approximation algorithms that solve the RSP problem, only very few papers propose ϵ–approximation algorithms for the MCP problem. In this section the contribution of two papers [136], [177] is discussed.

5.2.1 Puri's algorithm

Puri and Tripakis [136] presented three algorithms for routing with multiple constraints (the MCP problem), an exact pseudo-polynomial-time algorithm, an ϵ–approximation algorithm and a heuristic. The algorithms were first presented for the case $m = 2$, after which the generalization to $m \geq 2$ of the algorithms was dealt with. The pseudo-polynomial algorithm (like the algorithm of Yuan [179]) uses a Bellman-Ford-like search and removes unfeasible and dominated paths. The ϵ–approximation algorithm is based on their pseudo-polynomial algorithm, but it uses a more strict principle of non-dominance, where non-dominated paths that are "too close" to another non-dominated path are considered dominated (see ε-dominance in Section 4.3). The authors [136] have defined a step error $\varepsilon = \frac{\min(L_1,L_2)\epsilon}{N}$ for the case $m = 2$ and have increased the constraints by factor $(1+\epsilon)$. Thus only the set of ε-dominated paths with distance ε is maintained. The total error accumulated over a path is then upper bounded by $\min(L_1, L_2)\epsilon$ and the constraints are therefore never exceeded more than this error. The authors have claimed that an extension to $m > 2$ is trivial and that the worst-case complexity of this algorithm equals $O(N^m M(1 + \frac{1}{\epsilon})^m)$, but this statement is wrong. Such a complexity can be attained only at the loss of ϵ-approximation. The heuristic proposed by Puri and Tripakis is essentially a Lagrangian relaxation approach (see Section 5.1) and is not further discussed here.

5.2.2 Xue's algorithm

Xue, Sen and Banka [177] proposed an ϵ–approximation algorithm that solves the MCOP problem. Like Puri's algorithm [136], first an exact pseudo-polynomial-time algorithm was devised, after which the pseudo-polynomial algorithm was reduced via some techniques to the ϵ–approximation algorithm. The pseudo-polynomial algorithm solves a special case of the MCP problem, where all constraints have the same value. It achieves this by first constructing a directed graph G_L with node set $V_L = V\{0, 1, ..., L\}^{m-1}$ and edge set E_L. If (u, v) is an undirected edge in E, then E_L contains directed edges from $(u, d_2[u],..., d_m[u])$ to $(v, d_2[v],..., d_m[v])$, where $d_i[u] = d_i[v] + w_i(u, v)$. The length of all such edges is $w_1(u, v)$. In addition, E_L also contains zero length edges from $(t, d_2[t],$ $... , d_j[t], ... , d_m[t])$ to $(t, d_2[t], ... , d_j[t] + 1, ... , d_m[t])$ for one $j \in \{2, 3, ..., m\}$. Based on this graph, the shortest paths from $(s, 0, ..., 0)$ to all other nodes in G_L are computed and if a path with length $\leq L$ exist towards $(t, L, L, ..., L)$, then this path is a feasible path. The worst-case complexity of this pseudo-polynomial algorithm is $O(ML^{m-1} + NL^{m-1} \log(NL^{m-1}))$. Xue *et al.* applied an approximate test procedure (similar to Hassin's test procedure [73]) to the pseudo-polynomial algorithm along with scaling and rounding of the weights to arrive at an ϵ–approximate solution. The worst-case complexity of this ϵ–approximation algorithm is $O((MN^{m-1} + N^m \log(N^m)) \log\log\left(\frac{mM}{2}\right)$ $+ \left(\frac{1}{\epsilon}\right)^{m-1} \left(MN^{m-1} + N^m \log\left(\frac{N^m}{\epsilon^{m-1}}\right)\right))$. The authors have recognized that the complex-

ity of their algorithm is high and even claimed that this holds for almost all FPTAS algorithms. The simulations we have performed with ϵ-approximation algorithms agree with this claim.

5.3 Exact algorithms

Because the MCP problem is NP-complete, only a few exact algorithms were proposed. In this section these exact algorithms are elucidated.

5.3.1 SAMCRA

SAMCRA (see Section 4.6) stands for a Self-Adaptive Multiple Constraints Routing Algorithm and is the exact successor of TAMCRA [38]. SAMCRA is based on four fundamental concepts: (1) a non-linear measure for the path length, (2) a k-shortest path approach [32], (3) the principle of non-dominated paths [77] and (4) the concept of look-ahead. These four principles were explained in Chapter 4.

1. *non-linear path-length measure.* The length of a path P is defined, as $l(P) = \max_{1 \leq i \leq m} \left(\frac{w_i(P)}{L_i} \right)$. Depending on the specifics of a constrained optimization problem, SAMCRA can be used with different length functions, provided they obey the criteria for length in vector algebra. Examples of length functions were given in Chapter 4.

2. *k-shortest path algorithm.* In SAMCRA the k-shortest path concept is applied to intermediate nodes i on the path from the source node s to the destination node t to keep track of multiple sub-paths from s to i. The value of k is adaptively controlled by SAMCRA.

3. *Principle of non-dominance.* A path Q is said to be dominated by a path P, if $w_i(P) \leq w_i(Q)$ for $i = 1, .., m$, with an inequality for at least one i. SAMCRA only considers non-dominated (sub)-paths.

4. *Concept of look-ahead.* First calculating paths in polynomial time from the destination and then applying this information to find a feasible path between the source and destination is especially useful when graphs become "hard to solve," i.e., N, M and m grow large. This look-ahead property allows us to compute lower bounds on end-to-end paths, which can be used to check the feasibility of paths. Moreover, better preference rules can be adopted to extract nodes from the queue.

SAMCRA has a worst-case complexity of

$$O(kN\log(kN) + k^2mM)$$

SAMCRA self-adaptively controls the value for k, which can grow exponentially in the worst case. Knowledge about k is crucial to the complexity of SAMCRA.

The self-adaptivity in k makes SAMCRA an exact MCOP algorithm: SAMCRA guarantees to find the shortest path within the constraints, provided that such a path exists. In this process, SAMCRA only allocates queue space when truly needed and self-adaptively adjusts the number of stored paths k in each node.

5.3.2 HAMCRA

We [102] have also proposed an exact hybrid MCP algorithm that integrates the speed of TAMCRA with the exactness of SAMCRA. Both SAMCRA and TAMCRA use a non-linear length function, the k-shortest path approach and only consider non-dominated paths.

Since HAMCRA is composed of SAMCRA and TAMCRA, it is also based on these three concepts. In HAMCRA, first the TAMCRA algorithm is executed with a queue-size $k = 1$ from the destination node to all other nodes in the graph. This is similar to bi-directional search, because HAMCRA also scans from the destination node. However, the scanning procedure does not alternate between the source and the destination. TAMCRA could also be used with $k > 1$, which would lead to a better accuracy at the cost of increased complexity (of TAMCRA). The running time of TAMCRA (with $k = 1$) is comparable to that of the Dijkstra algorithm. At each node, the path weight vector found by TAMCRA from that node to the destination is stored. These values will later be used to predict the end-to-end path length. If TAMCRA has found a path within the constraints between the source and the destination, HAMCRA can stop and return this path. If TAMCRA was not able to return a feasible path, HAMCRA continues by executing the SAMCRA algorithm from the source node. The difference between HAMCRA and SAMCRA is that HAMCRA uses the information obtained by TAMCRA and only stores predicted end-to-end lengths in the queue, instead of the real lengths of the sub-paths. The predicted end-to-end length is found by summing the real weights of a path from source s to the intermediate node u with the weights of the TAMCRA path from u to the destination t. The algorithm continues searching in this way until a feasible path from s to t is found or until the queue is empty.

HAMCRA also uses the look-ahead technique discussed in Section 4.4 to reduce the search space. However, the difference with SAMCRA is that HAMCRA uses TAMCRA instead of the lower-bounds for its predictions. Such a prediction (if erroneous) could be larger than the real end-to-end path length. Ergo, HAMCRA may extract a non-shortest path first. Consequently, HAMCRA using TAMCRA cannot guarantee an

exact solution to the MCOP problem. If $l_{predicted}(P) \leq l_{actual}(P)$, as is the case with lower-bound predictions, a solution to MCOP can be guaranteed. Unfortunately, simulations have shown that such lower-bound predictions are usually not as good as the TAMCRA predictions, leading to an increased running time [102]. The hybrid combination of SAMCRA and TAMCRA, on the other hand, seems "a winning combination" for the MCP problem.

5.3.3 A*Prune

Liu and Ramakrishnan [109] considered the problem of finding not only one but multiple (K) shortest paths satisfying the constraints. The length function used is the same as Jaffe's length function. The authors proposed an exact algorithm called A*Prune. If there are no K feasible paths present, the algorithm will only return those that are within the constraints. For the simulations K was assigned the value 1.

For each QoS measure, A*Prune first calculates the shortest paths from the source s to all nodes $i \in V\backslash\{s\}$ and from the destination t to all nodes $i \in V\backslash\{t\}$. The weights of these paths will be used to evaluate whether a certain sub-path can indeed become a feasible path (similar look-ahead features were also used in [97]). After this initialization phase, the algorithm proceeds in a Dijkstra-like fashion. The node with the shortest predicted end-to-end length[2] is extracted from a heap and then all of its neighbors are examined. The neighbors that form a loop or lead to a violation of the constraints are pruned. The A*Prune algorithm continues extracting/pruning nodes until K constrained shortest paths from s to t are found or until the heap is empty.

The A*Prune algorithm is therefore similar to the SAMCRA algorithm, except that the principle of non-dominance is not used and an other length function is adopted. The worst-case complexity of A*Prune is $O(N!(m + h + N \log N))$, where h is the number of hops of the retrieved path. The authors [109] have mentioned that it is possible to implement a Bounded A*Prune algorithm with a polynomial-time complexity, at the risk of loosing exactness.

5.4 Special (non-MCP) QoS algorithms

Several proposals in the literature have aimed at addressing special, yet important, sub-problems in QoS routing. For example, researchers addressed QoS routing in the context of bandwidth and delay. Routing with these two measures is not NP-complete. Wang and Crowcroft [171] presented a *bandwidth-delay based routing algorithm*, which simply prunes all links that do not satisfy the bandwidth constraint and then finds the shortest path with respect to delay in the pruned graph. A much researched problem

[2] The length function is a linear function of all measures. If there are multiple sub-paths with equal predicted end-to-end lengths, the one with the so-far shortest length is chosen.

is the NP-complete *Restricted Shortest Path* (RSP) problem (see Section 1.4). In the literature, the RSP problem is also studied under different names such as the delay-constrained least-cost path, minimum-cost restricted-time path, and constrained shortest path. Many heuristics were proposed for this problem, e.g., [73], [138], [90], [67]. Several path selection algorithms based on different combinations of bandwidth, delay, and hop count were discussed in [125] (for example widest-shortest path and shortest-widest path). In addition, new algorithms were proposed to find more than one feasible path with respect to bandwidth and delay (for instance Maximally Disjoint Shortest and Widest Paths) [156]. Kodialam and Lakshman [94] proposed bandwidth guaranteed dynamic routing algorithms. Orda and Sprintson [126] considered the pre-computation of paths with minimum hop count, and bandwidth guarantees. They also provided some approximation algorithms that take into account certain constraints during the pre-computation. Guerin and Orda [62] focused on the impact of advance reservation on the path selection process. They described possible extensions to path selection algorithms in order to make them advance-reservation aware and evaluated the added complexity introduced by these extensions. Fortz and Thorup [52] investigated how to set link weights based on previous measurements so that the shortest paths can provide better load balancing and can meet the desired QoS constraints. The path selection problem becomes simpler when dependencies exist between the QoS measures, for example as a result of implementing specific scheduling schemes at network routers [113]. Specifically, if Weighted Fair Queueing (WFQ) scheduling is employed at the routers, and the constraints are on bandwidth, queueing delay, jitter, and loss, then the problem can be reduced to a standard shortest path problem by representing all the constraints in terms of bandwidth. However, although queueing delay can be formulated as a function of bandwidth, this is not the case for the propagation delay, which cannot be ignored in high-speed networks.

5.5 Performance evaluation

In this section we will evaluate MCP algorithms via simulations and complexity analysis.

5.5.1 Simulation set-up

We have simulated with Waxman graphs and lattices. The weights of a link were assigned independent uniformly distributed random variables in the range (0, 1]. We also simulated with two negatively correlated QoS measures, for which the link weights were assigned as follows: w_1 was uniformly distributed in the range (0, 1] and $w_2 = 1 - w_1$.

The choice of the constraints is important, because it determines how many (if any) feasible paths exist. We adopted two sets of constraints, namely strict and loose

constraints. We have omitted the results for loose constraints, because under loose constraints all MCP algorithms obtained a (near) optimal success rate in a low *execution time*. For MCOP algorithms, loose constraints increase the number of feasible paths and hence the search space. This makes it difficult to find the optimal path. Fortunately, MCOP algorithms can be easily adapted to solve only MCP, by stopping as soon as a feasible path is reached. The set of strict constraints was chosen as follows:

$$L_i = w_i(P),\ i = 1, ..., m$$

where P is the path for which $\max_{j=1,...m}(w_j(P))$ is minimum. In this case only one feasible path is present in the graph and hence MCP equals MCOP. This allows us to fairly compare MCP and MCOP algorithms.

During all simulations, the *success rate* and the *normalized execution time* were stored. The *success rate* of an algorithm is defined as the number of times that an algorithm returned a feasible path divided by the total number of iterations. The *normalized execution time* of an algorithm is defined as the *execution time* of the algorithm (over all iterations) divided by the *execution time* of Dijkstra's algorithm.

5.5.2 Simulation results

Our simulations revealed that the ϵ-approximation algorithms and the Bellman-Ford-based algorithms (Chen's algorithm and the Limited Path Heuristic) required significantly more *execution time* than their Dijkstra-based counterparts. These algorithms have therefore been omitted from the results presented here.

Figure 5.2 gives the *success rate* and *normalized execution time* for the class of Waxman graphs and lattices, with $m = 2$. The exact algorithms SAMCRA and A*Prune always give *success rate* = 1. The difference in the *success rate* of the heuristics under strict constraints is significant. Jaffe's algorithm and Iwata's algorithm perform significantly worse than the others. In the class of two-dimensional lattices with negatively correlated weights this difference disappears as the *success rates* of *all* heuristics tend to zero as N increases, even for fairly small N.

Figure 5.2 also displays the *normalized execution time* that the algorithms needed to obtain the corresponding *success rate*. For the class of Waxman graphs (with independent link weights), the *execution time* of the exact algorithm SAMCRA does not deviate much from the polynomial time heuristics. In fact, all algorithms display a polynomial *execution time*. For the class of lattices (with negatively correlated link weights), the *execution times* of the exact algorithms grow exponentially, which is the price paid for exactness in hard topologies.

We have also simulated the performance of the algorithms as a function of the number of constraints m ($m = 2$, 4, 6, and 8) under independent uniformly distributed link weights. The results for the class of Waxman graphs ($N = 100$) and lattices

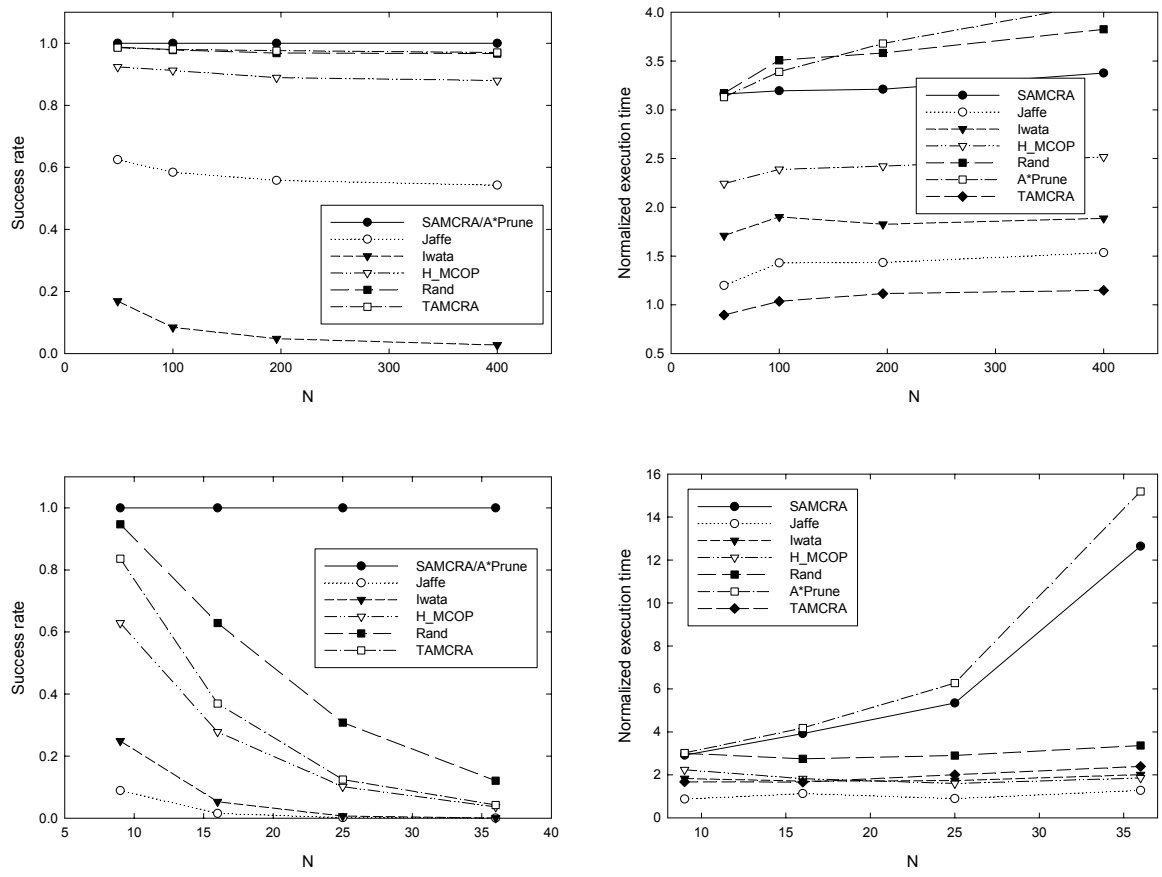


Figure 5.2: For $m = 2$ and under strict constraints, the success rate (left) and normalized execution time (right) for the class of Waxman graphs (above) and lattices (below) as a function of the number of nodes.

($N = 49$) are plotted in Figure 5.3. The algorithms display a similar ranking in *success rate* as in Figure 5.2. Some algorithms display a linear increase in execution time. All these algorithms have an initialization phase in which they execute the Dijkstra algorithm m times. Finally, if m grows, A*Prune slightly outperforms SAMCRA. This can be attributed to the non-dominance principle, which looses in strength if m grows. However, the time needed to check for non-dominance (under independent weights) is only manifested in a small difference between the *execution times* of SAMCRA and A*Prune.

5.5.3 Simulation conclusions

The conclusions presented here are only valid for the considered classes of graphs, namely the Waxman graphs and the square lattices. The simulation results indicated that SAMCRA-like algorithms performed best at an acceptable computational cost, which can be attributed to the following features, which were discussed in Chapter 4:

1. *Dijkstra-based search*
   Our simulations indicated that, even on sparse graphs, Dijkstra-like search runs significantly faster than a Bellman-Ford-like search.

2. *A non-linear length function*
   A non-linear length function is a prerequisite for exactness. When the link weights are positively correlated, a linear approach may give a high *success rate* in finding feasible paths, but under different circumstances the returned path may significantly violate the constraints.

3. *Search space reduction*
   Reducing the search space is always desirable, because this reduces the *execution time* of an algorithm. The non-dominance principle is a very strong search-space-reducing technique, especially when the number of constraints m is small. When m grows the look-ahead concept together with the constraint values provide a better search space reduction.

4. *Tunable accuracy through a k-shortest path functionality*
   Routing with multiple constraints may require that multiple paths be stored at a node, necessitating a k-shortest path approach. By tuning the value of k, a good balance between *success rate* and *computational complexity* may be reached.

5. *Look-ahead functionality*
   The look-ahead concept is based on information from path trees rooted at the destination, which are computed in polynomial time. These path trees are used to reduce the search space and to facilitate the search for a feasible path. In the latter functionality a predicted end-to-end path length may lead the search sooner in the correct direction, thereby saving in *execution time*.

The exactness of the TAMCRA-like algorithms depends on the value of k. If k is not restricted, then both MCP and MCOP problems can be solved exactly, as done by SAMCRA. Although k is not restricted in SAMCRA, simulations on Waxman graphs with independent uniformly distributed random link weights show that the *execution time* of this exact algorithm increases only linearly with the number of nodes, providing a scalable solution to the MC(O)P problem. Simulation results also show that TAMCRA-like heuristics with small values of k render near-exact solutions. The results

for the class of two-dimensional lattices with negatively correlated link weights are completely different. In such hard topologies, the heuristics are useless whereas the exact algorithms display an exponential execution time. Perhaps the best approach for such (unrealistic) graphs is via a hybrid algorithm (like HAMCRA [102]) that uses a good heuristic to make intelligent choices on which path to follow, combined with an exact SAMCRA-like algorithm that incorporates all the four above-mentioned concepts. If a solution to MCP suffices, then this algorithm should be stopped as soon as a feasible path is encountered. In Chapter 8 we will argue that the probability of encountering hard topologies is very low in practice.

5.6 Conclusions

Several researchers have investigated the constraint-based path selection problem and have proposed various algorithms, mostly heuristics. This chapter has evaluated these algorithms as proposed for the *multi-constrained (optimal) path* problem, via simulations in the class of Waxman graphs and the much harder class of two-dimensional lattices. Tables 5.1 and 5.2 display the worst-case complexities of the algorithms evaluated in this chapter.

Algorithm	time complexity
Jaffe's algorithm	$O(N \log N + mM)$
Iwata's algorithm	$O(mN \log N + mM)$
SAMCRA, TAMCRA	$O(kN \log(kN) + k^2 mM)$
EBF	$O(x_2 \cdots x_m NM)$
Randomized algorithm	$O(mN \log N + mM)$
H_MCOP	$O(N \log N + mM)$
A*Prune	$O(N!(m + N + N \log N))$

Table 5.1: Worst-case time complexity of the considered QoS path selection algorithms.

The simulation results show that the worst-case complexities of Tables 5.1 and 5.2 should be interpreted with care. For instance, the real execution time of H_MCOP will always be longer than that of Jaffe's algorithm under the same conditions, since H_MCOP executes the Dijkstra algorithm twice compared to one time for Jaffe's algorithm. In general, the simulation results indicate that SAMCRA-like algorithms that use a k-shortest path algorithm with a non-linear length function, while eliminating paths via the non-dominance and look-ahead concepts, give the better performance for the considered problems (RSP [99], MCP, MCOP). The performance and complexity of these algorithms is easily adjusted by controlling the value of k. When k is not restricted, the SAMCRA-like algorithms lead to exact solutions. In the class of

Algorithm	space complexity
Jaffe's algorithm	$O(N)$
Iwata's algorithm	$O(N)$
SAMCRA, TAMCRA	$O(kmN)$
EBF	$O(x_2 \cdots x_m N)$
Randomized algorithm	$O(mN)$
H_MCOP	$O(mN)$
A*Prune	$O(mN!)$

Table 5.2: Worst-case space complexity of the considered QoS path selection algorithms.

Waxman or random graphs with uniformly distributed link weights, simulation results suggest that the execution times of such exact algorithms increase linearly with the number of nodes. The exponential increase in execution time is only observed in the class of two-dimensional lattices. Heuristics perform poorly in such topologies, whereas exactness comes at a high price in complexity. In our simulations the polynomial-time ϵ-approximation schemes displayed an extensive execution time and were therefore omitted from the plots. More research is necessary to indicate whether these algorithms might provide a good alternative for exact algorithms in large and hard topologies.

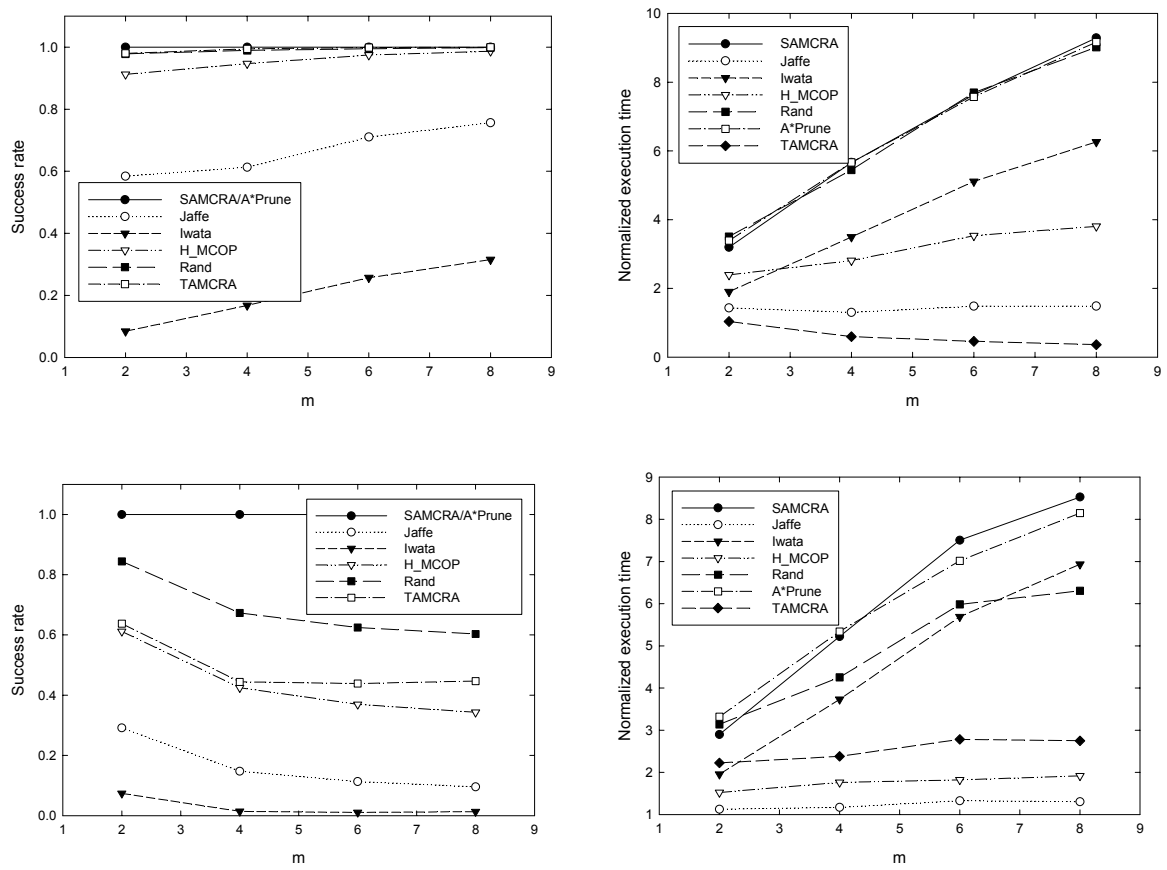


Figure 5.3: The success rate and normalized execution time as a function of m, under strict constraints. The results above are for Waxman graphs, with $N = 100$ and below for the lattices with $N = 49$. In both classes of graphs the link weights were independent uniformly distributed random variables.

Chapter 6

Multicast QoS routing

In the previous chapters we have explicated unicast QoS routing, in which the goal is to find a path between a source and one destination, subject to multiple constraints. Although unicast QoS routing is the prime focus of this thesis, we will devote this chapter to multicast QoS routing. The main problem considered is that of routing from a single source node to a set of p destination nodes, also called multicast source routing or point-to-multipoint routing. The advances in technology and the fast emerging multimedia applications have provided great impetus for new (real-time) multicast applications. A frequently encountered example of a real-time multicast application is video-conferencing. Video-conferencing, as a good representative of the class of real-time multicast applications, requires sufficient bandwidth and, in addition, limits on the maximum delay, jitter and (packet) loss. These requirements, which are considered the same for all members, are referred to as QoS constraints. Many multicast applications will not operate properly if QoS cannot be guaranteed. Hence, future multicast algorithms must be capable of satisfying a set of QoS-constraints. To the best of our knowledge, we [101] were the first to investigate the general problem of multicast QoS routing, with $m \geq 2$ constraints.

A main property of multicast routing is the efficient use of resources [165]. Because each of the p destination nodes will receive the same information, unicast (sending the information p times over each shortest path to each individual multicast participant) is inefficient since, most likely, there will be some overlap among the set of shortest paths. Multicasting as few duplicate packets as possible and only duplicating them if necessary clearly is more efficient. For the case of a single measure, multicast source routing can be implemented by forwarding the packet of a flow or session over the shortest path tree. The more general problem of multipoint-to-multipoint leads to the minimum Steiner tree problem [80].

The main contributions of this chapter are arranged as follows. Section 6.1 will give a formal definition of the main problems of multicast routing with multiple constraints. Since all previous research on multicast routing focuses on finding a multicast tree [143]

and often only considers a fixed number of constraints (for instance only delay and jitter in [141]), Section 6.2 shall discuss multi-constrained multicast trees. In this section, we will demonstrate that finding a tree subject to multiple constraints is not always possible. In Section 6.3 SAMCRA is extended to a multicast QoS algorithm called MAMCRA: Multicast Adaptive Multiple Constraints Routing Algorithm. MAMCRA finds the set of shortest paths to all destinations and then reduces the consumption of resources without violating the QoS constraints. Section 6.4 gives a discussion on multicast routing and poses some suggestions. Section 6.5 gives a small performance evaluation of MAMCRA, after which Section 6.6 concludes this chapter.

6.1 Problem definition

A communication network is modeled as an undirected graph $G = (V, E)$, where V is the set of nodes and E is the set of links. Each link is characterized by a link weight vector $\vec{w}$ consisting of m components w_i, for $i = 1, \ldots, m$. We presume that the full network topology is known at a certain time interval and regard the topology measures as frozen. The m QoS-constraints L_i, for $i = 1, \ldots, m$ are represented by the constraint vector $\vec{L}$. Since all participants receive the same multicast application emitted by the source, the constraint vector $\vec{L}$ is the confining vector for all multicast members of the group.

A multicast sub-graph $G_M = (V_M, E_M) \subseteq (V, E)$ has $p < N$ multicast destination nodes (multicast group members or participants) represented by the set $D = \{d_1, \ldots, d_p\}$. Each of these destination nodes is connected to the source node s, by the links in $E_M \subseteq E$. As will be exemplified below, G_M is best regarded as a set of paths from s to d_j, $j = 1, .., p$, which use the links in E_M.

Problem 34 *Multiple Constrained Multicast (MCM): Given s and D, find G_M such that for each path $P(s, d_j)$ from s to $d_j \in D$, $j = 1, \ldots, p$:*

$$w_i(P(s, d_j)) \leq L_i, \qquad \text{for } i = 1, \ldots, m$$

where $\vec{w}(P)$ is the vector sum of the links that constitute P:

$$w_i(P) = \sum_{e \in P} w_i(e), \qquad \text{for } i = 1, \ldots, m$$

Note that if for a certain $d_j \in D$ no feasible path exists, d_j should be removed from D.

Define $\vec{w}(G_M) = \sum_{1 \leq i \leq |E_M|} \vec{w}(e_i \in E_M)$ and $l(G_M)$ as the length (or vector norm) of $\vec{w}(G_M)$. The length $l(G_M)$ can be any function $f(\vec{w}(G_M))$ on the weight vector of G_M that returns a real number, provided that $f(.)$ is a vector norm (see Section 4.1).

Throughout this chapter we consider $l(G_M) = \max_{i=1,\ldots,m}\left(\frac{w_i(G_M)}{L_i}\right)$, which has been motivated in Section 4.1.

Problem 35 *Multiple Parameter Steiner Tree (MPST): For s and D given, find G_M for which $l(G_M)$ is minimum.*

Problem 36 *Multiple Constrained Minimum Weight Multicast (MCMWM): For s and D given, find G_M such that for each path $P(s,d_j)$ from s to $d_j \in D$, $j = 1,\ldots,p$:*

$$w_i(P(s,d_j)) \leq L_i, \quad \text{for } i = 1,\ldots,m$$

and

$$l(M) \text{ is minimum}$$

Problem 36 is a combination of problems 34 and 35. Section 6.4 will further investigate these three problems. Solving the first problem results in satisfying the QoS constraints. The second problem minimizes the total resource consumption and the third optimizes the resources subject to the QoS constraints. Clearly, the last of the three problems is the most desirable objective for QoS multicast routing.

6.2 Properties of multicast QoS routing

In this section first, problems MCM, MPST and MCMWM are shown to be NP-complete. Subsequently, the MCM and MPST problems are demonstrated to lead to potentially non-compatible solutions.

Theorem 37 *MCM, MPST and MCMWM are NP-complete.*

Proof. MCMWM is a combination of MCM with MPST. Consequently, by proving that MCM is NP-complete, we will also have proved that MCMWM is NP-complete. Let us first consider MCM. For $p = 1$ this problem reduces to unicast QoS-routing, which is proved to be NP-complete for $m \geq 2$ additive parameters ([57], [171]). For $m = 1$, MPST reduces to the minimum Steiner tree problem, which is known to be NP-complete ([93]). ■

We focus on solving the MCMWM problem, which can be considered the hardest of the three problems, since it incorporates two NP-complete problems.

Lemma 38 *The solution to the MPST problem does not necessarily obey the constraints for all multicast members, even if there exist feasible paths towards these members.*

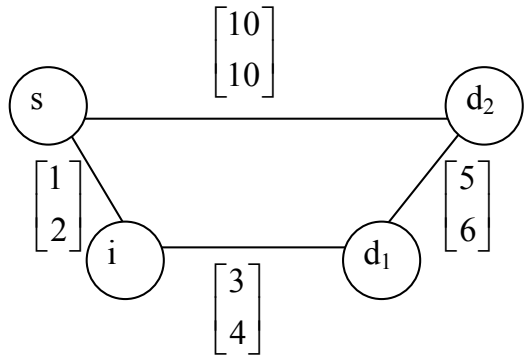


Figure 6.1: Example topology.

Proof. It suffices to prove this lemma by providing an example. Consider the topology in Figure 6.1. Here s is the source node, i is some intermediate node, and d_1 and d_2 are the two destination nodes participating in the multicast session. The MPST connecting s, d_1, d_2 is the tree $s-i-d_1-d_2$ with a total weight vector of $(1,2)+(3,4)+(5,6) = (9,12)$. If the constraints were $(13,13)$, then this would be the best solution to the MCMWM problem, since all the QoS constraints are met with a minimum consumption of resources. However, if the constraints are more stringent, say $(11,11)$ then the path from s to d_2 exceeds these constraints. In this case the multicast tree $[(s-i-d_1),(s-d_2)]$ obeys the requested constraints. This tree has a weight vector of $(1,2)+(3,4)+(10,10) = (14,16)$ and is the second shortest MPST. ■

We have proved that the MPST, although optimal in terms of resource utilization, does not always satisfy the constraints. Since in the example topology of Figure 6.1 the second shortest MPST was the best solution, this may suggest that considering k-shortest MPSTs will lead to the optimal solution for MCMWM. Again, this is not always the case. In order to guarantee QoS, the concept of trees in multiple dimensions cannot be maintained. Only for a single measure, the MCMWM solution is a tree. If the solution to MCMWM would have always been a tree, then k-shortest MPST would have given the exact solution.

Lemma 39 *The solutions to the MCM and MCMWM problems are not necessarily trees.*

Proof. Again, this lemma is proved via an example. The MPST for the topology in Figure 6.2 consists of the links (s,a), (a,c), (c,d_1) and (c,d_2) and has a total weight (vector) of $(1,5)+(1,6)+(1,8)+(10,2) = (13,21)$. The path weight vector for the path from s to d_1 is $(3,19)$ and for s to d_2 is $(12,13)$. The second shortest MPST consists of links (s,b), (b,c), (c,d_1), (c,d_2) and has weight vector $(7,2)+(7,3)+(1,8)+(10,2) =$

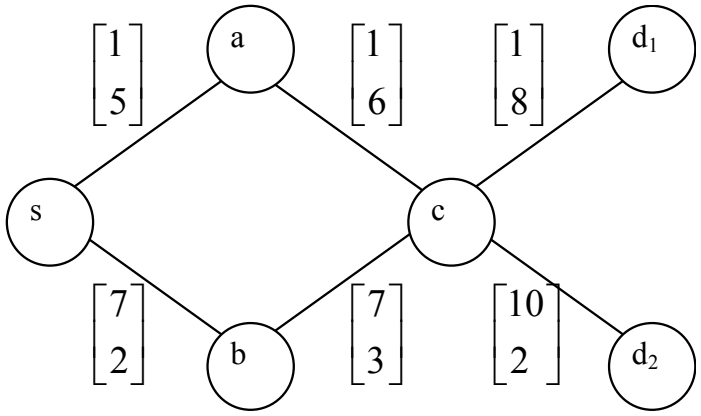


Figure 6.2: Example topology.

$(25, 15)$. The path weight vector is $(15, 13)$ between s and d_1 and $(24, 7)$ between s and d_2. In this example, no other non-dominated trees connecting s, d_1, d_2 exist, i.e. none of the components of the weight vectors of the remaining (two) trees connecting s, d_1, d_2 are smaller than those of the given weight vectors. Therefore, if the above-mentioned trees do not satisfy the constraints, then no tree can satisfy those constraints. If the constraints are $(16, 16)$ then no tree can provide the requested QoS. The only way to obey these constraints is by means of two paths: $s - b - c - d_1$ and $s - a - c - d_2$. In that case the multicast sub-graph G_M is not a tree. ■

Lemma 40 *The solution to MPST is always a tree.*

Proof. If the solution G_M to MPST is not a tree, then it must contain a cycle as depicted in Figure 6.3. The length of the solution G_M equals $l(G_M)$ and according to problem definition 35, $l(G_M)$ must be minimum, i.e.: $l(G_M) \leq l(G'_M), \forall G'_M$.
Without loss of generality, assume that G_M only contains the cycle depicted in Figure 6.3. Further define $G'_M = G_M \backslash \{P_2\}$, i.e. G'_M is a tree. Since all weights are positive:
$l(G_M) = l\left(\sum_{e \in E_M \backslash \{P_1, P_2\}} \vec{w}(e) + \vec{w}(P_1) + \vec{w}(P_2)\right)$
$\geq l\left(\sum_{e \in E_M \backslash \{P_1, P_2\}} \vec{w}(e) + \vec{w}(P_1)\right) = l(G'_M)$.
This leads to a contradiction, because according to problem definition 35, $l(G_M) \leq l(G'_M)$. Therefore, the sub-graph G_M (containing the cycle) cannot be a solution to the MPST problem. ■

We will use SAMCRA (see Section 4.6) as a basis for our multicast QoS routing algorithm. Provided a suitable length is chosen, SAMCRA applies to numerous constraint and/or optimization problems. MAMCRA, the multicast version of SAMCRA, can therefore also be used for different types of constrained-based (optimization) problems.

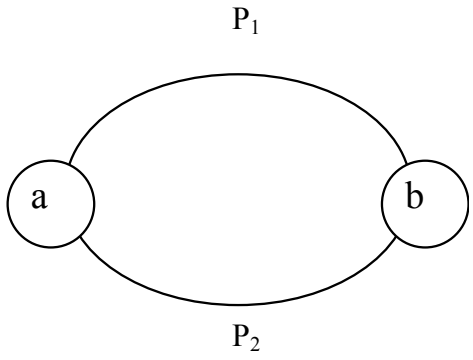


Figure 6.3: A cycle formed by the paths ("branches") P_1 and P_2.

6.3 MAMCRA

This section presents the algorithm MAMCRA, the Multicast Adaptive Multiple Constraints Routing Algorithm. MAMCRA solves the MCM problem exactly and approximates problem MCMWM in the sense that is does not always find the multicast sub-graph G_M with minimum weight (= minimum resource consumption). Although MAMCRA was not designed to solve problem MPST, it may also be considered a heuristic to this problem. The quality of MAMCRA, with respect to problem MCMWM, can be improved at the expense of QoS. The use of MAMCRA in this respect can therefore be considered vendor-specific.

MAMCRA operates as follows:

A First, the set S of shortest paths from s to all p multicast members is calculated via the SAMCRA algorithm.

B Next, the multicast subgraph G_M is optimized, i.e., $l(G_M)$ is reduced without violating the constraints

Step A in the basic MAMCRA operation is readily obtained since it only requires a small modification to SAMCRA. SAMCRA's stop condition is altered, so that it only stops if all destinations (within the constraints) have been reached. Because p destinations are present, also p different lower-bound vectors may exist per node. The look-ahead concept must use the m minimum components of these vectors and therefore reduces in strength. For simplicity we have therefore chosen to omit the look-ahead concept in this chapter. Since S(M)AMCRA operates in a Dijkstra-like manner, during the computation of the set of shortest paths to the multicast members, also shortest

paths to other destinations may be found. One may choose to also include these paths in the set S. Then, if one of these destinations decides to join the multicast session, a compliant path is already present.

When removing the overlap of paths, the set S, which forms G_M, may lead to a tree, but this tree may not be optimal in terms of resource consumption. For instance, consider the example topology of Figure 6.1, with the constraints $(13, 13)$. The set S consists of the paths $s - i - d_1$ and $s - d_2$, which form a tree. The tree $s - i - d_1 - d_2$, however, also obeys the constraints and is more efficient in terms of resource consumption.

Step A is easily completed and provides us with a solution to MCM. Merely the overlap in the set S needs to be removed, so that duplicate packets are only generated when necessary. The elimination of overlap (including the check on min/max constraints) will be addressed below and in Section 6.4.

Step B requires some more effort. Some additional terminology is needed. We define the concatenation of two paths P and Q by PQ, i.e. PQ is the path generated by appending path Q to path P. Note that $\vec{w}(PQ) = \vec{w}(P) + \vec{w}(Q)$.

The symbol $\leq_d$refers to non-dominance, i.e. $\vec{w}(P) \leq_d \vec{w}(Q)$ means that path Q is dominated by path P.

Consider two paths $P_1(s, d_1)$ and $P_2(s, d_2)$ that form a cycle, i.e. both paths have two nodes in common. The first node in common is the source node and the other is node $x \in V \backslash \{s, d_1, d_2\}$. If the two paths have more than two nodes in common, we have a concatenation of cycles with x the (common) node that is most hops away from s.

Property 41 *If $\vec{w}(P_2(s, d_2)) - \vec{w}(P_2(s, x)) + \vec{w}(P_1(s, x)) \leq_d \vec{L}$, then $P_2(s, d_2)$ may be rerouted to $P_1(s, x)P_2(x, d_2)$ without violating the constraints.*

Proof. Let $P_{old} = P_2(s, d_2)$ and $P_{new} = P_1(s, x)P_2(x, d_2)$, with $\vec{w}(P_{old}) \leq_d \vec{L}$
If

$$\vec{w}(P_{old}) - \vec{w}(P_2(s, x)) + \vec{w}(P_1(s, x)) \leq_d \vec{L}$$

and the fact that

$$\vec{w}(P_{old}) - \vec{w}(P_2(s, x)) = \vec{w}(P_2(x, d_2))$$

then

$$\vec{w}(P_2(x, d_2)) + \vec{w}(P_1(s, x)) = \vec{w}(P_{new}) \leq_d \vec{L}$$

■

Property 41 shows that removing cycles, optimizes the total weight vector, since:
$\vec{w}(P_1(s, d_1)) + \vec{w}(P_2(x, d_2)) \leq_d \vec{w}(P_1(s, d_1)) + \vec{w}(P_2(s, d_2))$

For example consider Figure 6.2, where $P_1(s, d_1) = s - b - c - d_1$ and $P_2(s, d_2) = s - a - c - d_2$. The total weight vector of these two paths is $(27, 26)$. Assuming that property 41 is obeyed, rerouting $P_2(s, d_2)$ to $P_1(s, x)P_2(x, d_2)$ leads to a reduction of the total weight vector to $(25, 15)$.

Property 42 *Given a path $P(s,t)$ from s to t within the constraints that uses the sub-path $P(s,a)$, then $P(s,a)$ also lies within the constraints (but is not necessarily the shortest path from s to intermediate node a).*

Property 42 follows from the basic property of a non-linear length in multiple dimensions, namely that sub-paths of shortest paths in multiple dimensions are not necessarily shortest paths.

Part A of MAMCRA's meta-code consisted of a small modification of the SAMCRA algorithm. Based on properties 41 and 42 we can now present part B more detailed:

1. If $S \neq \varnothing$: add the path with most members (d_j) to G_M. (If there are more maximum member paths available, choose the one with smallest length).
2. Else return G_M.
3. If the newly added path forms a cycle in G_M, try to optimize G_M by means of property 41.
4. Check if the new path does not violate the min/max constraints.
5. Remove from S all paths to nodes that are already visited by G_M (property 42).
6. Go to 1.

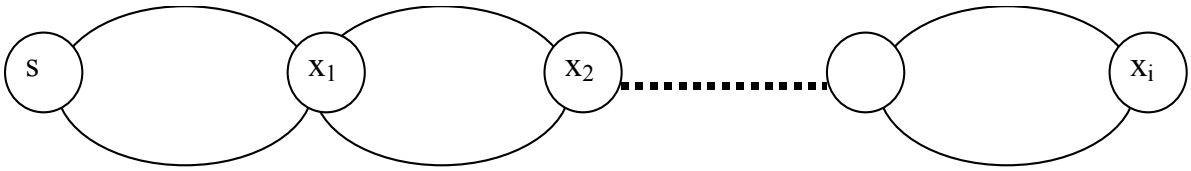


Figure 6.4: Concatenation of cycles.

In part B of MAMCRA, by sequentially adding and optimizing paths, the set S of shortest paths found in part A is lowered in order to obtain a multicast sub-graph G_M that uses as few links from S as possible. In step 1, if the set S is not empty, this means that at least one member is not part of G_M yet. If there are multiple paths in S left, the one that traverses most members is chosen. In case there are multiple

paths with the largest number of members, the path with smallest length is chosen. The selected path is added to G_M. In step 2, the newly added path may form multiple cycles ($< N$) as depicted in Figure 6.4. In that case, MAMCRA first tries to optimize for all cycles, by considering them as being one large cycle[1] $s \rightarrow x_i \rightarrow s$, where i equals the number of cycles. If this is not possible, the procedure is repeated without examining the last cycle, i.e. $s \rightarrow x_{i-1} \rightarrow s$. When a cycle cannot be removed/optimized without violating the constraints, this means that some overlap may be introduced that cannot be removed, i.e. G_M is not a tree and therefore some link(s) may see duplicate packets. When considering a min/max constraint on bandwidth, this means that the link has to be able to provide more bandwidth than the bandwidth consumption of the source, i.e. the capacity of the link must be equal or larger than r times the bandwidth constraint, where r equals the number of replicated packets on the link. Therefore, if a tree cannot be formed, an additional check on the min/max constraints is required. This check is made in step 3. This procedure is repeated until S is empty, which means that G_M contains all feasible members.

The worst-case complexity of MAMCRA is $O(kN\log(kN) + k^2mM)$ for part A and $O(Np^2)$ for part B.

An example of the operation of MAMCRA will further illustrate these steps.

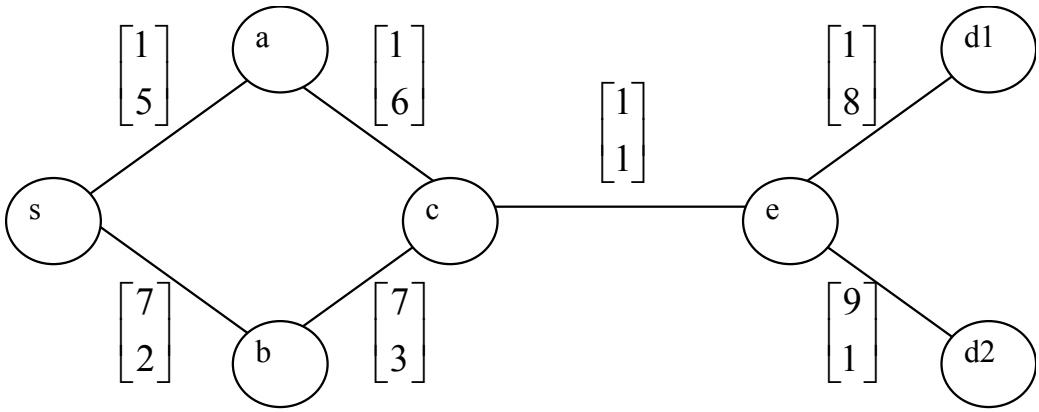


Figure 6.5: Example topology.

The steps taken by MAMCRA in part A, given the topology of Figure 6.5, are put in Table 6.1. During initialization, the source node is set to $(0, 0)$. In the next step s scans its neighbors and finds paths to a and b, with path vectors respectively $(1, 5)$ and $(7, 2)$. We also keep track of the previous node. If we use the length function (4.3), vector $(1, 5)$ is the smallest entry and therefore a is extracted next and the scanning procedure is

[1]Since the weight vector of a concatenation of cycles equals the sum of the individual weight vectors of those cycles, this assumption is justified, with only a slight abuse of the definition of a cycle.

	s	a	b	c	e	d_1	d_2
0	$(0,0)$						
1		$s(1,5)$	$s(7,2)$				
2				$a(2,11)$			
3				$b(14,5)$			
4					$c(3,12)$		
5						$e(4,20)$	$e(12,13)$
6							d_2
7					$c(15,6)$		
8						$e(16,14)$	$e(24,7)$
9							d_1

Table 6.1: Activity table of MAMCRA when executed on the topology in Figure 6.5.

repeated. This process of extracting and scanning is continued until both destinations have been extracted once (i.e., a shortest path has been found). By back-tracing the paths from d_1, d_2 to s we receive the set $S = \{(s-b-c-e-d_1), (s-a-c-e-d_2)\}$.

So far, the constraints have not yet been taken into account. (This is an exception, used solely to simplify this example and because we choose the constraints identical this does not alter the true operation. Normally, part A also includes the constraints).

Now we optimize S (part B) to gain G_M, using two scenarios. First given the constraints $(20, 20)$:

1. We add $(s-a-c-e-d_2)$ to G_M, because this path is shortest in length where all paths have the same amount of members (only one).

2. No cycle was formed, so we add $(s-b-c-e-d_1)$ to G_M.

3. $(s-b-c-e-d_1)$ creates a cycle $(s-a-c-b-s)$ in G_M. When property 41 is applied, we see that $\vec{w}(s-b-c-e-d_1) - \vec{w}(s-b-c) + \vec{w}(s-a-c) = (16,14) - (14,5) + (2,11) = (4,20) \leq_d (20,20)$. We therefore reroute $(s-b-c-e-d_1)$ to $(s-a-c-e-d_1)$.

The result is $G_M = \{(s-a-c-e-d_1), (s-a-c-e-d_2)\}$, which can be written as a tree $G_M = \{(s-a-c-e), (e-d_1), (e-d_2)\}$ if we remove the overlap.

If the constraints are $(16, 16)$, we cannot optimize S and therefore $S = G_M = \{(s-a-c-e-d_2), (s-b-c-e-d_1)\}$.

Part B constructs the multicast sub-graph by sequentially adding paths. This approach allows MAMCRA to add new members to an existing "tree." Part A first calculates the paths to the new members, after which part B sequentially and efficiently

adds them to the existing sub-graph. However, re-computing the entire multicast sub-graph is hardly more intensive and may therefore be preferred. The choice of how to add/remove members is part of the QoS multicast protocol.

6.4 Discussion of multicast QoS routing

6.4.1 Tuning MAMCRA

MAMCRA gives an efficient, but not always optimal, solution to MCMWM. It is possible to further optimize MAMCRA by considering not only the shortest paths to d_j ($j = 1, \ldots, p$), but by storing all $k_{requested}$-shortest paths (within the constraints) from s to d_j in S. The cost of optimizing MAMCRA in this way lies in complexity (running time). Each node now has a queue-size $k_{requested} \leq k \leq k_{\max}$. Although this approach does not alter the worst-case complexity of part A (after all, MAMCRA already works with k-shortest paths, the only difference now is that we examine at least $k_{requested}$ shortest paths instead of as few as possible), it will have an effect on the running time, which will increase proportional to k^2. The worst-case complexity of part B becomes $O(kNp^2)$. However, the larger we choose k, the higher the probability of finding the exact solution to MCMWM. Therefore k is considered to be a tuning parameter. Note that since part B is heuristic, this approach will never be able to guarantee that the exact solution to MCMWM is always found. As mentioned previously, we have chosen in this chapter to solve the MCMWM problem with MAMCRA. If other problems are considered more relevant, MAMCRA should be adapted accordingly to solve such problems. For instance, if the problem is to find a tree that guarantees QoS to all (or if this is not possible, to a subset) of the multicast members, then part B of MAMCRA should always remove cycles.

6.4.2 QoS negotiation

Guaranteeing QoS and optimizing resource utilization can be two conflicting interests. Depending on the wishes of the client (multicast member), a trade-off can be made between QoS and resource utilization. This trade-off will be based on monetary cost, since guaranteeing a high level of QoS will inflict a large consumption of resources, which has to be paid for. It is not likely that all members are willing to pay the same price. It would therefore be beneficial if some sort of negotiation between QoS and price could take place. For instance, MAMCRA (in step A) computes the set of shortest paths based on the maximum allowable constraints ($\vec{L}_{\max}$). In the optimization phase, the different wishes ($\vec{L}_{d_j} \leq_d \vec{L}_{\max}$) of the members can be taken into account. For instance, if a certain level of QoS (within the $\vec{L}_{\max}$ constraints) has a high price and another slightly worse level of QoS (also within the $\vec{L}_{\max}$ constraints) has a lower price, the

client may negotiate his requested level of QoS. MAMCRA only examines the solutions within the constraints and hence we can apply any length-function, provided it obeys the criteria for length (see Section 4.1). Because price is often considered the most important parameter to minimize, we can take $l(P) = w_i(P)$, where w_i corresponds to the price measure (see Section 4.1).

6.4.3 QoS multicast protocol

In the first scenario of our example (Figure 6.5), both paths use $s - a - c - e$ and a packet needs only to be sent once over this sub-path, after which it is duplicated at node e and sent to d_1 and d_2. In the second scenario the paths have an overlap on $c - e$. Since the packets at the source are duplicated and forwarded to a and b, duplicate packets arrive at node c. These duplicate packets have traveled different paths towards c and have different weights. A packet may arrive later at c than its duplicate counterpart, but its price/loss/jitter may be less. Since the paths from c to the destinations also have different weights, both packets must be kept. Only allowing one packet on link $c - e$ would result in a violation of the constraints at one of the destinations. As argued earlier, in case of overlap, we should check if the min/max constraints are still guaranteed or perhaps renegotiate the constraints. The task of efficiently forwarding/replicating packets is part of the multicast protocol in use and not of MAMCRA. Several traditional multicast protocols exist, like DVMRP [168], MOSPF [119] and PIM [46]. These protocols were designed for best-effort traffic. The protocols in [23], [47] and [28] are better suited for delivering QoS. However, these and other protocols only consider multicast trees. MAMCRA, if allowing non-trees, therefore either requires a new or modified multicast protocol. This protocol has to cope with different dynamics, e.g. network dynamics or the joining/leaving of multicast members. It must ensure stability of the multicast "tree," but it must also (efficiently) guarantee QoS, which may be conflicting targets. Since providing QoS is the goal, also some type of resource reservation is desirable.

It has been a goal of this chapter to address the difficulties in providing guaranteed multicast QoS. We have seen that the constraints, imposed by guaranteed QoS, introduce numerous difficulties mainly related to the possible overlap (caused by the absence of a tree) and hence an objective of a network provider should be to always strive towards a multicast tree.

6.4.4 QoS multicast in an active network

Inspired by Connectionless Multicast (CLM, e.g. [150]) we touch upon DiffServ multicast and its exact active counterpart. In CLM, the packet header carries the IP addresses of all the multicast members. Each router determines the next hop for each destination

and constructs a new header for every distinct hop. The new header only contains destinations for which the next hop is on the shortest path. In conformance with unicast DiffServ, we can extend CLM, such that each packet belongs to a certain Class of Service (CoS) and each router has a routing table for each CoS. We have proved in [163] that hop-by-hop destination-based QoS routing can only be guaranteed in an active network. If we store the history of an active packet in its header, then for each packet arriving at a router, MAMCRA is used to compute the best forwarding/replication strategy. The best use for such an active strategy is in highly dynamic (for example wireless) environments, since we do not need (to recalculate) routing tables. However, we do need to have an accurate view of the network.

6.5 Performance evaluation of MAMCRA

Finally, this section will present a small performance evaluation of MAMCRA. At the time that MAMCRA was proposed [101] it was the only multicast QoS algorithm that could handle an arbitrary number (m) of constraints. Later Tsai and Chen [159] also proposed two multicast QoS algorithms (M_MCOP and M_QDMR) for $m \geq 2$ and compared them against MAMCRA. However, they considered different objectives as MAMCRA leading to biased simulations and erroneous conclusions. Together with Tsai we have repeated the simulations in a correct and fair manner.

We have simulated on the class of Waxman graphs (400 iterations) with $N = 101$ nodes. We have simulated with $m = 4$ QoS measures and one unconstrained monetary cost. The link weights of all (in total five) QoS measures were independent uniformly distributed in the range [1, 141]. The source s and the p destination nodes were randomly assigned. The four constraints were uniformly chosen with the assurance that for each path from the source to any of the p destinations a feasible path existed. Besides the algorithms M_MCOP, M_QDMR proposed by Tsai and Chen [159], we have simulated with two different versions of MAMCRA, namely MAMCRA as defined in this chapter with the non-linear length (4.3) and MAMCRA-cost with a semi-linear length, which only optimizes on the cost-measure, provided that the four constraints are met. The success rate is defined as the number of times that the constraints were met for all p destinations. Figure 6.6 gives the success rate as a function of p. MAMCRA always obtains a success rate of 100% if feasible paths exist to all destinations. The performance of M_MCOP and M_QDMR on the other hand decreases with the number of destinations. Moreover, if the constraints get stricter (not shown), the difference between MAMCRA and M_MCOP, M_QDMR increases. Figures 6.7 and 6.8 represent the cost of the multicast subgraphs in terms of the cost measure and the link overhead, respectively. Figure 6.7 gives the cost ratio, which is defined as the cost of the multicast subgraph ($c(G_M) = \sum_{e \in G_M} c(e)$) divided by the combined cost of the p unicast shortest paths computed via the Dijkstra algorithm. We have included in

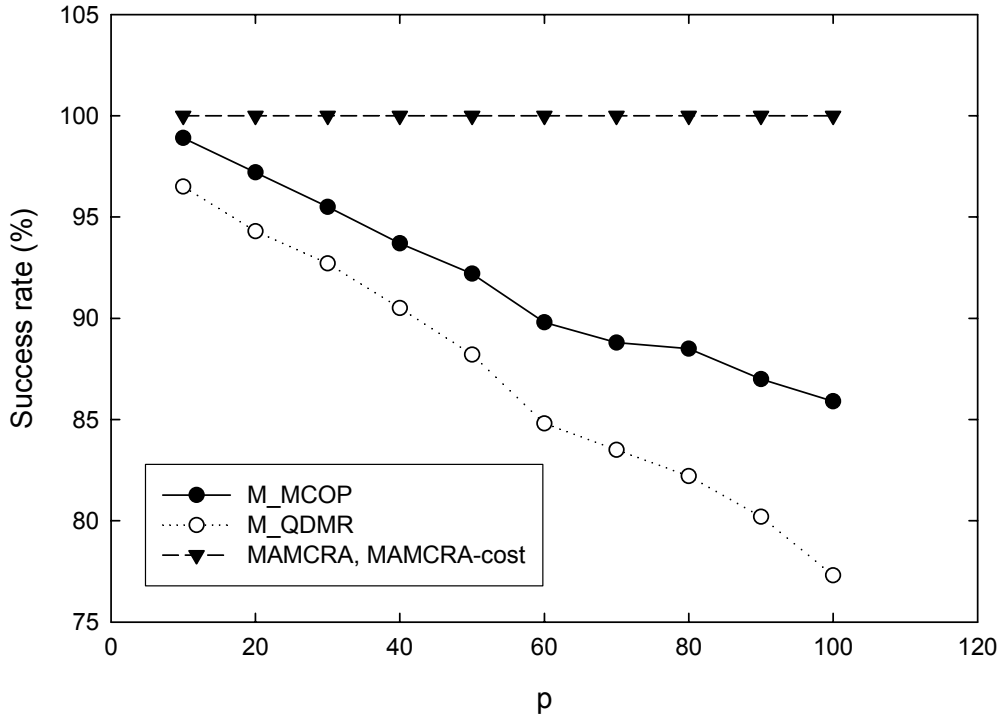


Figure 6.6: The success rate as a function of the number of destinations p.

the plot a theoretical law for the efficiency of multicast [165]. This law holds for the class of random graphs and can be considered a good indicator of the efficiency that can be obtained with multicast in the one-dimensional (unconstrained) case. Figure 6.7 shows that MAMCRA-cost performs best and therefore indicates that the choice of the length function is important and should be carefully tailored to the optimization problem under consideration.

Contrary to M_MCOP and M_QDMR, MAMCRA does not necessarily return a tree. To evaluate how many "extra" links the graph G_M contains, Figure 6.8 shows the link overhead defined as $\frac{G_M(M)}{G_M(N)-1}$, where $G_M(M)$ is the total number of links in the G_M and $G_M(N)$ is the total number of nodes in G_M. We can see that the link overhead remains small, suggesting that MAMCRA provides efficient solutions.

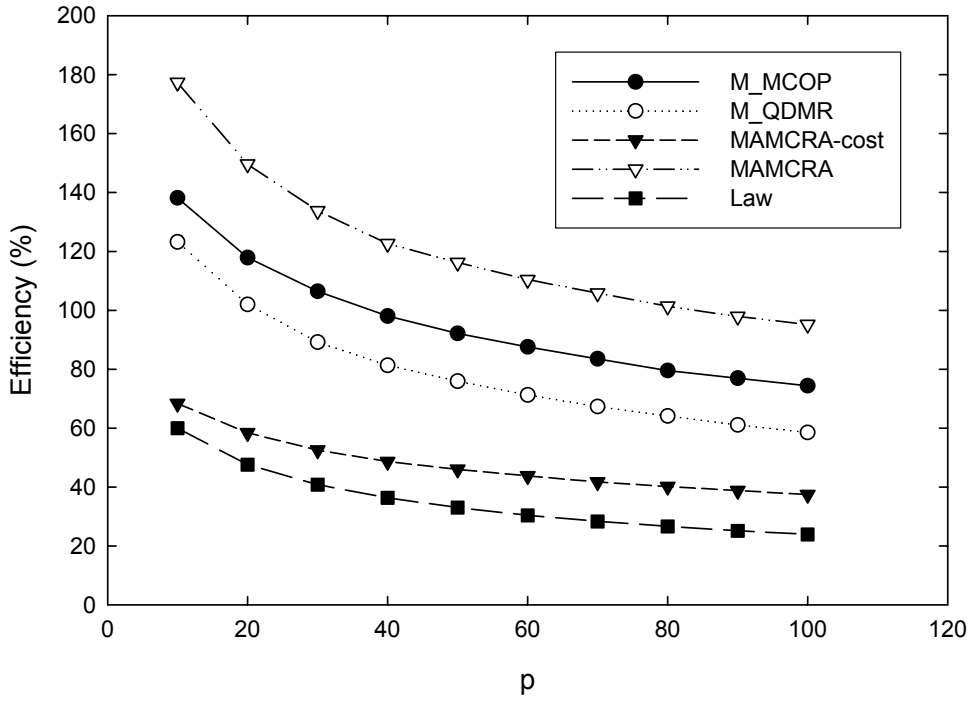


Figure 6.7: The cost-efficiency of multicast as a function of the number of destinations p.

6.6 Conclusions

This chapter has shown that a multicast tree may not always guarantee the requested QoS constraints, while multiple unicast QoS sessions can. This property increases the complexity of constrained multicast routing (besides the proven NP-completeness). A trade-off between efficient use of resources and QoS has to be made, which resulted in the proposed algorithm MAMCRA. MAMCRA computes the set S of shortest paths from source s to all the destination nodes, and then reduces this set to an efficient set of multicast routes, without compromising the requested level of QoS. Simulations with MAMCRA indicate that it often returns trees or sub-graphs that closely resemble a tree. It was shown that it is desirable to always construct (or strive for) a multicast tree, either by fine-tuning MAMCRA or by renegotiating the constraints.

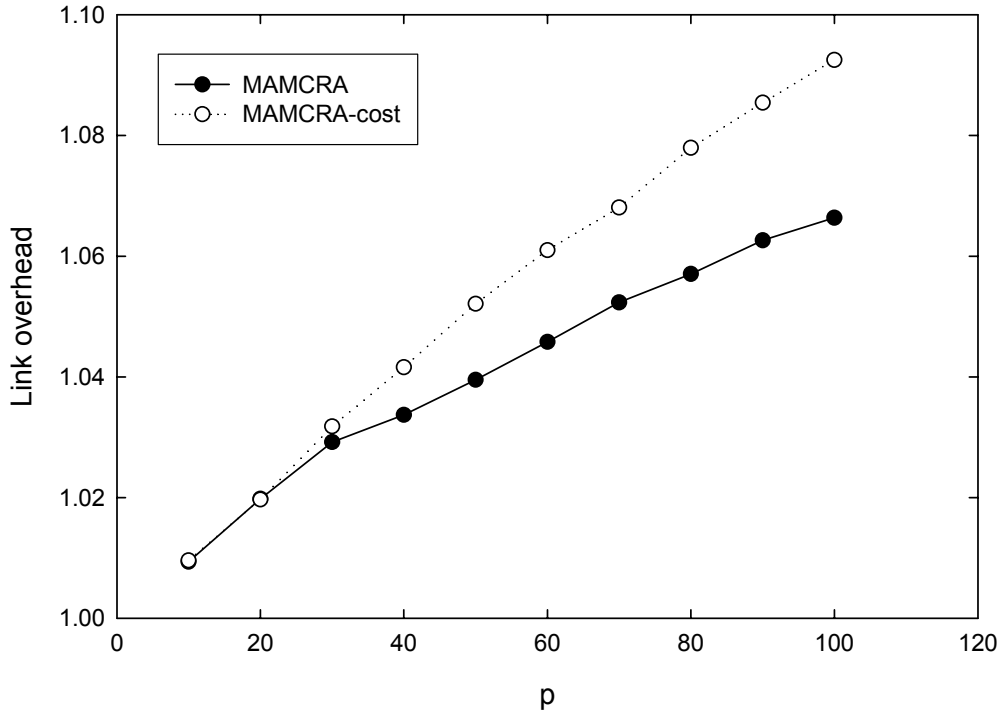


Figure 6.8: The link overhead as a function of the number of destinations p.

Chapter 7

Link-disjoint QoS routing

The problem of finding disjoint paths in a network has been given much attention in the literature due to its theoretical as well as practical significance to many applications, such as layout design of integrated circuits, survivable design of telecommunication networks and restorable/reliable routing. Paths between a given pair of source and destination nodes in a network are called link disjoint if they have no common (i.e., overlapping) links, and node disjoint if, besides the source and destination nodes, they have no common nodes. With the development of optical networks and the deployment of MPLS or GMPLS [10] networks, the problem of finding disjoint paths is receiving renewed interest as fast restoration after a network failure is crucial in such kind of networks. In robust communication networks, a connection usually consists of two link- or node-disjoint paths: one active path and one backup path. A service flow will be redirected to the backup path if the active path fails. Load balancing, another important aspect for communication networks to avoid network congestion and to optimize network throughput, also requires disjoint paths to distribute flows. Robustness and load balancing are, among others, both aspects of Quality of Service (QoS) routing.

In this chapter the focus lies on finding QoS-aware link-disjoint paths. In general a link-disjoint paths algorithm can be extended to a node-disjoint algorithm with the concept of node splitting, i.e. replacing one node with two nodes that are linked together via a link with zero-valued weights [155].

7.1 Problem definition

In the context of finding link-disjoint paths, a path P between a source s and destination t is considered to be composed out of an ordered set of links. If a path P_1 is link-disjoint with a path P_2, there is no common link element in the link sets representing each path, and $P_1 \cap P_2 = \varnothing$, else $P_1 \cap P_2 \neq \varnothing$.

For $m = 1$, when no constraint is required (see the LPP problem defined below), the

linear length of a path is computed as $l(P) = \sum_{(u,v)\in P} w_i(u,v)$. For $m > 1$, the non-linear length (4.3) is used: $l(P) = \max_{1\leq i\leq m} \left[\frac{w_i(P)}{L_i}\right]$, where $w_i(P) = \sum_{(u,v)\in P} w_i(u,v)$.

If a path P_1 is link-disjoint with a path P_2, then $l(P_1 \cup P_2) = l(P_1) + l(P_2)$ for $m = 1$, but for $m > 1$, $l(P_1 \cup P_2) \leq l(P_1) + l(P_2)$. The goal is to find two link-disjoint paths that both obey multiple constraints. The total length of two paths is defined as

$$l(P_1) + l(P_2) \tag{7.1}$$

for $m \geq 1$.

Problem 43 *Link-disjoint Path Pair (LPP) Problem. Given a directed graph $G = (V, E)$ with one weight per link ($m = 1$), and a source-destination pair (s, t). Find a set of two paths P_1 and P_2, such that $P_1 \cap P_2 = \varnothing$ and the total length $l(P_1) + l(P_2)$ is minimized.*

The LPP problem can be solved in polynomial time [19], [154], [155].

Problem 44 *Multi-Constrained Link-disjoint Path Pair (MCLPP) Problem. Given a directed graph $G = (V, E)$ with $m > 1$ weights per link, a constraint vector $\vec{L}$ and a source-destination pair (s, t). Find a pair of link-disjoint paths P_1 and P_2, such that $P_1 \cap P_2 = \varnothing$ and both paths obey the constraint vector $\vec{L}$.*

Theorem 45 *The MCLPP problem is NP-complete.*

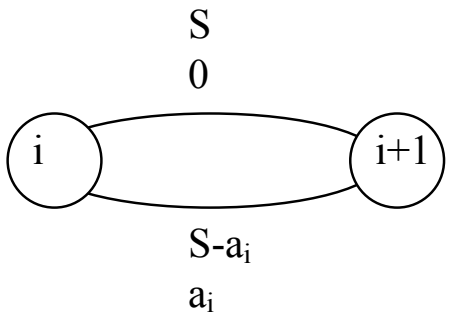


Figure 7.1: The assignment of link weights between nodes i and $i + 1$ in the chain topology.

Proof. Given a chain topology with $n + 1$ nodes and $2n$ links, each with a two-component weight vector as depicted in Figure 7.1 and a set of numbers $a_i \in A$, $0 \leq a_i \leq S$, for $i = 1, ..., n$, where $S = \sum_{i=1}^{n} a_i$. The constraints are chosen as follows:

$L_1 = nS - \frac{S}{2}$ and $L_2 = \frac{S}{2}$.
To solve the MCLPP problem, we need to find two paths P and P' from node 1 to node $n+1$ that obey the constraints. Since, for all link weight vectors, the sum of the components equals S, we have that $w_1(P) + w_2(P) = nS$ and $w_1(P') + w_2(P') = nS$. Accordingly, a solution satisfying the constraints is only found if $w_1(P \text{ and } P') = nS - \frac{S}{2}$ and $w_2(P \text{ and } P') = \frac{S}{2}$. The problem has now become an instance of the well-known NP-complete partition problem [57] and can only be solved by finding the set $A' \subseteq A$, for which $\sum_{a_i \in A'} a_i = \frac{S}{2}$. A feasible path P exists if the set A' exists. A feasible path P consists of the lower link if $a_i \in A'$ and the upper link if $a_i \notin A'$. The path P' then follows the remaining links. ■

In this chapter we focus on solving the MCLPP problem. Related work on finding disjoint paths in one dimension between a source and a destination will be reviewed in Section 7.2 and the simple link-disjoint algorithm LBA will be explained in Section 7.3. In Section 7.4 an extension of LBA to multiple dimensions is discussed and shown to be difficult. Therefore, a heuristic algorithm DIMCRA for solving the MCLPP problem is proposed in Section 7.5. Section 7.6 presents the conclusions.

7.2 Related work

7.2.1 Link-disjoint paths in one dimension

An intuitive method to determine two shortest link-disjoint paths between a pair of source and destination nodes consists of two steps. The first step retrieves the shortest path between a given pair of nodes in a graph. The second step is to prune all the links of that path from the graph and to find the shortest path in the reduced graph. We will refer to this method as the Remove-Find (RF) method. Although the RF method is direct and simple, it has at least two disadvantages: (a) provided that two link-disjoint paths exist, there is no guarantee that they will be found and (b) the second link-disjoint path may have a significantly larger length than the first shortest path.

To surmount the disadvantages of the RF method, other methods have been devised to find a pair of shortest link-disjoint paths with minimal total length [19], [24], [29], [123], [151], [154], [155], [174]. In [154], Suurballe proposed an algorithm, referred to as Suurballe's algorithm, to find K node-disjoint paths with minimal total length using path augmentation. The path augmentation method was originally used to increase the size of a matching with an augmenting path [39] and to find a maximum flow or a minimum cost flow in a network [2], [128]. The problem to find link/node disjoint paths can be viewed as a special case of the minimum cost flow problem as demonstrated in [19], [154], [155]. The basic idea of Suurballe's algorithm is to construct a solution set of two disjoint paths based on the shortest path and a shortest augmenting path. K disjoint paths can be obtained by augmenting the $K-1$ optimal disjoint paths with

this algorithm. In 1984, Suurballe and Tarjan [155] improved Suurballe's algorithm, such that pairs of link-disjoint paths from one source node to n destination nodes could be efficiently obtained in a single Dijkstra-like computation. This algorithm is referred to as the S-T algorithm. To find n pairs of disjoint paths, the S-T algorithm requires $O(M \log_{(1+M/n)} n)$ time and Suurballe's algorithm $O(n^2 \log n)$, where n is the number of destination nodes and M is the number of links. Kar *et al.* [92] and Kodialam and Lakshman [94], [95] incorporated the S-T algorithm into their algorithms to find a pair of link-disjoint paths serving as active and backup paths for routing bandwidth guaranteed connections. Liang [108] extended the S-T algorithm to find two link-disjoint paths between a pair of nodes, while optimizing both network load and routing costs.

In 1994, Bhandari [19] proposed an algorithm to find a pair of span-disjoint[1] paths between two nodes in optical-fiber networks. The disjoint paths algorithm used by Bhandari is a simplified version of Suurballe's algorithm [154] that requires a special link weight transformation to facilitate the use of Dijkstra's algorithm. Shaikh [148] made an extension to Bhandari's algorithm [19] to solve the span-disjoint paths problem in more complicated structured optical networks.

Li *et al.*, [107] (and Sen *et al.*, [146]) proved that the LPP problem will be NP-complete if the total length of the two disjoint paths is defined as $\max(l(P_1), l(P_2))$, instead of $l(P_1) + l(P_2)$. Ho and Mouftah [78] proposed another objective function $\alpha \cdot l(P_1) + l(P_2)$, where P_1 and P_2 are the active path and the backup path, respectively. The parameter α can be set large for a shared protection scheme ($1 : p$ or $q : p$) and could be as small as unity for a dedicated protection scheme ($1 : 1$).

Heuristic algorithms based on matrix calculation [157] or recursive matrix-calculation [124] to solve the K-shortest link-disjoint paths problem with a bounded hop count have been proposed as well. Even some algorithms were proposed for finding K-best paths, i.e. K disjoint or maximally disjoint paths with smallest total length between a pair of nodes in a trellis graph [24], [174]. An optimal algorithm for finding K-best paths, without hop count limitation, between a pair of nodes is given by Lee and Wu in [103], where they transfer the K-best paths problem into a maximum network flow and minimum cost network flow algorithm via some modifications to the original graph. Distributed algorithms for the link/node-disjoint paths algorithms can be found in [29], [123], [151].

7.2.2 Disjoint paths in multiple dimensions

To the best of our knowledge we [68] were the first to consider the MCLPP problem. Recently some papers on disjoint paths in QoS routing have emerged. However, they only considered bandwidth and/or delay as their QoS measures [15], [95], [83], [65], [110]. The maximally disjoint shortest and widest paths (MADSWIP) algorithm from

[1] A span is the set of all transmission links between two nodes that share the same facilities structure (e.g., cable, duct, trench). In the case that a physical cut would take place, all transmission links would fail.

Taft-Plotkin, *et al.* [156], involves a modified version of the S-T algorithm to find a pair of disjoint paths. MADSWIP can produce a pair of widest or shortest maximally link-disjoint paths from a source node to all other nodes. Moreover, it tries to find two paths simultaneously to satisfy the maximal link-disjointness to each other. However the only QoS measures used by MADSWIP are bandwidth and delay. After our work [68], some new papers emerged that look at multiple QoS measures, e.g. the paper of Orda and Sprintson [127].

7.3 Path augmentation for solving LPP

This section shall present a simplified variant of Bhandari's Algorithm [19], referred to as LBA (Link-disjoint version of Bhandari's Algorithm), which can exactly solve the LPP problem. First, the basic steps and the fundamental concepts of LBA will be explained. Subsequently, the optimality of LBA is proved and LBA is shown to be loop-free.

7.3.1 The steps of LBA

Bhandari's algorithm [19] has been designed to find a pair of span-disjoint paths in an optical network. We have modified Bhandari's algorithm into a link-disjoint path pair algorithm LBA by omitting the node-splitting operation that ensures the node-disjointness and the graph transformations that ensure span-disjointness.

Before explaining the operation of LBA, first some notations are introduced. If we reverse the direction and the sign of the link weights of each link on the path P_1 between s and t, i.e. $w(v,u) = -w(u,v), \forall (u,v) \in P_1$, then we will have a path directed from t to s, denoted by $-P_1$, which consists of the reversed P_1 links. We define[2] $l(-P_1) = -l(P_1)$. A set consisting of these P_1 links whose reversed links also appear on P_2 and vice versa, is denoted as $P_1 \tilde{\cap} P_2 = \{(u,v) \text{ and } (v,u) | (u,v) \in P_1 \text{ and } (v,u) \in P_2\}$. In all the figures of this chapter, bold lines represent links on the shortest path(s) in a graph or its corresponding modified graph, dashed lines represent reversed links that do not exist in the original graph and bold dashed lines represent such reversed links that appear on the shortest path.

Given a directed graph $G = (V, E)$ and a source-destination pair (s, t), LBA executes the following steps:

1. Find the shortest[3] path P_1 from node s to node t.

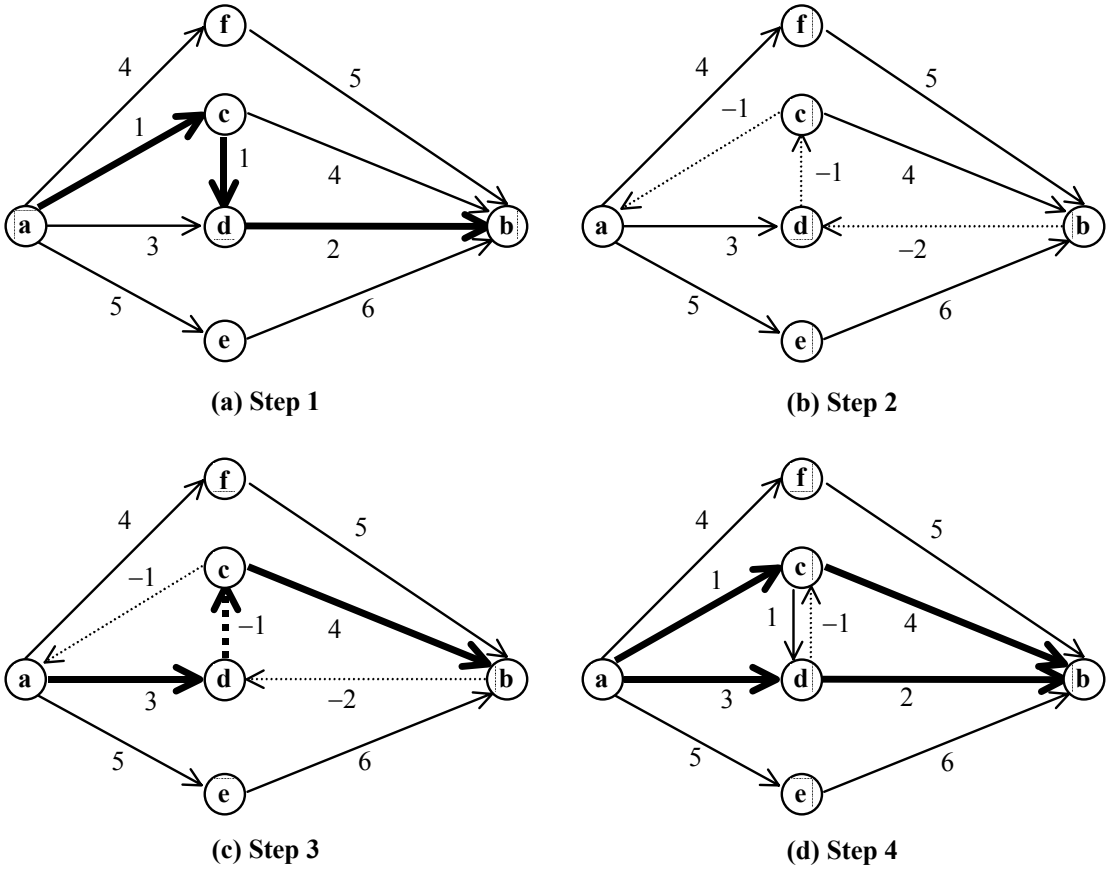
[2] With the definition of length in (4.3), we have $l(-P_1) = -l(P_1)$ only for $m = 1$.

[3] If there exist multiple equi-length shortest paths in the original graph or in the modified graph, either one of them can be chosen. Choosing different shortest paths may lead to different solution sets. However, these solution sets will have the same minimum total length.

2. Replace P_1 with $-P_1$, a modified graph $G' = (V, E')$ is created.

3. Find a shortest path P_2 from node s to node t in the modified graph G'; if P_2 does not exist, then stop.

4. Take the union of P_1 and P_2, remove from the union the set of links consisting of those P_1 links whose reversed links appear on P_2, and vice versa; then group the remaining links into two paths P_1' and P_2', i.e. $P_1' \cup P_2' = (P_1 \cup P_2) \setminus (P_1 \tilde{\cap} P_2)$.



(a) Step 1 **(b) Step 2**

(c) Step 3 **(d) Step 4**

Figure 7.2: Example of the operation of LBA.

The steps of LBA are best explained through the example in Figure 7.2. The problem is to find two link-disjoint paths between a and b. In Step 1, the shortest path from a to b is found as $P_1 = a - c - d - b$, with length 4. In Step 2, a modified graph $G' = (V, E')$ is created by reversing the direction and the sign of the weights of each

link on P_1. For instance, link (c, d) with weight 1 is replaced by link (d, c) with weight –1. In Step 3, the shortest path $P_2 = a - d - c - b$ in the modified graph has length 6. In Step 4, $P_1 \tilde{\cap} P_2 = \{(c, d), (d, c)\}$ is removed from the union $P_1 \cup P_2$. The solution set of disjoint paths $\{P_1', P_2'\} = \{a - c - b, a - d - b\}$ is obtained. The total length of these paths equals $5 + 5 = 10$, which is exactly the minimal total length of two link-disjoint paths in this graph.

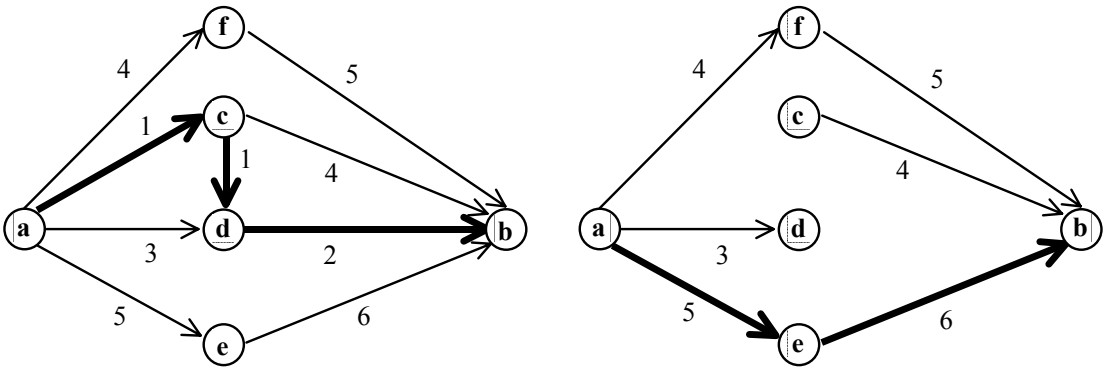


Figure 7.3: Example 1 of the operation of RF.

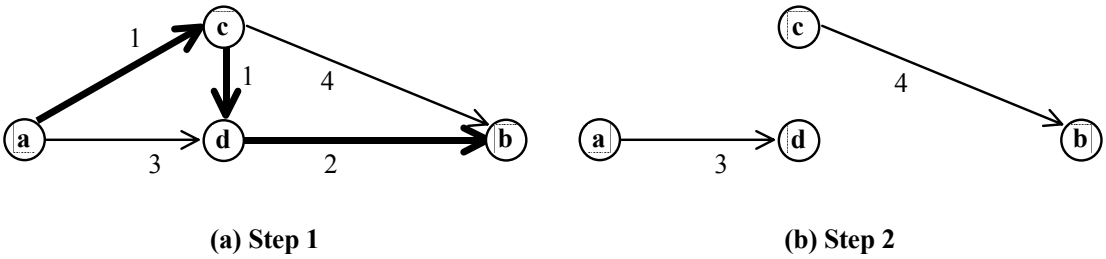


Figure 7.4: Example 2 of the operation of RF.

For comparison, in Figure 7.3, the RF method is applied on the same topology with the same requirements. In Step 1, the shortest path $a - c - d - b$ is retrieved. In Step 2, a modified graph is created by removing all the links on $a - c - d - b$. The shortest path in the modified graph is $a - e - b$ with length 11. Thus the set $\{a - c - d - b, a - e - b\}$ has a total length of $4 + 11 = 15$, which is longer than 10 as found with LBA. This example

illustrates that the RF method cannot guarantee to find the optimal solution. More importantly, in the graph shown in Figure 7.4(a), although there exist two link-disjoint paths between a and b, RF cannot find the second path in Step 2 as shown in Figure 7.4(b). LBA, on the other hand, still returns the optimal set in this case.

7.3.2 LBA is based on the shortest path

The goal of this subsection is to clarify why the optimal solution set of LBA, as well as other path augmentation algorithms [19], [110], [123], is based on the shortest path. Although the theory presented here can be derived from the theory of min-cost flows [51], [128], it is instructive to give an overview.

We will first show that the optimal set for the LPP problem is based on the shortest path. The next is to show that the optimal set of two link-disjoint paths has the smallest difference in length from the shortest path, among all the possible sets of link-disjoint paths. Finally, the logical difference set (defined below) is demonstrated to compose a path.

Given a directed graph $G = (V, E)$ and a pair of source-destination nodes (s, t), the relation between a set of two link-disjoint paths $\{P_{d1}, P_{d2}\}$ and the shortest path P_1 belongs to one of the following types:

1. P_1 itself equals P_{d1} or P_{d2}, i.e. $P_1 = P_{d1}$ or $P_1 = P_{d2}$.
2. P_1 overlaps with both paths P_{d1} and P_{d2}, i.e. $P_1 \cap P_{d1} \neq \varnothing,\ P_1 \neq P_{d1}$ and $P_1 \cap P_{d2} \neq \varnothing,\ P_1 \neq P_{d2}$.
3. P_1 only overlaps with one path in the set $\{P_{d1}, P_{d2}\}$, but not with the other, i.e. $P_1 \cap P_{d1} \neq \varnothing,\ P_1 \neq P_{d1}$ and $P_1 \cap P_{d2} = \varnothing$ (or $P_1 \cap P_{d2} \neq \varnothing,\ P_1 \neq P_{d2}$ and $P_1 \cap P_{d1} = \varnothing$).
4. P_1 is link-disjoint with both paths in $\{P_{d1}, P_{d2}\}$, i.e. $P_1 \cap (P_{d1} \cup P_{d2}) = \varnothing$.

Lemma 46 *Given a directed graph $G = (V, E)$ and a source-destination pair (s, t), if the optimal set $\{P_1', P_2'\}$ of LPP exists, $P_1' \cup P_2'$ must contain either the first shortest path P_1 itself or some P_1 links on each of its two paths.*

Proof. If $P_1' \cup P_2'$ is of type 4, then each path in $\{P_1', P_2'\}$ is link-disjoint with P_1. As P_1 is the shortest path, both $\{P_1, P_1'\}$ and $\{P_1, P_2'\}$ have a total length that is shorter than the total length of $\{P_1', P_2'\}$. Hence, the optimal set $\{P_1', P_2'\}$ cannot be of type 4 and must contain some or all P_1 links.
If $P_1' \cup P_2'$ is of type 3, only one path in $P_1' \cup P_2'$ contains some P_1 links. Without loss of generality, suppose P_1' contains some P_1 links, and the other path P_2' is link-disjoint with P_1, then $\{P_1, P_2'\}$ is a set which is shorter than $\{P_1', P_2'\}$. Hence the optimal set $\{P_1', P_2'\}$ cannot be of type 3.
Therefore, if the optimal set exists, it must be either of type 1 or 2. ■

Lemma 47 *The optimal set has the smallest difference in length*

$$Y = l(P_1') + l(P_2') - l(P_1) \geq 0 \tag{7.2}$$

with the shortest path P_1, among all the possible sets of link-disjoint path pairs.

Proof. In the set $P_1' \cup P_2' \cup (-P_1)$, the P_1 links contained in the set $P_1' \cup P_2'$ will form loops with the $-P_1$ links. For example, if a P_1 link (u, v) is contained in the set $P_1' \cup P_2'$, then it will create a loop with the link (v, u) on $-P_1$ between the nodes u and v. The length of this loop is zero because $w(v, u) = -w(u, v)$. Let us denote $O_l = (P_1' \cup P_2') \tilde{\cap} (-P_1)$, which means that the set O_l consists of each P_1 link in the union of $P_2 \cup P_1$ and its corresponding $-P_1$ link. We define the logical difference set[4] between $P_1' \cup P_2'$ and P_1 as $(P_1' \cup P_2') - P_1 = P_1' \cup P_2' \cup (-P_1) \backslash O_l$. In fact, $l(O_l) = 0$ because the set O_l consists of loops with zero length, each consisting of a pair of opposite P_1 and $-P_1$ links. With $l(-P_1) = -l(P_1)$, we have $l((P_1' \cup P_2') - P_1) = l((P_1' \cup P_2') \cup (-P_1)) - l(O_l) = l(P_1') + l(P_2') + l(-P_1) = l(P_1') + l(P_2') - l(P_1)$, which is exactly Y in (7.2). ■

Lemma 48 shows that the logical difference set forms the shortest path in the modified graph, where P_1 is replaced with $-P_1$.

Lemma 48 *Given a directed graph $G = (V, E)$ and the source-destination pair (s, t) and let P_1 be the shortest path between s and t in G. $G' = (V, E')$ is defined as the graph $G = (V, E)$, for which the path P_1 is replaced with $-P_1$. The logical difference set $P_1' \cup P_2' - P_1$, between the optimal set of two link-disjoint paths $\{P_1', P_2'\}$ and the shortest path P_1, forms the shortest path P_2 from node s to node t in G'.*

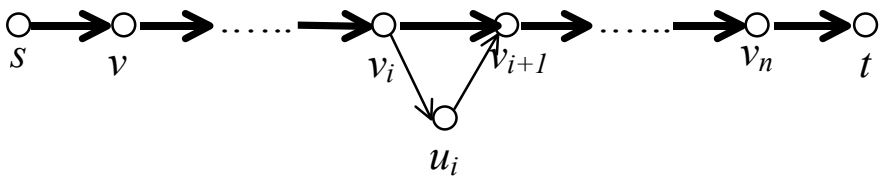
Proof. We will first prove that $P_2 = P_1' \cup P_2' - P_1$ is a complete path from s to t in $G' = (V, E')$, then we will prove that P_2 is the shortest path in $G' = (V, E')$.
Part A. From Lemma 46, the optimal set of two link-disjoint paths $P_1' \cup P_2'$ must contain either the first shortest path P_1 itself or some P_1 links on each of its two paths.
If $(P_1' \cup P_2') \supset P_1$, without loss of generality suppose $P_1' = P_1$, then $O_l = P_1 \cup (-P_1)$. With the definition of logical difference set, $P_2 = ((P_1' \cup P_2') \cup (-P_1)) \backslash O_l = (P_1 \cup P_2' \cup (-P_1)) \backslash (P_1 \cup (-P_1)) = P_2'$. Hence, P_2 must be a complete path from s to t.
If $P_1' \cup P_2'$ contains some P_1 links on each of its two paths, and given that $-P_1$ is the path from t to s in $G' = (V, E')$, and neither P_1' nor P_2' contains any $-P_1$ links, then the union $P_1' \cup P_2' \cup (-P_1)$ contains two cycles: one cycle consists of P_1' and $-P_1$, the other

[4] The logical difference set $P_2 - P_1$ can also be computed as $P_2 - P_1 = \{(u, v) | (u, v) \in P_2 \backslash (P_2 \cap P_1)\} \cup \{(v, u) | (u, v) \in P_1 \backslash (P_2 \cap P_1)\}$, which means that if a link (u, v) of P_2 does not appear on P_1, then this link belongs to the difference set $P_2 - P_1$, and if a link (u, v) of P_1 does not appear on P_2, then its direction reversed link (v, u) belongs to the difference set $P_2 - P_1$, with a link weight $w(v, u) = -w(u, v)$. In set theory, the difference operation is defined as $P_2 - P_1 = P_2 \backslash (P_1 \cap P_2)$ and the symmetric difference operation is defined as $P_2 - P_1 = (P_2 \cup P_1) \backslash (P_1 \cap P_2)$. The concept of logical difference set in this chapter resembles the symmetric difference set, but it is not the same.

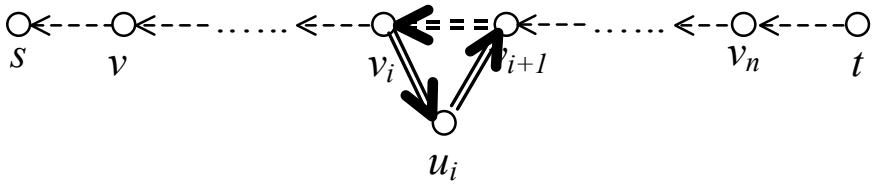
consists of P_2' and $-P_1$. When the set O_l is removed from the union set, the remaining links compose the logical difference set P_2. Hence, P_2 must be a complete path from s to t.
Part B. Assume that the shortest path in $G' = (V, E')$ is $P_3 \neq P_2$, then we must have $l(P_3) < l(P_2)$. As $l(P_2) = l(P_1') + l(P_2') - l(P_1)$, we have $l(P_3) + l(P_1) < l(P_1') + l(P_2')$, which contradicts the assumption that $\{P_1', P_2'\}$ is the optimal set. ■

7.3.3 LBA is loop-free

Many routing algorithms assume non-negative link weights to avoid negative-length cycles. However, negative link weights introduced to a graph in LBA will not cause negative cycles.



(a) The shortest path $P_1(s,t)$



(b) A loop containing some $-P_1$ link

Figure 7.5: A loop containing a negative link.

Theorem 49 *Given a directed graph $G = (V, E)$ and source-destination pair (s, t) and let P_1 be the shortest path between s and t in G. The modified graph $G' = (V, E')$ is defined as the graph $G = (V, E)$ for which P_1 is replaced with $-P_1$. A cycle containing some negative weight(s) in G' will not have a negative length.*

Proof. Assume $sv_1...v_iv_{i+1}...v_nt$ is the shortest path P_1 from node s to node t in $G = (V, E)$, as shown in Figure 7.5(a). The corresponding path $-P_1$ in $G' = (V, E')$ (Figure 7.5(b)) has a link (v_{i+1}, v_i), which appears on loop (i.e., cycle) $P_l = u_iv_{i+1}v_iu_i$. Suppose

the loop P_l has a negative length $l(P_l) = w(u_i, v_{i+1}) + w(v_{i+1}, v_i) + w(v_i, u_i) < 0$. Because $w(v_{i+1}, v_i) = -w(v_i, v_{i+1})$, we must have $w(v_i, u_i) + w(u_i, v_{i+1}) < w(v_i, v_{i+1})$. Hence, the sub-path $sv_1...v_iu_iv_{i+1}$ is shorter than the sub-path $sv_1...v_iv_{i+1}$. This contradicts the assumption that $sv_1...v_iv_{i+1}$ $...v_nt$ is the shortest path. ■

7.3.4 Optimality of LBA

Theorem 50 *Given a directed graph $G = (V, E)$ and source-destination pair (s, t), the algorithm LBA returns the optimal set for the LPP problem.*

Proof. Let P_1 be the shortest path in the graph $G = (V, E)$ found in Step 1 of LBA and P_2 be the shortest path in the modified graph $G' = (V, E')$ found in Step 3 of LBA. $\{P_1', P_2'\}$ is the solution set generated by LBA. The proof consists of three parts.
Part A (Proof of Link-disjointness). By construction of the solution set, there must hold that $P_1' \cap P_2' = \varnothing$.
Part B (Proof of Minimal Total Length). Suppose the optimal set of link-disjoint paths is $\{P_1^*, P_2^*\}$, instead of $\{P_1', P_2'\}$. According to Lemma 48, the logical difference set of $\{P_1^*, P_2^*\}$ with P_1 is the shortest path in the modified graph G'. This contradicts that P_2 is the shortest path in the modified graph G'.
Part C (Proof of Loop-freeness). On Theorem 49, LBA is loop-free.
Thus the solution set returned by LBA must be the optimal set. ■

7.4 Extending LBA to multiple dimensions

The extension of LBA to multiple dimensions using SAMCRA (see Section 4.6) is called MLBA (Multi-constrained LBA). The basic steps of MLBA are presented and the problems appearing in multiple dimensions are addressed.

7.4.1 Operations of MLBA

The basic steps of MLBA are the same as those for LBA, except that the shortest path routing algorithm is replaced with SAMCRA. We will illustrate the operation of MLBA with the example topology shown in Figure 7.6(a). For the sake of simplicity, each link has been assigned a two-dimensional weight vector, but it is possible to use any m-dimensional weight vector ($m \geq 1$). The problem is to find two link-disjoint paths from source node a to destination node b that both obey the constraints vector $\vec{L} = (20, 20)$, with the minimum total length, according to (4.3). The shortest multi-constrained path from node a to node b is the path $a - c - d - b$. Its path weight vector equals $(4, 5)$. The optimal set of two shortest link-disjoint paths (according to (7.1)) in this topology is $\{a - c - b, a - d - b\}$, with path vectors $(5, 6)$ and $(5, 5)$ respectively and minimum total length $0.3 + 0.25 = 0.55$.

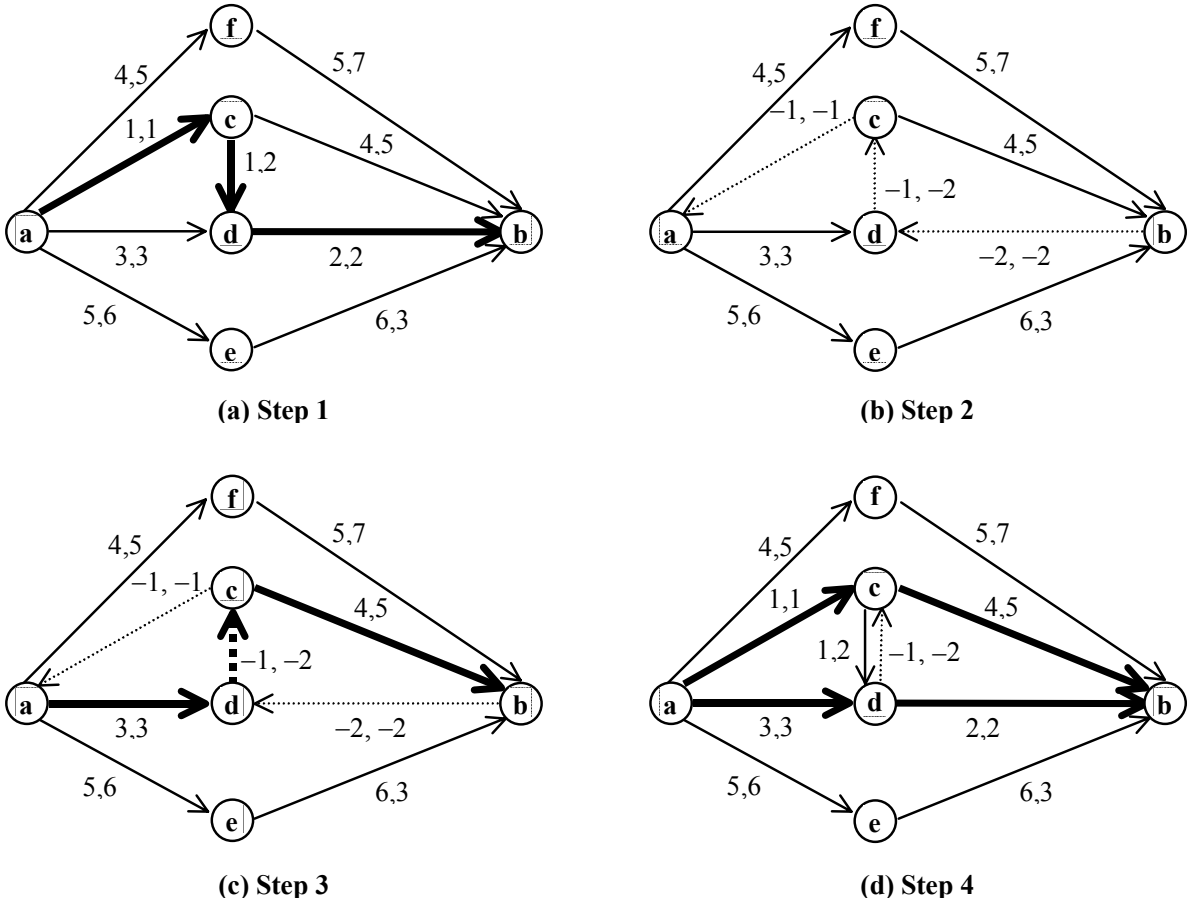


Figure 7.6: Example of the operation of MLBA.

The execution of MLBA on this topology follows: in Step 1, the shortest path $P_1 = a - c - d - b$ is found. In Step 2, the original graph is modified by replacing all the P_1 links with $-P_1$ links. In this case, each link weight vector component of a $-P_1$ link is set negative. For instance, the link (c, d) with weight vector $(1, 1)$ is replaced with the link (d, c) with weight vector $(-1, -1)$. In Step 3, the shortest path in the modified graph, found with SAMCRA, is $P_2 = a - d - c - b$, with path weight vector $(3, 3) + (-1, -1) + (4, 5) = (6, 6)$. In Step 4, the set O_l, consisting of a pair of opposite P_1 and P_2 links (c, d) and (d, c), is removed from the union of P_1 and P_2. Then the optimal solution set $\{a - c - b, a - d - b\}$ is returned.

7.4.2 Problems in multiple dimensions

Loops caused by negative cycles

For $m = 1$, SAMCRA acts just like Dijkstra's algorithm, therefore MLBA reduces to LBA and negative cycles cannot occur. For $m > 1$, theorem 49 may not hold for all m measures. Some of the components of the loop weight vector may become negative, causing MLBA to pass this loop multiple times, as clarified through Figure 7.7.

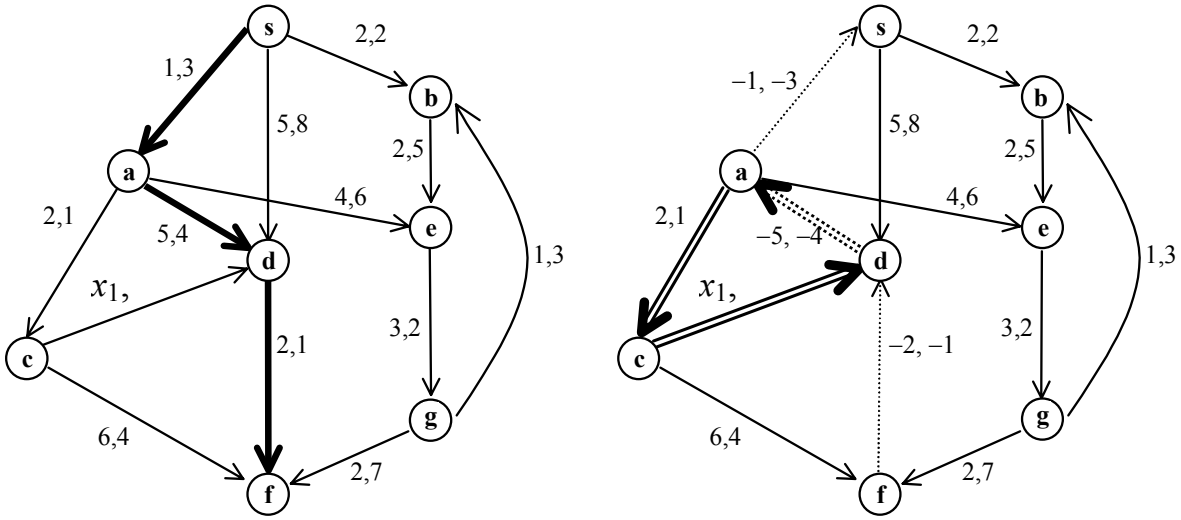


Figure 7.7: The non-dominance concept may fail to remove a loop when $m > 1$.

Suppose that the shortest path P_1 is $s-a-d-f$, depicted with bold lines in Figure 7.7(a). The link weight vector (x_1, x_2) of link (c, d) must be chosen in such a way that the path $s-a-c-d-f$ is longer than $s-a-d-f$, i.e.

$$\begin{aligned} & \max \begin{bmatrix} w_1(s,a) + w_1(a,c) + x_1 + w_1(d,f) \\ w_2(s,a) + w_2(a,c) + x_2 + w_2(d,f) \end{bmatrix} \\ > \ & \max \begin{bmatrix} w_1(s,a) + w_1(a,d) + w_1(d,f) \\ w_2(s,a) + w_2(a,d) + w_2(d,f) \end{bmatrix} \end{aligned}$$

Numerically,

$$\max \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} > \max \begin{bmatrix} w_1(a,d) - w_1(a,c) \\ w_2(a,d) - w_2(a,c) \end{bmatrix} = \begin{bmatrix} 5-2 \\ 4-1 \end{bmatrix} = \begin{bmatrix} 3 \\ 3 \end{bmatrix} \tag{7.3}$$

After Step 2 of MLBA is completed, there appears a loop $P_l = d-a-c-d$ shown with double lines in Figure 7.7(b), containing the link (d, a) with negative link weights $(-5, -4)$.

If equation (7.3) holds and each component of vector (x_1, x_2) is greater than 3, then the sub-path $s-d-a-c-d$ will be dominated by the direct link (s,d) with weight vector $(5,8)$ and will be removed by the non-dominance check in SAMCRA. However, if equation (7.3) holds, but one component of (x_1, x_2) is not greater than 3 say $x_1 < 3$ and $x_2 > 3$, then a negative cycle for measure 1 appears and

$$\left[\begin{array}{l} w_1(P_l) = w_1(d,a) + w_1(a,c) + w_1(c,d) = -5 + 2 + x_1 < 0 \\ w_2(P_l) = w_2(d,a) + w_2(a,c) + w_2(c,d) = -4 + 1 + x_2 > 0 \end{array}\right]$$

where $(w_1(P_l), w_2(P_l))$ is the path vector of the loop P_l. In this case, the sub-path $s-d-a-c-d$ is not dominated by the link (s,d), although $l(P_l) > 0$. Hence, loops may occur in MLBA that continue until one of the constraints is violated. Checking paths to assure that they are loop-free could be a viable solution to this problem.

As mentioned in Section 7.2, in Suurballe's algorithm [154] and the S-T algorithm [155], a transformation of link weights $w'(u,v) = w(u,v) + d[u] - d[v]$ is applied to each link, where $d[u]$ is the distance from source node s to node u on the shortest path tree. This transformation is made to guarantee that the links on the shortest path tree have zero-valued link weights and those links not on the tree have link weights greater than zero in the modified graph. However, an artifact of a non-linear length is that subsections of shortest paths are not necessarily shortest paths [38], [163]. As a result, for $m > 1$, Suurballe's transformation cannot ensure non-negative cycles and hence loops may emerge.

Total length of the solutions produced with MLBA

Assume for the moment that the constraints are large enough such that all paths are feasible. If $m = 1$, it has been proved in Section 7.3 that the solution set $\{P_1', P_2'\}$ produced with MLBA has minimum total length. With the total length defined in (7.1), Lemma 46 still holds for $m > 1$. The optimal set of two link-disjoint multi-constrained paths with minimum total length either contains the first shortest path P_1 itself or some P_1 links on each of its two paths. Also, the optimal set $\{P_1', P_2'\}$ still obeys Property 47. Unfortunately, the logical difference set $(P_1' \cup P_2') - P_1$ is not necessarily the shortest path P_2 in the modified graph, since $l((P_1' \cup P_2') - P_1) = l(P_1') + l(P_2') - l(P_1)$ does not necessarily hold for $m > 1$. Hence, Lemma 48 may not hold for $m > 1$ and the solution set constructed based on P_1 and P_2 is not necessarily the optimal set with minimum total length. This problem does not occur when using the semi-linear length function (4.4). However, regardless of the length function, the solution set for $m > 1$ may violate the constraints or a feasible solution may not be acquired.

7.5 DIMCRA

The previous section established that it is not straightforward to extend LBA to multiple dimensions. Due to the problems existing in MLBA, we propose a heuristic algorithm DIMCRA (link-DIsjoint Multiple Constraints Routing Algorithm) for the MCLPP problem.

7.5.1 Operations of DIMCRA

Given a directed graph $G = (V, E)$, a constraint vector $\vec{L}$ and a source-destination pair (s, t), DIMCRA carries out the following steps:

1. Find with SAMCRA the shortest path P_1 obeying $\vec{L}$; if P_1 does not exist, then stop.

2. Reverse the direction of all the links on the shortest path P_1, and set the value of their link weights zero, $w_i(v, u) = 0$, $\forall(u, v) \in P_1$ and $i = 1, ..., m$: a modified graph G' is created.

3. Find with SAMCRA the shortest path P_2 constrained by $2\vec{L}$ in the modified graph G'; if P_2 does not exist, then stop.

4. Make the union of P_1 and P_2, remove from the union the P_1 links whose reversed links appear on P_2, and vice versa. Then group the remaining links into a set of two paths $\{P_1', P_2'\}$, i.e. $P_1' \cup P_2' = (P_1 \cup P_2) \setminus (P_1 \tilde{\cap} P_2)$.

5. Check the length of each path in the set $\{P_1', P_2'\}$. If the path P_i' $(i = 1, 2)$ violates the constraints, then update the modified graph G' by removing the link set $P_i' \setminus (P_i' \cap P_1)$ from it, and go to Step 3. Otherwise stop and return the current solution set $\{P_1', P_2'\}$.

Compared to MLBA, DIMCRA uses a different transformation to create the modified graph. In Step 2 of DIMCRA, the shortest path links are still reversed in direction but the corresponding direction–reversed links are assigned zero-valued link weight vectors, instead of negative ones. In this way negative cycles cannot occur. In MLBA, P_2 is required to obey the constraints, which may cause some feasible sets to be ignored by MLBA. In fact, when $\vec{w}(P_2) > \vec{L}$, if P_2 contains no reversed P_1 link(s), then $\{P_1', P_2'\}$ is actually $\{P_1, P_2\}$ and cannot be a feasible set. But, if P_2 contains some reversed P_1 link(s), it is possible that $\{P_1', P_2'\}$ is a feasible set, for instance, $l(P_1) = 0.6, l(P_1') = 0.8, l(P_2') = 0.9$, and $l(P_2) = 1.1$. However, if $\vec{w}(P_2) > 2\vec{L}$, then it is not possible to find a feasible solution, because $\vec{w}(P_1' + P_2') = \vec{w}(P_2 + P_1 - P_{1r}) > \vec{w}(P_2) \geq 2\vec{L}$, where P_{1r} denotes the set of P_1 links whose reversed links appear on P_2, and P_{1r} must be a

proper subset of P_1. Therefore, in Step 3 of DIMCRA, the constraints check on path P_2 in SAMCRA is performed with $2\vec{L}$ as the constraints vector, otherwise a feasible solution set may not be found. We have also added an extra step, Step 5 of DIMCRA, to check that the constraints are obeyed. With only the condition $\vec{w}(P_1' + P_2') \leq 2\vec{L}$, DIMCRA cannot ensure both paths to be within the constraints, i.e. $\vec{w}(P_1') \leq \vec{L}$ and $\vec{w}(P_2') \leq \vec{L}$. Hence, Step 5 of DIMCRA checks both paths in the solution set returned at Step 4. If each of them obeys the constraints, DIMCRA will return the solution set and stop. On the other hand, if either of them does not obey the constraints, DIMCRA is redirected to Step 3 to continue the search for a feasible set. In Step 3, if no P_2 exists, DIMCRA will stop with no solution. We will elucidate the operation of DIMCRA with the following examples.

Example 1: Consider the example graph in Figure 7.8(a). The problem is to find two link-disjoint paths between a and b, each within the constraints $\vec{L} = (20, 20)$ and preferably with minimum total length. In Step 1, the shortest path $P_1 = a - c - d - b$ is found. In Step 2, each P_1 link is reversed and is assigned zero weights. In Step 3, the shortest path in the modified graph G' is found as $P_2 = a - d - c - b$, with path vector $(3, 3) + (0, 0) + (4, 5) = (7, 8)$, as shown with bold lines in Figure 7.8(c). In Step 4, only for the P_1 link (c, d), its reversed link (d, c) appears on P_2, and vice versa. Thus these two links are removed from the union of P_1 and P_2 and the remaining links are grouped into a set of two paths $\{P_1', P_2'\} = \{a - c - b, a - d - b\}$, shown with bold lines. In Step 5, both paths pass the constraints check and DIMCRA stops. In this case, the optimal set of $\{a - c - b, a - d - b\}$ is returned by DIMCRA. The solution set that would have been returned by RF is not optimal.

Example 2: Consider the graph in Figure 7.9(a), which is the same as in the previous example except that the link (e, b) is assigned a different vector $(2, 1)$. The constraints remain the same. In this example the optimal set of two link-disjoint multi-constrained paths is still the set $\{a - d - b, a - c - b\}$ with path vectors $(5, 5)$ and $(5, 6)$, respectively, and the minimum total length $5/20 + 6/20 = 0.55$. In Step 3, the shortest path in the modified graph is $P_2 = a - e - b$ with path vector $(7, 7)$, shown in Figure 7.9(c). In Step 4, the solution set $\{P_1', P_2'\}$ is constructed as $\{a-c-d-b, a-e-b\}$, exactly P_1 and P_2 themselves. The total length of $\{a - c - d - b, a - e - b\}$ is $5/20 + 7/20 = 0.6$. In this example, DIMCRA failed to return the optimal set, but DIMCRA's solution set is close to the optimal one and both paths obey the constraints.

Example 3: Consider again example 2, but with different constraints $\vec{L} = (6, 6)$. Running DIMCRA, Step 1 to Step 4 obtain the same results as in example 2. But in Step 5, the longer path $P_2' = a - e - b$ with path vector $(7, 7)$ does not obey the constraints. This means that the current solution set is not feasible. The links that only appear on $P_2' = a - e - b$, i.e. link (a, e) and (e, b), are removed from the modified graph shown in Figure 7.9(b). The updated modified graph is shown in Figure 7.10(a) and DIMCRA is redirected to Step 3. In Step 3, a shortest path in the updated modified

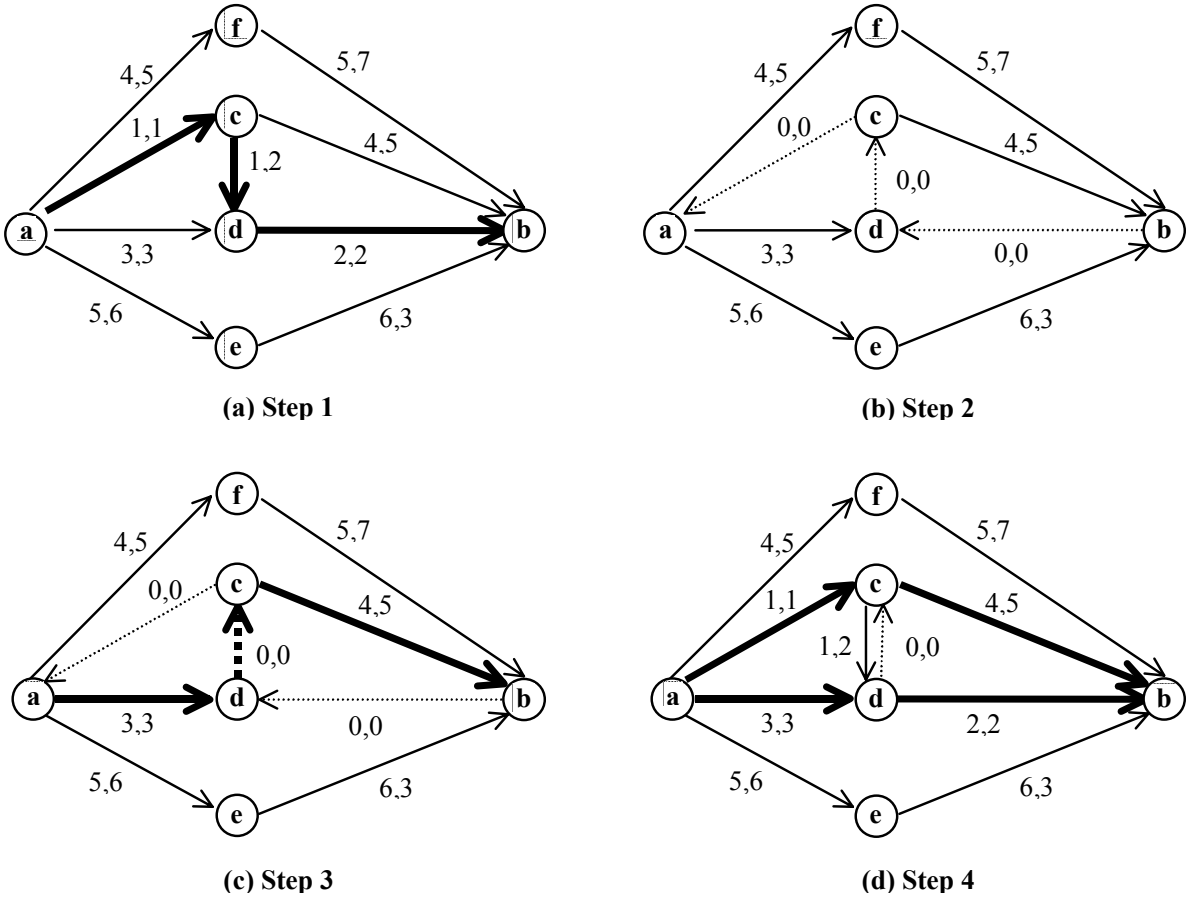


Figure 7.8: Example 1 of the operation of DIMCRA.

graph is found as $P_2 = a - d - c - b$, depicted in Figure 7.10(b). In Step 4, the solution set becomes $\{a - c - b, a - d - b\}$, as shown in Figure 7.10(c). At last, in Step 5, each path in the solution set obeys the constraints. The set $\{a - c - b, a - d - b\}$ is returned and DIMCRA stops. RF would have failed to return a solution.

With the constraints check on each path in the solution set, Step 5 guarantees that DIMCRA returns a feasible set of link-disjoint multi-constrained paths, as illustrated in the above examples.

However it may occur, as illustrated in Figure 7.11, that DIMCRA cannot return a feasible set, even if there exists one. The RF method also fails to return a feasible set in such cases.

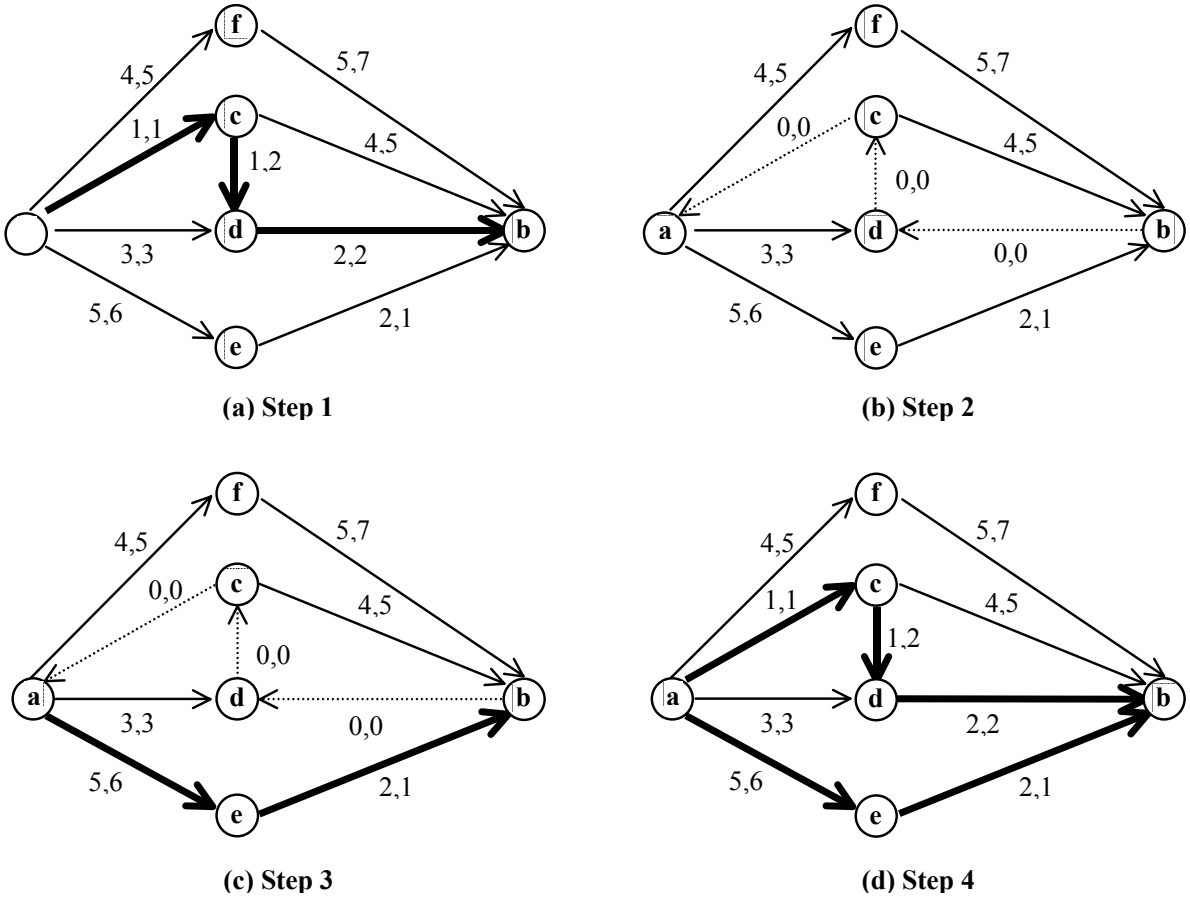


Figure 7.9: Example 2 of the operation of DIMCRA.

7.5.2 Properties of DIMCRA

By reversing the shortest path P_1, finding a shortest path P_2 in the modified graph and constructing the solution set based on these two shortest paths P_1 and P_2, the disjointedness of the two paths in the solution set is guaranteed. Setting the weights of the direction-reversed P_1 links to zero guarantees loop-freeness. For, if no negative link weights are used in a graph, negative cycles cannot occur and loops can be avoided by the non-dominance check in SAMCRA. Compared to the operation of negating direction-reversed P_1 link weights, assigning the weights zero still encourages the choice of the reversed P_1 links on a path, but with less intensity.

DIMCRA does not always find the set of feasible link-disjoint paths. Hence, it may be possible to further optimize DIMCRA, such that it can guarantee to always find a set of feasible link-disjoint paths, if they exist. However, DIMCRA in its current state performs better than the RF method (as was indicated in the examples). Both

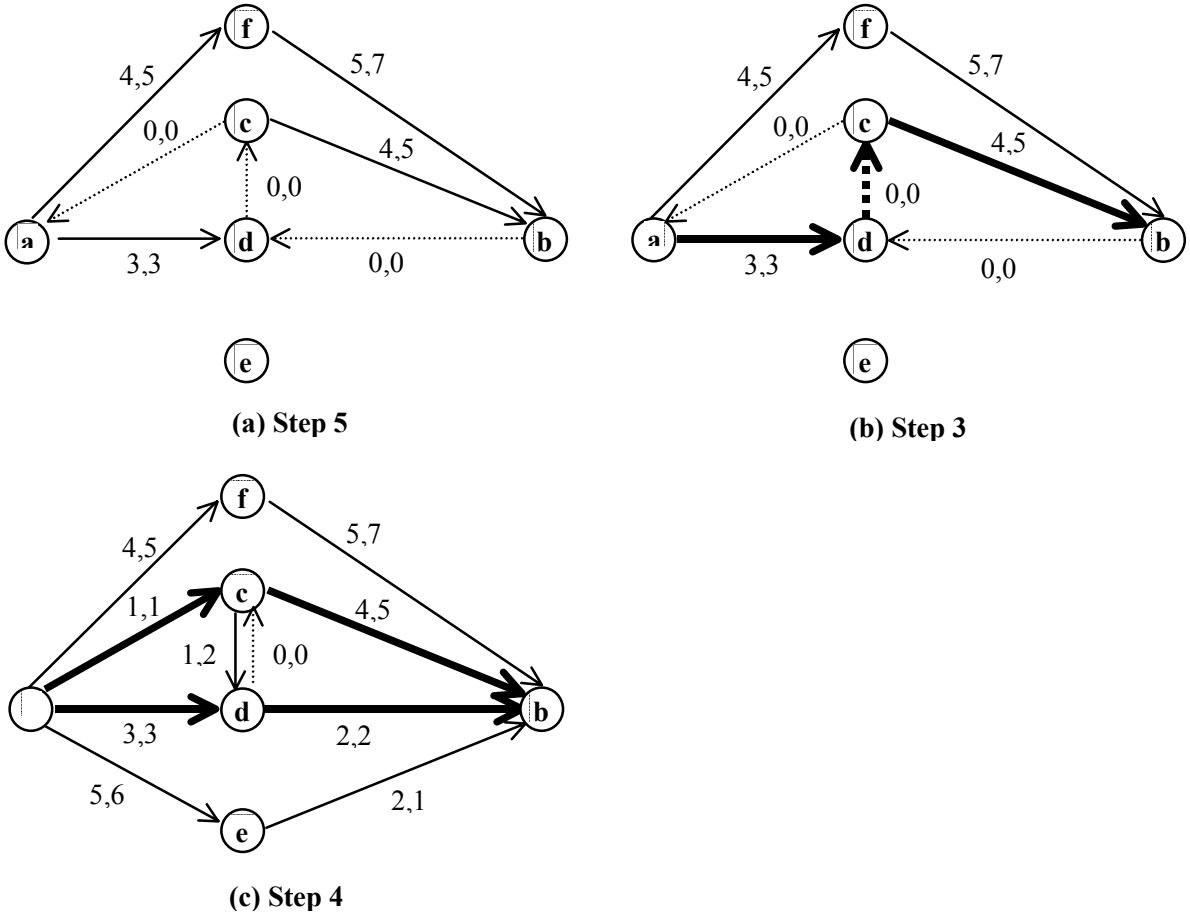


Figure 7.10: Example 3 of the operation of DIMCRA.

methods return the same solution when $\{P_1', P_2'\} = \{P_1, P_2\}$ and $P_1 \cap P_2 = \varnothing$. In all other cases DIMCRA either returns a better solution than RF or RF does not find a solution where DIMCRA does. Since, to our knowledge, currently no other algorithms for solving MCLPP exist, the performance of DIMCRA has not been evaluated and compared. We have only indicated its superiority over the RF method.

7.6 Conclusions

The link-disjoint paths problem occurs in network design, where aspects as survivability, load balancing, and efficient resource utilization are strived for. This problem has barely been investigated in the context of QoS routing, where a path is characterized by multiple measures. A simple algorithm for solving the one-dimensional problem

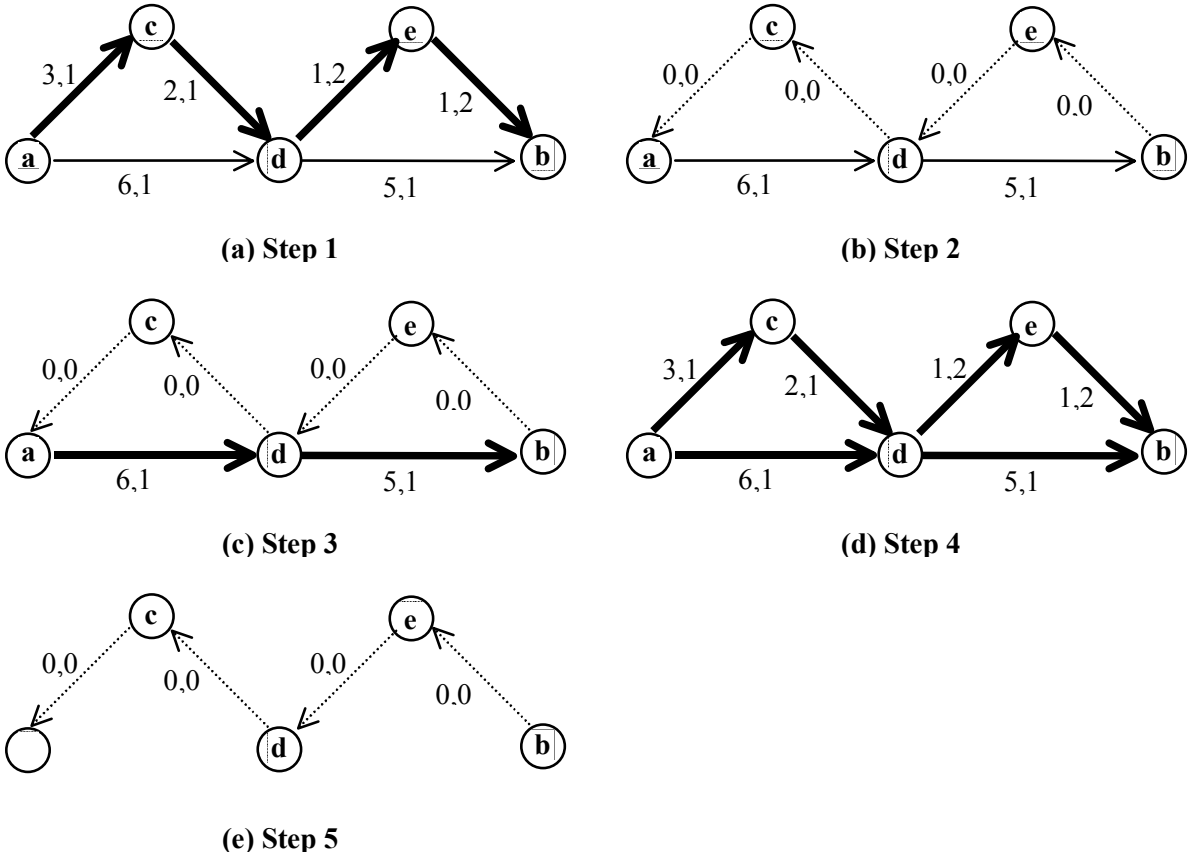


Figure 7.11: Example of the operation of DIMCRA with constraints (10,10).

was presented in this chapter. The problems surrounding the extension of this simple algorithm to multiple dimensions were discussed. A heuristic algorithm DIMCRA was proposed to find link-disjoint multi-constrained paths between a pair of source and destination nodes. If DIMCRA returns a link-disjoint pair of paths, then they always obey the constraints. However, DIMCRA's solution is not necessarily optimal in terms of minimizing the total length of the returned paths or guaranteeing to always find a feasible set. Its performance however is better than the simple Remove-Find method.

Chapter 8

The complexity of exact MCP algorithms

In the previous chapters, the concepts of QoS routing were presented and QoS algorithms for unicast, multicast, and link-disjoint QoS routing were proposed. QoS routing embodies an NP-complete problem. This chapter will look into the complexity of unicast QoS routing to see whether exact algorithms for it will provide a feasible solution in practice.

In particular, the complexity of exactly solving the MCP problem is explored. The MCP problem and the notation used were presented in Chapter 1.

This chapter is organized as follows. Section 8.1 presents an overview of related work. Section 8.2 analyzes the worst-case NP complexity of the MCP problem. The proof that the MCP problem is NP-complete strongly depends on the size of the link weights and the level of correlation between those link weights. Section 8.3 evaluates, mathematically and simulative, the impact of correlation on the complexity of solving the MCP problem. Section 8.4 discusses the impact of the constraint values on the complexity and introduces the concept of phase transitions in the MCP problem. Finally, Section 8.5 epitomizes the conclusions.

8.1 Related work

Garey and Johnson [57] were the first to list the MCP problem with $m = 2$ as being NP-complete, but they did not provide a proof. Wang and Crowcroft have provided this proof for $m \geq 2$ in [171] and [170]. Their proof basically consisted of reducing the MCP problem for $m = 2$ to an instance of the *partition* problem, a well-known NP-complete problem [57]. The effect of this proof has been tremendous, because it connotes to many that the MCP problem is intractable, in which case heuristics should be used. Many simulations performed in [37], [38], [100], [163], [166] suggest that exact QoS routing

may not be intractable in practice. There are certain NP-complete problems, such as *partition*, which are considered by many practitioners to be tractable in practice. The reason for this is that, although no algorithms for solving them in time bounded by a polynomial in the input length (e.g., N, M) are known, there exist algorithms that solve those problems in time bounded by a polynomial in the input length and the magnitude of the largest number (e.g., largest QoS weight or constraint), in the given problem instance [56]. Such algorithms are called pseudo-polynomial time algorithms. NP-complete problems for which no exact pseudo-polynomial time algorithm exists, are called NP-complete in the strong sense. In the case of the *partition* problem, the NP-completeness strongly depends on the fact that arbitrarily large numbers are allowed. If any upper bound was imposed on these numbers in advance, even a bound which is a polynomial function of the input length, there would exist a polynomial time algorithm for solving this (restricted) problem [56].

David Pisinger [134] has evaluated Knapsack problems, which are NP-complete problems (proved via reduction to the *partition* problem) and found that in practice these problems are tractable. For many more NP-complete problems, typical cases are "easy" to solve. A study of the phenomenon that typical cases are "easy," was performed by Cheeseman *et al.* [26], who introduced the concept of phase transitions in NP-complete problems. According to Cheeseman *et al.*, NP-complete problems which are very under-constrained are soluble and it is usually easy to find one of the many solutions. NP-complete problems which are very over-constrained are insoluble. In the phase transition in between, as illustrated in Figure 8.1, problems are "critically constrained" and it is typically very hard to determine if they are soluble or insoluble [58]. For a more formal discussion of phase transitions we refer to [41]. Cheeseman *et al.* have conjectured that all NP-complete problems have at least one order parameter and that the hard to solve problems are around a critical value of this order parameter. Although this conjecture does not hold for all NP-complete problems, there seems to be a connection between complexity and phase transitions. The lack of a phase transition seems to have significant computational implications [82]: such problems are either computationally tractable, or well-predicted by a single, trivial algorithm. Note that the existence of a phase transition may also occur in problems that are not NP-complete (e.g., see [44]). Monasson *et al.* [116], have reported an analytic solution and experimental investigation of the phase transition in K-satisfiability (the first problem shown to be NP-complete). Gent and Walsh [58] have shown that phase transitions occur in the *partition* problem.

Levin [106] advocated a different study of NP-complete problems by introducing the concept of average-case complexity. He indicated that some NP-complete problems are "easy on average," while other (average case NP-complete) problems may not be.

There exists also some work in the literature revealing important properties of the MCP problem. Three of those properties strengthen our belief that in practice exact QoS routing is tractable. First of all, the MCP problem is not strong NP-complete.

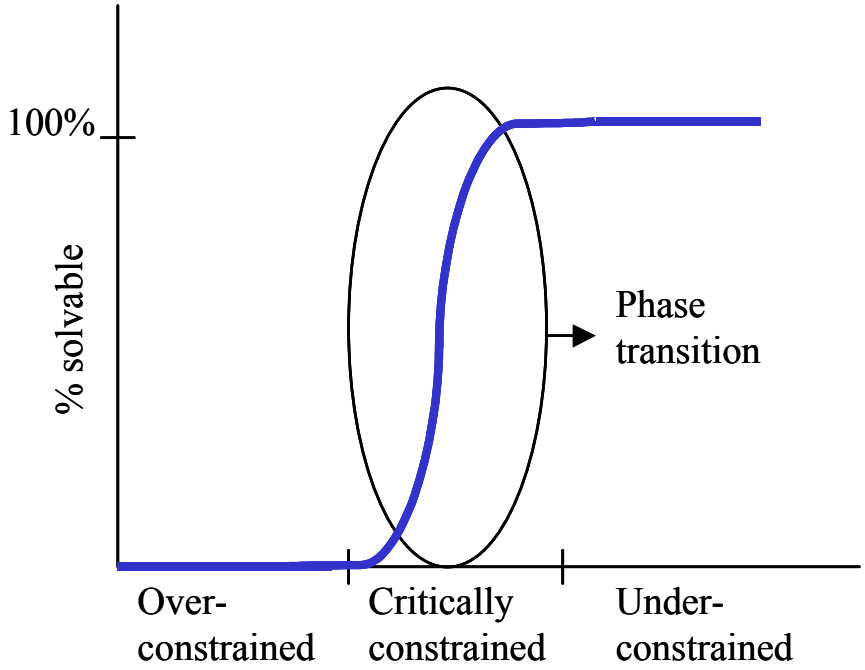


Figure 8.1: The solubility of an NP-complete problem around a phase transition.

Secondly, if all, but one, measures take bounded integer values, then the MCP problem is solvable in polynomial time [27]. Finally, if some specific dependencies exist between the QoS measures, exact QoS routing can be performed in polynomial time [114]. The goal of this chapter is to provide more evidences that suggest the tractability of exact QoS routing, in practice.

8.2 Worst-case complexity analysis

In this section, the worst-case complexity of the MCP problem will be analyzed. First, for perspicuity, we will provide a more condensed version of the proof that the MCP problem is NP-complete [171], [170], and refer to it as the NP-proof.

Theorem 51 *The MCP problem for $m \geq 2$ is NP-complete.*

Proof. First, the theorem is proved for $m = 2$. Given a chain topology with $n + 1$ nodes and $2n$ links, each with a two-component weight vector $\vec{w}$ as depicted in Figure 8.2 and a set of numbers $a_i \in A$, $0 \leq a_i \leq S$, for $i = 1, ..., n$, where $S = \sum_{i=1}^{n} a_i$. The constraints are chosen as follows: $L_1 = nS - \frac{S}{2}$ and $L_2 = \frac{S}{2}$. To solve the MCP problem, we need to find a path from node 1 to node $n+1$, that obeys the constraints.

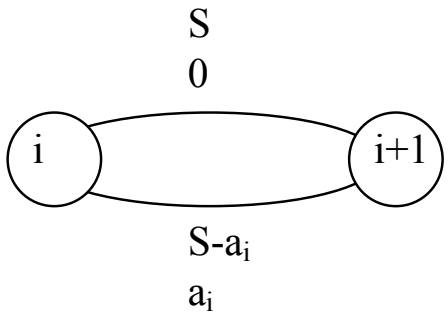


Figure 8.2: The assignment of link weights to the links between nodes i and $i+1$, in a chain topology.

Since, for all link weight vectors, the sum of the components equals S, we have that $w_1(P) + w_2(P) = nS$. Accordingly, a solution satisfying the constraints is only found if $w_1(P) = nS - \frac{S}{2}$ and $w_2(P) = \frac{S}{2}$. The problem has now become an instance of the well-known NP-complete *partition* problem [57] and can only be solved by finding the set $A' \subseteq A$, for which $\sum_{a_i \in A'} a_i = \frac{S}{2}$. A feasible path exists if the set A' exists, in which case it is retrieved by choosing the lower link if $a_i \in A'$ and the upper link if $a_i \notin A'$.

This proves that the MCP problem with $m = 2$ is NP-complete. The proof that MCP for $m > 2$ is NP-complete inductively follows. Assume that the MCP problem with m measures is NP-complete. If we extend the number of measures with 1 to $m+1$ and choose $L_{m+1} = \sum_{(u,v) \in E} w_{m+1}(u, v)$, then all paths between source and destination obey this constraint. The MCP problem with $m+1$ measures is then only solved if the MCP problem with m measures is solved. This concludes the proof. ■

If the constraints are chosen such that only one feasible path exists, then the MCP problem is equal to the MCOP problem and hence the MCOP problem is also NP-complete.

Corollary 52 *The MCP problem is not NP-complete in the strong sense.*

Proof. The MCP problem is not strong NP-complete, because there exist pseudo-polynomial algorithms that exactly solve this problem (see [85], [111]). ■

The proof that a problem is NP-complete or not is entirely based on a worst-case argument. A problem is called polynomially solvable if it can be solved by an algorithm that terminates after a number of steps (instructions) that is bounded by a polynomial in the input length. A problem is called NP-complete if there is even one instance that is not polynomially solvable (unless P = NP). It may occur that, in some instances, the running time required to solve the MCP problem is polynomial. We call those

polynomially solvable instances "tractable" and we will use the term "intractable" when instances require a running time that cannot be bounded by a polynomial function in the input length (i.e., they are not polynomially solvable).

We desire to distinguish the instances of the MCP problem that are tractable and those that are intractable. If we look at the graph on which the MCP problem should be solved, we could delineate the class of polynomially solvable graphs, i.e. the class of graphs in which the number of paths between two nodes increases as a polynomial function of N (e.g., tree-, circle-, and star-topologies). This class of graphs is most likely very small and therefore most graphs potentially can lead to intractability. Fortunately, the underlying topology *alone* is not sufficient to lead to intractability: a specific link weight structure is mandatory. For instance, if all link weights are assigned the value 1, then the MCP problem is polynomially solvable, regardless of the underlying topology. We will proceed by defining a link weight structure that leads to intractability in the chain topology. We will use the chain topology as depicted in Figure 8.3 and ascertain that all paths from source s to destination t are non-dominated (see Section 4.3).

In general, there are two important properties that can reduce the search space, when solving the MCP problem, without loosing exactness, namely non-dominance and the constraints themselves. If a sub-path P, from source node s to node i, exceeds one or more constraints, it can never become a feasible path[1], because the path weight vector from i to destination node t consists of non-negative weights. Similar, if for two paths P_1, P_2 from s to i holds that P_1 dominates P_2, then all weights of P_1 are smaller than (or equal to) those of P_2 and hence P_2 can be eliminated from the search space [38]. In Section 4.3, the maximum number of non-dominated paths that obey the constraints was shown to be upper bounded by $\frac{\prod_{i=1}^{m} L_i}{\max_j(L_j)}$, where the constraints L_i are expressed as an integer number of the smallest granularity. This value provides a worst-case estimate of the size of our search space.

According to Levin [106], some NP-complete problems are "easy on average," while other (average-case NP-complete) problems may not be. The average-case complexity therefore also gives some indication whether an NP-complete problem could be tractable in practice. In [166] we have shown that if the path weights are independently distributed in the solution space, then the MCP problem can be solved in polynomial time on average.

Without loss of generality, assume that the link weights in Figure 8.3 are chosen such that $a_i > c_i$ and $b_i < d_i$, for $i = 1, ..., N$ ($c_i > a_i$ and $d_i < b_i$ would also have been possible). It can be verified that if $a_i \geq c_i$ and $b_i \geq d_i$ or $c_i \geq a_i$ and $d_i \geq b_i$ were allowed, this would lead to dominance. Under this assumption, the following property holds.

[1]This also holds for the lower bound estimation of the end-to-end path weight vector $\vec{w}(P) + \vec{b}$, where $\vec{b}$ denotes a lower-bounds vector consisting of the m one-dimensional shortest path weights from i to t.

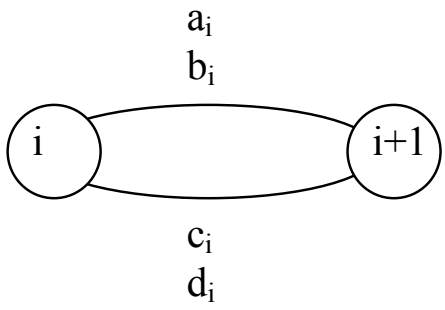


Figure 8.3: A chain topology with two QoS weights per link and N nodes in total.

Property 53 *If, in the chain topology in Figure 8.3, there holds that*

$$\begin{cases} a_i - c_i > \sum_{j=0}^{i-1} (a_j - c_j) \\ b_i - d_i < \sum_{j=0}^{i-1} (b_j - d_j) \end{cases} \tag{8.1}$$

for $i = 1, ..., N-1$, where $a_0 = b_0 = c_0 = d_0 = 0$, then all 2^{N-1} paths from node 1 to node N are non-dominated.

Proof. A proof by induction.
$i = 1$: There are two paths from node 1 to node 2, namely $P_1(1 \rightarrow 2) = \begin{pmatrix} a_1 \\ b_1 \end{pmatrix}$ and $P_2(1 \rightarrow 2) = \begin{pmatrix} c_1 \\ d_1 \end{pmatrix}$. According to formula (8.1): $a_1 > c_1$ and $b_1 < d_1$, which shows that both paths from node 1 to node 2 are non-dominated.
The inductive step is to assume the correctness of formula (8.1) for a certain i. It remains to prove that it also holds for $i + 1$:
There are 2^{i-1} paths from node 1 to i. From i there are two possible links to i+1, resulting in a total of 2^i paths from node 1 to node i+1. 2^{i-1} paths will follow the upper link from i to i+1, while the remaining 2^{i-1} paths will follow the lower link. Since all paths at i are non-dominated (inductive assumption), the paths following the upper link are also non-dominated, because the same vector is added to each of the path vectors. The same property applies to the paths that follow the lower link. It remains to show that if (8.1) holds, then the paths following the upper link and the paths following the lower link do not dominate each other.
If (8.1) is obeyed, then all paths following the upper link possess a first path weight that is larger than the first weights of the paths following the lower link. Similar, the paths following the lower link have a second weight, which is larger than the second weights of the paths following the upper link. Hence the paths following different links are non-dominated. ■

The *partition* problem is NP-complete, because the values involved in an instance of the *partition* problem may be arbitrarily large (or have an infinite granularity). The same phenomenon is observed in formula (8.1), where the difference between a_i and c_i (and correspondingly d_i and b_i) grows exponentially:

$$\begin{aligned} a_{i+1} - c_{i+1} &> \sum_{j=0}^{i} (a_j - c_j) = (a_i - c_i) + \sum_{j=0}^{i-1} (a_j - c_j) \\ &> 2\sum_{j=0}^{i-1} (a_j - c_j) > ... > 2^{i-1}(a_1 - c_1) \end{aligned}$$

If a_i in the NP-proof are not chosen according to formula (8.1), but if they take bounded integer values, then the problem becomes polynomially solvable.

A second important phenomenon observed from formula (8.1) is that the link weights display a perfect *negative correlation*. If the link weights would have had a positive correlation, then if $a_i > c_i$ most likely also $b_i > d_i$, leading to dominance.

Corollary 54 *Property 53 is a sufficient but also necessary condition for all paths in the chain topology to be non-dominated.*

Proof. We need to show that if formula (8.1) does not hold, then at least one path from node 1 to node $i+1$ is dominated. If (8.1) does not hold, either one of the following formulas holds.

$$\begin{cases} \sum_{j=0}^{i-1} c_j + a_i \leq \sum_{j=0}^{i-1} a_j + c_i \\ \sum_{j=0}^{i-1} d_j + b_i \geq \sum_{j=0}^{i-1} b_j + d_i \end{cases} \tag{8.2}$$

or

$$\begin{cases} \sum_{j=0}^{i-1} c_j + a_i > \sum_{j=0}^{i-1} a_j + c_i \\ \sum_{j=0}^{i-1} d_j + b_i \geq \sum_{j=0}^{i-1} b_j + d_i \end{cases} \tag{8.3}$$

or

$$\begin{cases} \sum_{j=0}^{i-1} c_j + a_i \leq \sum_{j=0}^{i-1} a_j + c_i \\ \sum_{j=0}^{i-1} d_j + b_i < \sum_{j=0}^{i-1} b_j + d_i \end{cases} \tag{8.4}$$

These formulas have been written in a slightly different from (8.1) to illustrate that they correspond to two paths, namely the path that followed all the lower links up to node i and took the upper link from node i to node $i+1$ and the path that took all the upper links towards node i and the lower link from node i to node $i+1$. Formula (8.2), without the equalities, is exactly the same as (8.1), but a is called c and b is called d. If the equality sign applies, then the path that followed all the lower links up to node i and took the upper link from node i to node $i+1$ is the same as the path that took all the upper links towards node i and the lower link from node i to node $i+1$. According to our definition of non-dominance (see Section 4.3), only one of these two paths is

non-dominated. When formula (8.3) applies, the path that followed all the lower links up to node i and took the upper link from node i to node $i+1$ is dominated by (or dominates in the case of formula (8.4)) the path that took all the upper links towards node i and the lower link from node i to node $i+1$. ■

Property 53 and corollary 54 seem very restrictive, because they are solely based on the chain topology and all paths must be non-dominated. If only a subset of all paths (that increases non-polynomially in N) were non-dominated, then the problem would still be intractable. However, if only such a subset of all paths would be non-dominated, then property 53 must hold for that subset. Otherwise, all link weights would be bounded and the problem would be polynomially solvable.

Also the chain topology can be put into perspective. Links in the chain topology can be seen as sub-paths.

Corollary 55 *If there are more than two links (all with two weights) between two nodes in the chain topology, formula (8.1) should hold for all possible pairs of links, in order for all paths from node 1 to node N to be non-dominated.*

In practice, it is not likely links/sub-paths obey formula (8.1). If formula (8.1) is not obeyed, corollary 55 suggests that when there are many sub-paths to a node, the probability that all these paths are non-dominated decreases and consequently also the search space decreases.

At the beginning of this section we have mentioned that there are two important properties to reduce the search space, namely non-dominance and the values of the constraints. If the constraint-values are chosen very large, then it will be easy to find a path that obeys these constraints. On the other hand, if the constraint values are very strict, there may not be a path available that can obey these constraints. For the chain topology, besides formula (8.1), the constraints must lie in the range:

$$\begin{cases} \sum_{j=0}^{N-1} c_j \leq L_1 \leq \sum_{j=0}^{N-1} a_j \\ \sum_{j=0}^{N-1} d_j \geq L_2 \geq \sum_{j=0}^{N-1} b_j \end{cases}$$

to lead to NP-completeness (i.e., then the MCP problem reduces to the *partition* problem as illustrated in the NP-proof). Since $c_i < a_i$, the shortest path for measure 1 from node 1 to node N, equals $\sum_{j=0}^{N-1} c_j$. If $L_1 < \sum_{j=0}^{N-1} c_j$, then no feasible path exists. If $L_1 > \sum_{j=0}^{N-1} a_j$, then all possible (loop-free) paths can obey this constraint. The same reasoning applies to L_2 and is further motivated in Section 8.4. In this section, the chain topology was used to create an intractable instance of the MCP problem. This instance provided some hints on the underlying causes of intractability. Section 8.3 will further evaluate the impact of correlation on the complexity of QoS routing.

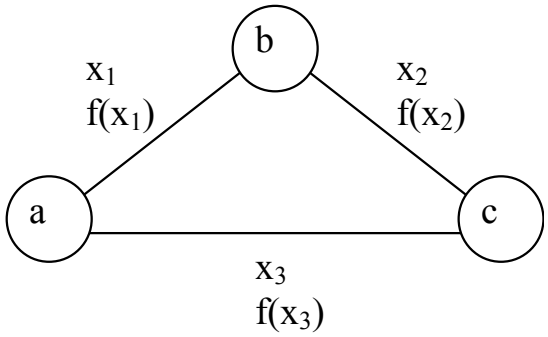


Figure 8.4: An example topology.

8.3 The impact of link correlation on complexity

Section 8.2 hinted at a connection between link correlation and complexity. This section will expatiate on the impact of link correlation on the complexity of QoS routing, by giving some properties and presenting simulation results. Specifically, *intra-link correlation* is treated, where the m measures on a link are correlated. *Inter-link correlation* refers to correlation over different links.

8.3.1 Theory

Ma and Steenkiste [114] have shown that when specific dependencies (correlation) exist between QoS measures due to Weighted Fair Queueing scheduling, QoS routing can be performed in polynomial time. However, it is a misconception that if all QoS measures are a function of a common measure, then by just minimizing this common measure, we will have minimized all measures. We will illustrate that this is not always the case and provide some conditions when this statement holds. We will denote by $f(.)$ a convex function, by $\varphi(.)$ a concave function, by $\psi(.)$ a linear function and by $g(.)$ a monotone increasing function.

Consider Figure 8.4. If $f(x)$ is a convex function, then the shortest path based on x is not necessarily the shortest path for $f(x)$. For example, suppose that $f(x) = e^x$ and $x_1 = 2$, $x_2 = 2$, $x_3 = 3$. Then, the shortest path from a to c is $a - c$ for x, but $a - b - c$ for $f(x)$.

Likewise, if $\varphi(x)$ is a concave function, then the shortest path based on x is not necessarily the shortest path for $\varphi(x)$, e.g. $\varphi(x) = \log(x)$ and $x_1 = 1.2$, $x_2 = 1.2$, $x_3 = 2.2$. In that case, the shortest path from a to c is $a - c$ for x, but $a - b - c$ for $\varphi(x)$.

If a linear function $\psi(x) = ax + b$ is used, then the shortest path based on x will also be the shortest path for $\psi(x)$ if $a > 0$ and $b = 0$. If $b \neq 0$, then the shortest path

based on x may differ from the shortest path based on $\psi(x)$.

In the rest of this subsection we consider graphs, for which all link weights are a function of a common link weight. Each link i has a weight vector $\vec{w} = \begin{bmatrix} f_1(x_i) \\ \vdots \\ f_m(x_i) \end{bmatrix}$, where x_i is the common link parameter (links may have different x_i and different f_j). In the sequel, this graph is referred to as G_w. We also introduce the graph G_x, which is identical in structure to G_w, but for which the links only have weight x_i.

Let P_x be the shortest path from source s to destination t in G_x, then

$$w(P_x) = \sum_{i \in P_x} x_i \leq w(P) = \sum_{i \in P} x_i$$

where P is any other path ($\neq P_x$) from s to t in G_x. Let $\varphi(x)$ be a concave function, then

$$\varphi(\frac{1}{h} \sum_{i=1}^{h} x_i) \geq \frac{1}{h} \sum_{i=1}^{h} \varphi(x_i)$$

where h is the hop count of a path P.

Property 56 *If the weight vector of a link,* $\vec{w} = \begin{bmatrix} \varphi_1(x_i) \\ \vdots \\ \varphi_m(x_i) \end{bmatrix}$ *with* $\varphi_j(x_i)$ *concave functions, is a function of a single parameter* x_i *and if* P *is the shortest path from* s *to* t *in* G_x *with length* $X = \sum_{i=1}^{h} x_i$ *and hop count* h*, then* P *in* G_w *satisfies the constraint vector* $\vec{L}$ *if*

$$X \leq h\varphi_j^{-1}\left(\frac{L_j}{h}\right), \quad 1 \leq j \leq m \tag{8.5}$$

Proof. The constraints are obeyed if $\sum_{i \in P} \varphi_j(x_i) \leq L_j$. Since φ_j are concave functions:

$$\sum_{i=1}^{h} \varphi_j(x_i) \leq h\varphi_j\left(\frac{1}{h} \sum_{i=1}^{h} x_i\right) \leq L_j$$

or,

$$\varphi_j\left(\frac{1}{h} \sum_{i=1}^{h} x_i\right) \leq \frac{L_j}{h}$$

Hence,

$$X = \sum_{i=1}^{h} x_i \leq h\varphi_j^{-1}\left(\frac{L_j}{h}\right)$$

■

Note that although P is the shortest path in G_x, this does not mean that P is also the shortest path in G_w (there may be another path P' for which $\sum_{i\in P'} \varphi(x_i) < \sum_{i\in P} \varphi(x_i)$). Equation (8.5) is a sufficient, but not a necessary condition, because there may be a path that does not obey (8.5) but still satisfies the constraints.

Property 57 *If the weight vector of a link,* $\vec{w} = \begin{bmatrix} f_1(x_i) \\ \vdots \\ f_m(x_i) \end{bmatrix}$ *with* $f_j(x)$ *convex functions, is a function of a single parameter* x_i *and if* P *is the shortest path from* s *to* t *in* G_x *with length* $X = \sum_{i=1}^{h} x_i$ *and hop count* h*, then* P *(and therefore all paths) violates the constraints in* G_w *if*

$$X > hf_j^{-1}\left(\frac{L_j}{h}\right) \tag{8.6}$$

for at least one j*.*

Proof. On the convexity,

$$hf_j\left(\frac{1}{h}\sum_{i=1}^{h} x_i\right) = hf_j\left(\frac{X}{h}\right) \leq \sum_{i=1}^{h} f_j(x_i)$$

The j-th constraint is violated if $\sum_{i=1}^{h} f_j(x_i) > L_j$, which is the case if $hf_j\left(\frac{X}{h}\right) > L_j$, which is equivalent to (8.6). ■

Property 58 *If the weight vector of a link* $\vec{w} = \begin{bmatrix} g_1(x_i) \\ \vdots \\ g_m(x_i) \end{bmatrix}$ *with* $g_j(x_i)$ *monotone increasing and* P *is the shortest minimum-hop path from* s *to* t *in* G_x *and* $x_i \leq x_i'$*, where* x_i' *is the* i*-th ordered common link weight of an other path* P' *from* s *to* t *in* G_x*, then* P *is also the shortest path in* G_w*.*

Proof. The property is a corollary from Theorem 107 from [70]: suppose that the sets (a) and (a') are arranged in descending order of magnitude. Then a necessary and sufficient condition that $g(a_1') + ... + g(a_n') \leq g(a_1) + ... + g(a_n)$ should be true for all continuous and increasing g is that $a_v' \leq a_v$ $(v = 1, 2, ..., n)$. ■

8.3.2 Simulation results

In this section we will evaluate the complexity of QoS routing through simulations. The simulation results will be based on several classes of graphs, namely the class of random graphs, the class of two-dimensional lattices, the class of power-law graphs and the chain topology. The class of random graphs is of the type $G_p(N)$ [21], where p is the expected link-density ($p = 0.2$). We only consider square two-dimensional lattices. For the class of Internet-like power-law graphs [48], we have chosen the power $\alpha = 2.4$ in the nodal degree distribution $\Pr[d = k] \sim k^{-\alpha}$. The chain topologies were of a triangular shape (like depicted in Figure 8.4). We have simulated with three different distributions for the $m = 2$ link weights, namely the uniform, exponential and Gaussian distributions. However, only the simulation results for correlated uniformly distributed link weights $\in [0, 1]$ with correlation coefficient[2] ρ [120] are presented, because they led to a higher complexity than the exponential and Gaussian distributions. All simulations consisted of generating 10^5 different graphs and in each graph a path has been computed via the SAMCRA algorithm (see Section 4.6). SAMCRA does not only exactly solve the MCP problem, but also exactly solves the MCOP problem by finding the optimal path within the constraints. Since the MCOP problem is more difficult than the MCP problem, the simulation results presented here should be interpreted as an upper bound. We have simulated a worst-case scenario by choosing the constraints so large that all paths can satisfy the constraints. Therefore, SAMCRA must search in the largest search space possible (all non-dominated paths between the source and destination), for the optimal path. If SAMCRA was only solving the MCP problem, choosing such large constraints would make the MCP problem "easy," because then any path is a solution to the MCP problem. During all simulations, we kept track of the minimum queue-size ($k_{\min}$) needed to find a feasible path. This queue-size can grow as a factorial in the worst-case and presents our measure for complexity in QoS routing. If TAMCRA, the polynomial-time predecessor of SAMCRA, had used this particular $k_{\min}$ under the same conditions, it would have found the same optimal path as SAMCRA did. If a smaller queue-size had been used, TAMCRA would not have been able to find the optimal path.

As illustrated in Figures 8.5-8.7, the results for the class of random graphs, do not display any intractability. We can see that a positive correlation leads to a slightly higher $E[k_{\min}]$ than with a negative correlation. This peculiar phenomenon has only been observed in the class of random graphs, with correlated uniformly distributed link weights. An explanation can be found by looking at Figure 8.8. Figure 8.8 shows that a positive correlation between the link weights may induce a higher expected hop count. When the link weights become more positively correlated, the weights become similar, and the problem approaches the $m = 1$ case. Since, the expected hop count of the m-dimensional shortest paths approach the minimum hop count if m grows to infinity

[2] We have verified that the correlation coefficient ρ' of the generated random variables equals the desired ρ.

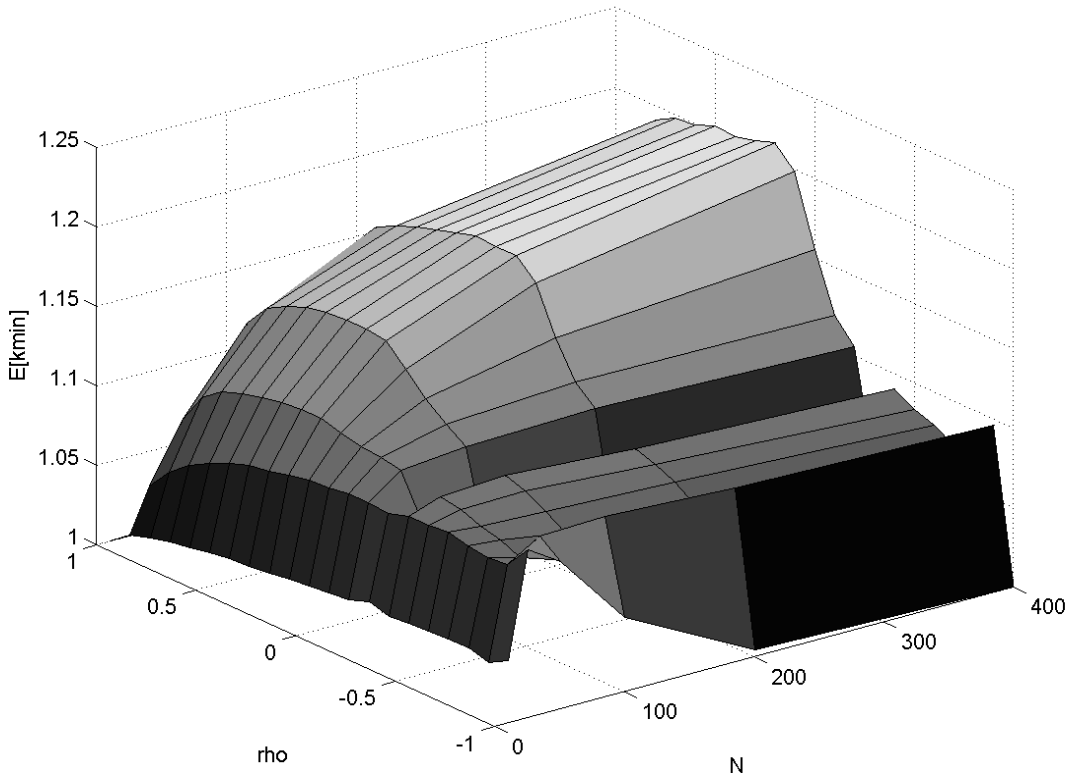


Figure 8.5: Expected queue-size for the class $G_p(N)$, with $m = 2$ uniformly distributed correlated link weights, as a function of the number of nodes N and the correlation coefficient ρ.

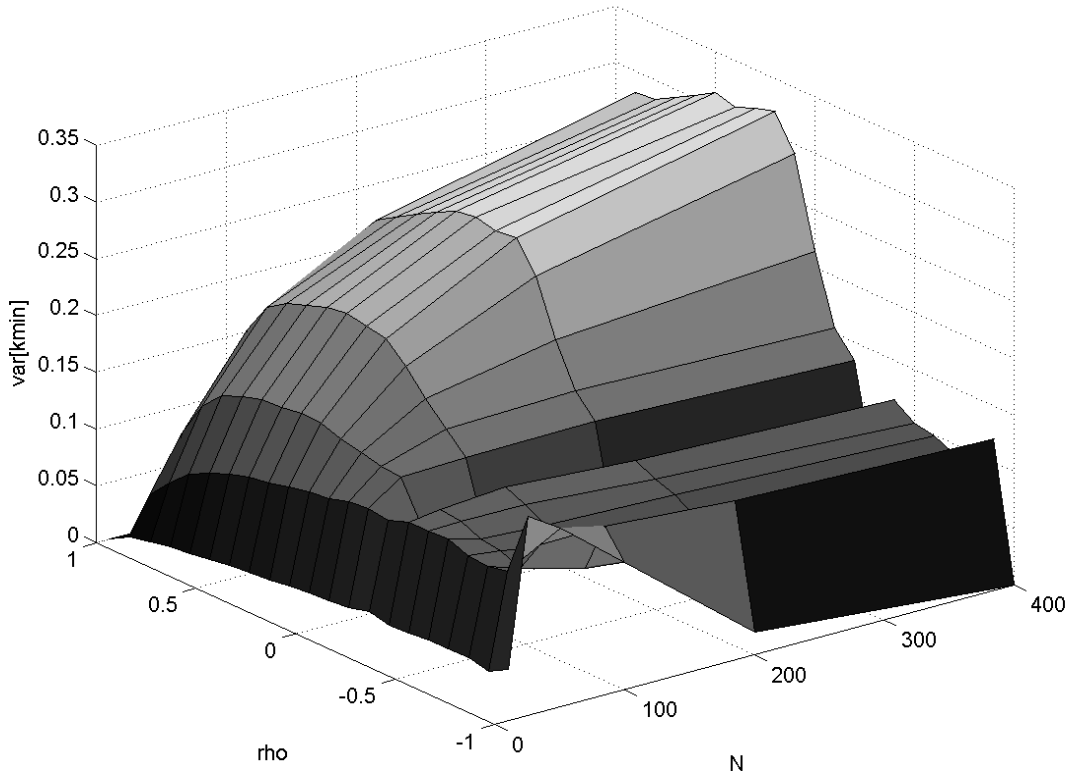


Figure 8.6: Variance in queue-size for the class $G_p(N)$, with $m = 2$ uniformly distributed correlated link weights, as a function of the number of nodes N and the correlation coefficient ρ.

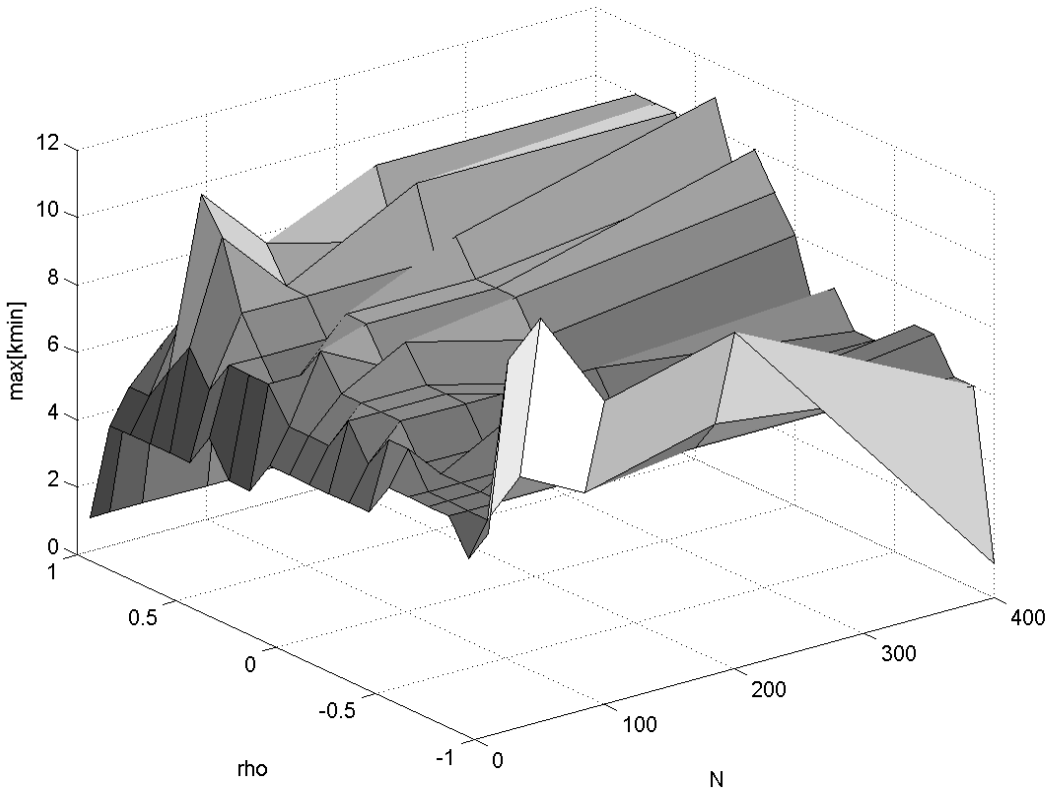


Figure 8.7: The maximum observed queue-size in the class $G_p(N)$, with $m = 2$ uniformly distributed correlated link weights, as a function of the number of nodes N and the correlation coefficient ρ.

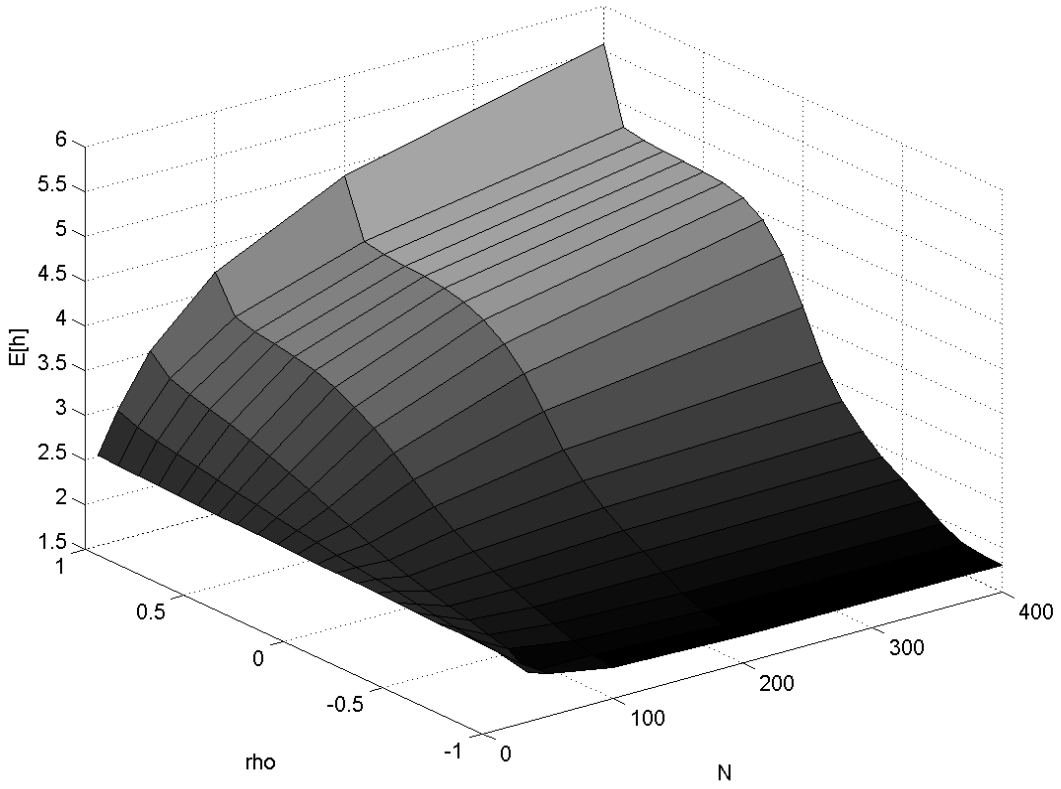


Figure 8.8: The expected hopcount for the class $G_p(N)$, with uniformly distributed correlated link weights, as a function of the number of nodes N and the correlation coefficient ρ.

[163], the $m = 1$ case is expected to have the largest hop count. A negative correlation between the link weights also leads to shorter paths, in terms of hop count. A low hop count is possible because there are sufficiently many paths in $G_p(N)$, which can be viewed as a thinning of a complete graph provided $p > \frac{\ln N}{N}$. For negative correlated link weights, a small link weight component is likely accompanied with a large one. For perfect negatively correlated link weight components ($\rho = -1$), SAMCRA's shortest length path (4.3) compensates outliers in the link weight components, such that (one or two) links with weight components close to $\frac{1}{2}$ are selected, which leads to the observed minimum-hop paths.

In general, the more hops that are traversed to find the shortest path, the more (sub)-paths must be evaluated and the more complex the computation becomes. We believe

that one of the measures for the "computational complexity" of a class of topologies is the expected (minimum) hop count of an arbitrary path in that topology. The expected hop count (for $m = 1$) scales as $O(\log N)$ in a random graph, while as $O(\sqrt{N})$ in a two-dimensional lattice and $O(N)$ in the chain topology. Besides the expected hop count in a graph, also the number of paths between a source and destination can provide a measure for the "computational hardness" of a class of topologies. The class of random graphs with $p = 0.2$ and N increasing, has an increasing number of paths and an increasing average nodal degree, giving the graph a small diameter (i.e., the source and destination are directly linked or a few hops apart). This can be interpreted from Figure 8.8. Figure 8.9 gives the expected queue-size for three different classes of graphs, namely the random graphs ($p = 0.2$), the two-dimensional lattices and the Internet-like power-law graphs (with power $\alpha = 2.4$). For all three classes of graphs, the source and

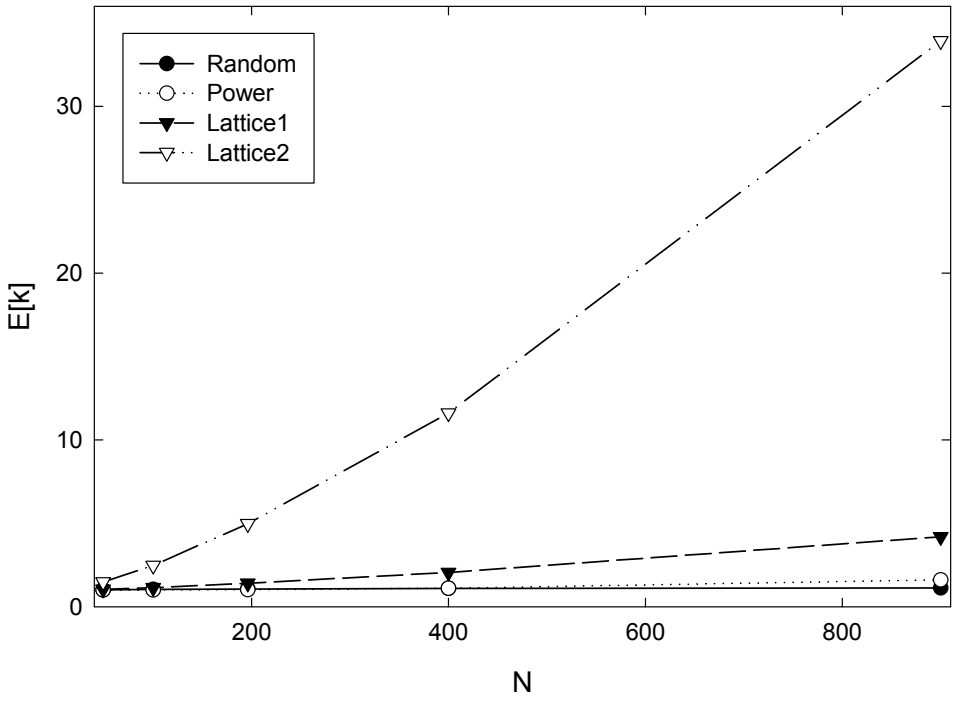


Figure 8.9: The expected queue-size for different topology classes as a function of the number of nodes N, with $m = 2$ independent ($\rho = 0$) uniformly distributed link weights.

destination nodes were chosen randomly. Only for the class of two-dimensional lattices "Lattice2," the source and destination nodes were chosen in opposite corners, to attain the largest minimum hop count. In the class of random graphs $G_p(N)$, although the

number of paths is large, the expected hop count is small, leading to a small complexity. For the extreme regular class Lattice2, the number of paths and the expected hop count is large, which leads to a large complexity. The class of power-law graphs may be considered, in terms of randomness, to lie between the random graphs and the two-dimensional lattices. The power-law graphs with $\alpha = 2.4$ have a moderate expected hop count and a small number of paths and lie, in terms of complexity, closer to the class of random graphs than to the class of two-dimensional lattices. We have also simulated with different link weight distributions, namely Gaussian and exponentially distributed correlated link weights. When using exponentially distributed correlated link weights, the first weight has a higher probability of being small, than with a uniform distribution. With a uniform distribution, each value for the first weight is equiprobable. Therefore, with exponentially (and also Gaussian) distributed correlated link weights, there is a higher probability that the link weight vectors are similar. For uniformly distributed link weights there is a larger variability, leading to a somewhat worse performance than in the exponential (or Gaussian) case. However, in all cases the expected queue-size in the class of random graphs was close to one, leading to a complexity similar to that of Dijkstra's algorithm. These simulation results therefore suggest that, irrespective of the link weight structure, QoS routing in the class of random graphs (and according to [162] also Waxman graphs) is possible in polynomial time. In contrast, the regularity and large expected hop count in the class of two-dimensional lattices, may provide ground for worst-case behavior. Indeed, we can observe a tendency towards exponential growth in Figure 8.10 and true exponential growth in Figure 8.11.

Because the chain topology was used in the proof that the MCP problem is NP-complete, we have also evaluated the performance of SAMCRA in chain topologies. The results are plotted in Figures 8.12 and 8.13.

Our simulation results[3] indicate that in the class of two-dimensional lattices and chain topologies, there is hardly any worst-case behavior for the entire range of correlation coefficient ρ, except for extreme negative values. Recall that the NP-proof is based on an extreme negative link correlation. We doubt that in practice link weights will display such a negative correlation, suggesting that exact QoS routing in practice, irrespective of the underlying topology, is possible in polynomial time.

8.3.3 Inter-link correlation

Inter-link correlation refers to the correlation amongst weights over multiple links. Because a graph has M links, M weights have to be correlated to each other. Even for Gaussian random variables this is a complex task, so only this Gaussian distribution is investigated.

The Gaussian random variable X is defined for all x by the probability density

[3] Recall that the simulation results reflect the complexity of the much more difficult MCOP problem.

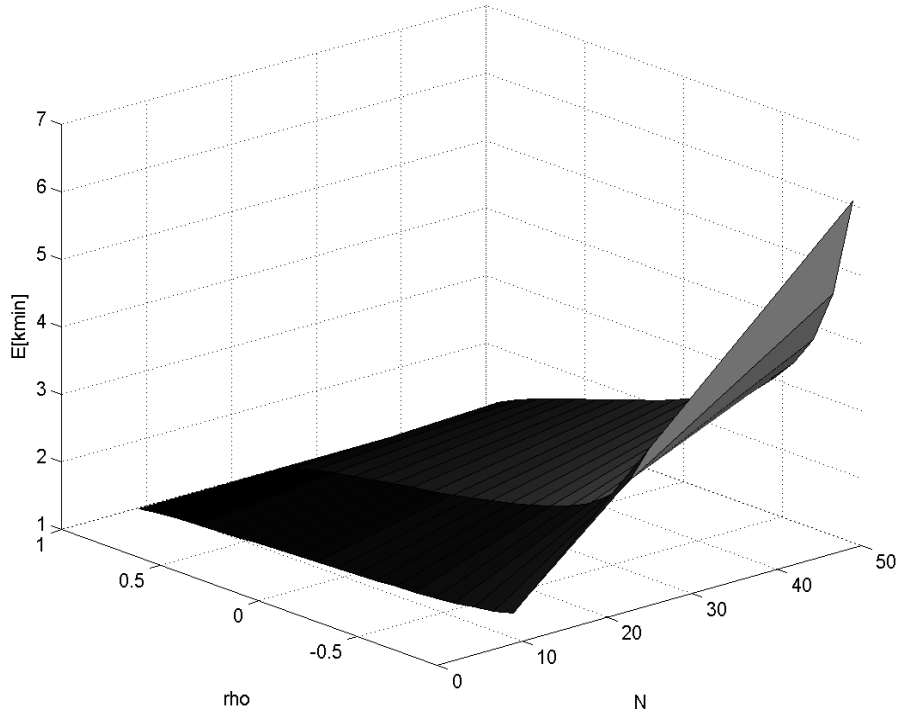


Figure 8.10: The expected queue-size in the class of two-dimensional lattices as a function of the number of nodes N and correlation coefficient ρ. The $m = 2$ link weights were uniformly distributed and the source and destination nodes were chosen in opposite corners.

function

$$f_X(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left[-\frac{(x-\mu)^2}{2\sigma^2}\right]$$

where μ gives the average and σ^2 the variance (often the notation $N(\mu, \sigma^2)$ is used).

M random variables $X = X_1, X_2, ..., X_M$ are called jointly Gaussian if their joint probability density function equals

$$f_{X_1,X_2,...,X_M}(x_1, x_2, ..., x_M) = \frac{1}{|C|^{1/2}\sqrt{(2\pi)^M}} \exp\left[-\frac{[x - \mathbb{E}[X]]^T[x - \mathbb{E}[X]]}{2C}\right]$$

where $[x - \mathbb{E}[X]] = \begin{bmatrix} x_1 - \mathbb{E}[X_1] \\ : \\ x_M - \mathbb{E}[X_M] \end{bmatrix}$ and C is an $M \times M$ covariance matrix with elements $C_{ij} = \mathbb{E}[(X_i - \mathbb{E}[X_i])(X_j - \mathbb{E}[X_j])]$. The covariance matrix (or correlation

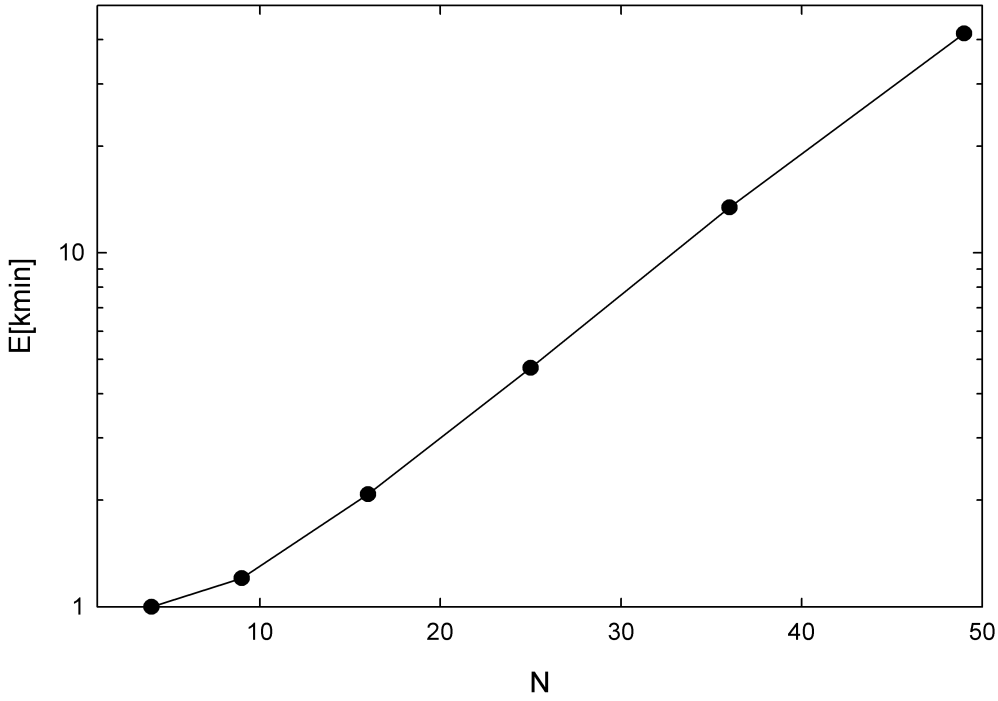


Figure 8.11: The expected queue-size in the class of two-dimensional lattices as a function of the number of nodes N, with correlation coefficient $\rho = -1$. The $m = 2$ link weights were uniformly distributed and the source and destination nodes were chosen in opposite corners.

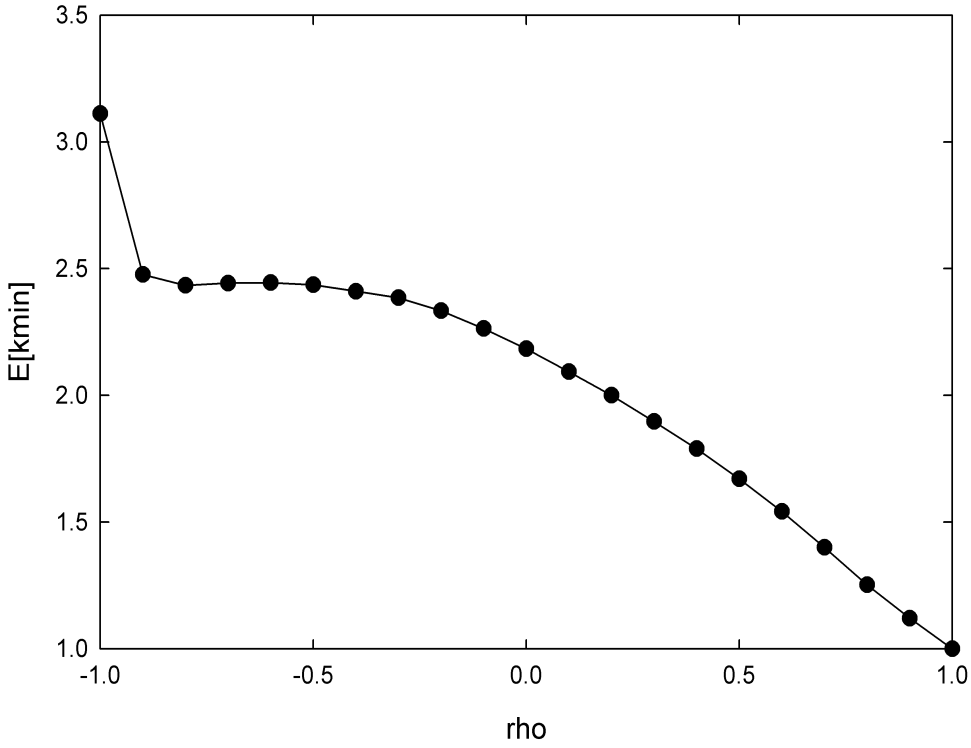


Figure 8.12: The expected queue-size in the chain topology, with $m = 2$ correlated uniformly distributed link weights for $N = 50$, as a function of the correlation coefficient ρ.

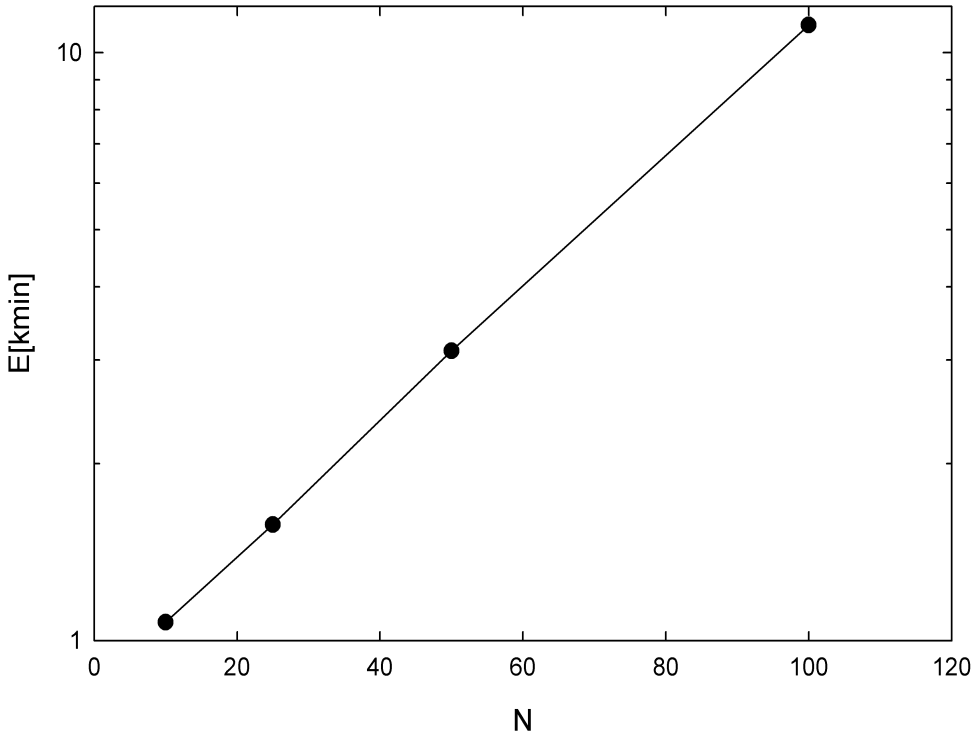


Figure 8.13: The expected queue-size in the chain topology, with $m = 2$ correlated ($\rho = -1$) uniformly distributed link weights as a function of the number of nodes N.

matrix if divided by the variances) specifies the correlation among the M variables. For C to be a valid covariance matrix it has to be a positive definite symmetric matrix, i.e. a real-valued, symmetric matrix with positive eigenvalues. Hence, it must be possible to express C in the form $C = \Gamma\Lambda\Gamma^T$, where Λ is a diagonal matrix that consists of the eigenvalues of C and Γ is an orthogonal matrix with the eigenvectors of C as columns.

It is possible to generate a vector $Y = AX$ of jointly Gaussian random variables with covariance $C_Y = AA^T = (\Gamma\Lambda^{1/2})(\Lambda^{1/2}\Gamma)^T$ and X a vector of zero-mean, unit-variance uncorrelated Gaussian random variables.

By means of simulations we have investigated whether inter-link correlation alone (i.e., with intra-link correlation $\rho_{intra} = 0$) can lead to worst-case scenarios. The difficulty in such simulations is to find a valid and "hard" covariance matrix. In [87] we have utilized several approaches, where we generated various types of (regular and random) covariance matrices and checked if they were indeed positive definite via Cholesky decomposition. In cases where Cholesky decomposition failed, the nearest positive definite covariance matrix was returned. The specific classes of matrices that were used can be found in [87]. Another approach was to directly generate valid covariance matrices via the random generation of Γ and Λ in $C = \Gamma\Lambda\Gamma^T$. Our last approach consisted of manually correlating links to each other. All simulations were conducted on square lattices and the results can be found in [87]. However, in all simulations for which $\rho_{intra} = 0$, the inter-link correlation as induced by the covariance matrices did not cause exponential running times. This suggests that inter-link correlation alone cannot establish a worst-case scenario. Unfortunately this claim cannot be substantiated without examining all possible types of covariance matrices. Since this is impossible, we have tried to vary our choice of covariance matrices as much as possible.

8.4 The impact of constraints on complexity

In this section we analyze the influence of the constraints on the complexity of the MCP problem. For this purpose, we will initiate an evaluation of a phase transition [26], [74] in the MCP problem.

8.4.1 Theory

Property 59 *Let $P_{s-t;i}$ denote the one-dimensional shortest path from source s to destination t for which $w_i(P_{s-t;i}) \leq w_i(P^*)$, $\forall P^*$. Then, the MCP and MCOP problems are not NP-complete when*

$$L_i < w_i(P_{s-t;i}) \tag{8.7}$$

for at least one constraint.

Proof. $P_{s-t;i}$ is the path with the shortest i-th weight $w_i(P_{s-t;i})$. Therefore, $w_i(P_{s-t;i})$ is a lower bound on the i-th weight $w_i(P_{s-t})$ that any path P_{s-t} between s and t can

attain. Thus, if for any constraint i holds that $L_i < w_i(P_{s-t;i})$, then no path P_{s-t} can obey L_i. Since $P_{s-t;i}$ can be found in polynomial time (e.g., via the Dijkstra algorithm), the MCP problem is solvable (it is verified that no solution exists) in polynomial time if any constraint obeys (8.7). ■

Property 60 *Let $P_{s-t;i}$ denote the one-dimensional shortest path from source s to destination t for which $w_i(P_{s-t;i}) \leq w_i(P^*)$, $\forall P^*$. Then, the MCP problem is not NP-complete when*

$$L_i \geq \max_{j=1,...,m} \left(w_i \left(P_{s-t;j} \right) \right) \tag{8.8}$$

for at least $m-1$ constraints.

Proof. If $L_i \geq \max_{j=1,...,m} \left(w_i \left(P_{s-t;j} \right) \right)$ for all m constraints, then all m one-dimensional shortest paths $P_{s-t;i}$ (for $i = 1, ..., m$) obey the constraints. Hence any path $P_{s-t;i}$ can be chosen as a feasible path.

If $L_i \geq \max_{j=1,...,m} \left(w_i \left(P_{s-t;j} \right) \right)$ for $m-1$ constraints (say $i = 1, ..., m-1$) and $L_i < \max_{j=1,...,m} \left(w_i \left(P_{s-t;j} \right) \right)$ for one constraint ($i = m$), then if $L_m \geq w_m(P_{s-t;m})$ path $P_{s-t;m}$ obeys all m constraints. If $L_m < w_m(P_{s-t;m})$, then according to property 59 no feasible path exists. Since the paths $P_{s-t;i}$ can be found in polynomial time (e.g., via the Dijkstra algorithm), the MCP problem is solvable in polynomial time if at least $m-1$ constraints obey (8.8). ■

For $m = 2$, properties 59 and 60 constitute a closed NP-complete range

$$L_i < w_i(P_{s-t;i}) < \max_{j=1,...,m} \left(w_i \left(P_{s-t;j} \right) \right) \tag{8.9}$$

The MCP problem with $m = 2$ is only NP-complete if both constraints lie in the NP-complete range (8.9). When the link weights are positively correlated, the NP-complete range (8.9) will be smaller than when the link weights are negatively correlated. This is illustrated in Figure 8.14 for $m = 2$. At the cost of increased (polynomial-time) complexity, a further reduction of the NP-complete range can be gained by using property[4] 61.

Property 61 *Let P_{s-t} denote the path from source s to destination t for which $\sum_{i=1}^{m} \alpha_i w_i(P_{s-t}) \leq \sum_{i=1}^{m} \alpha_i w_i(P^*_{s-t})$, $\forall P^*_{s-t}$. Then, if*

$$\sum_{i=1}^{m} \alpha_i L_i < \sum_{i=1}^{m} \alpha_i w_i(P_{s-t})$$

where $\alpha_i \geq 0$ with an inequality for at least one i, then there is no feasible path present that can solve the MCP or MCOP problem.

[4] We have not programmed property 61 into our simulations.

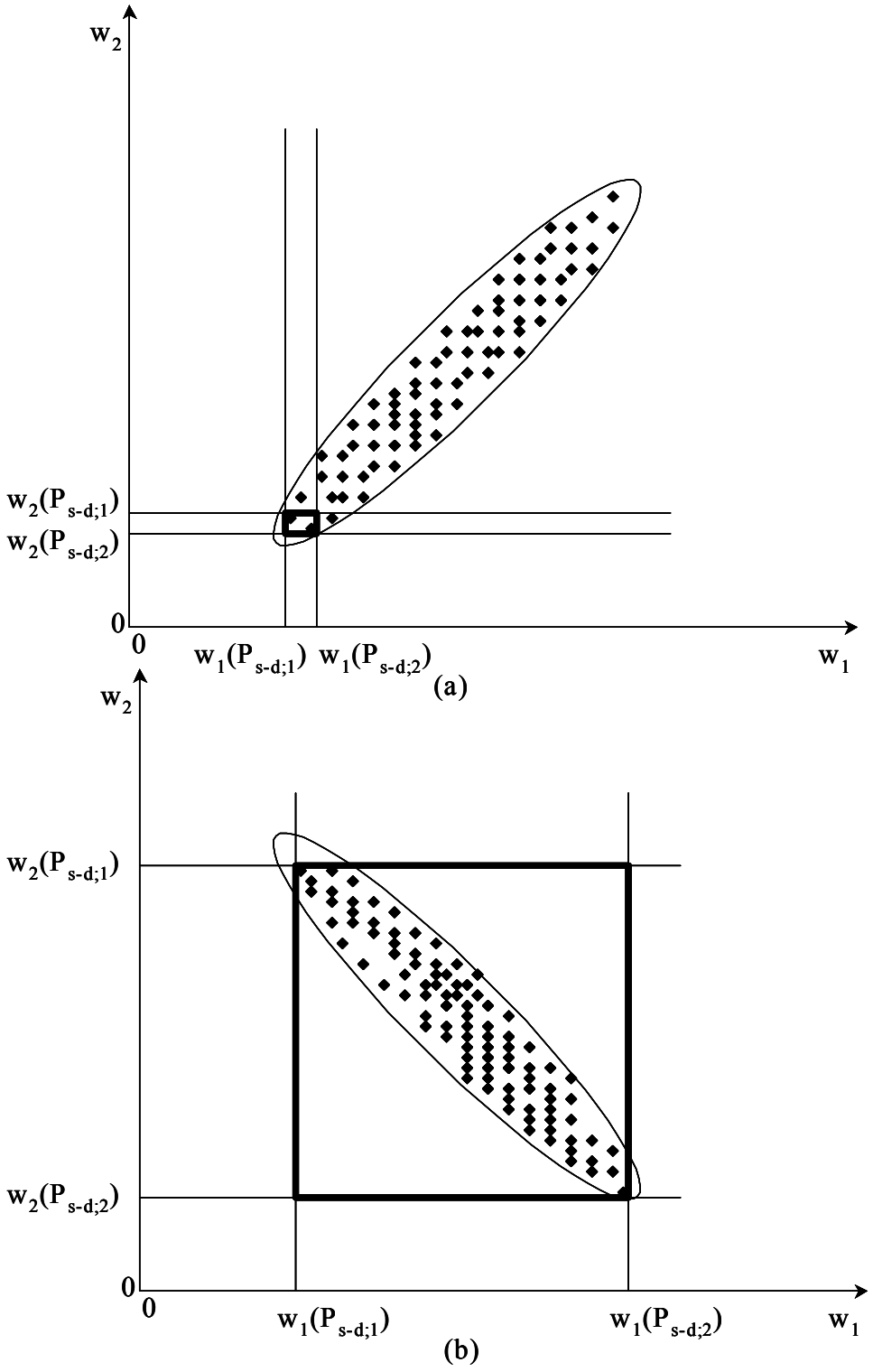


Figure 8.14: The constraints range (bold rectangle) for (a) positive correlation and (b) negative correlation. The dots in the figure denote paths in the two-dimensional space ($m = 2$).

Proof. A proof by contradiction. Assume that P_{s-t} denotes the path from source s to destination t for which $\sum_{i=1}^{m} \alpha_i w_i(P_{s-t}) \leq \sum_{i=1}^{m} \alpha_i w_i(P^*_{s-t}), \forall P^*_{s-t}$ and that $\sum_{i=1}^{m} \alpha_i L_i < \sum_{i=1}^{m} \alpha_i w_i(P_{s-t})$. If a path P^*_{s-t} would exist that obeys the constraints, then $\sum_{i=1}^{m} \alpha_i w_i(P^*_{s-t}) \leq \sum_{i=1}^{m} \alpha_i L_i$, for $i = 1, ..., m$ and thus $\sum_{i=1}^{m} \alpha_i w_i(P^*_{s-t}) \leq \sum_{i=1}^{m} \alpha_i L_i < \sum_{i=1}^{m} \alpha_i w_i(P_{s-t})$, which contradicts the assumption that $\sum_{i=1}^{m} \alpha_i w_i(P_{s-t}) \leq \sum_{i=1}^{m} \alpha_i w_i(P^*_{s-t}), \ \forall P^*_{s-t}$. Since the path P_{s-t} can be found in polynomial time (e.g., via the Jaffe algorithm [85]), the MCP problem is solvable in polynomial time if $\sum_{i=1}^{m} \alpha_i L_i < \sum_{i=1}^{m} \alpha_i w_i(P_{s-t})$. ■

The work presented in Section 8.1 suggests that there is a connection between worst-case complexity and phase transitions. Using the terminology of Gent and Walsh [58], if problems are very under-constrained, then it is usually easy to find one of the many solutions. When problems are very over-constrained, it is usually easy to determine that they are insoluble. In the phase transition in between, problems are "critically constrained" and it is typically very hard to determine if they are soluble or insoluble. Applied to the MCP problem, a phase transition appears based on the values of the constraints. If one of the constraints obeys (8.7), the probability of finding a path obeying the constraints is zero. Moreover, it can be verified in polynomial time, that there exists no path in the graph that obeys the constraints (property 59). On the other hand, if the values of the constraints are very large (under-constrained), such that all constraints follow (8.8), then a path satisfying these large constraints can be found in polynomial time. A phase transition is therefore expected to occur if the constraints do not obey (8.7) and (8.8). For small values of $L_i = w_i\left(P_{s-t,i}\right) + \epsilon$ (with $\epsilon > 0$) the MCP problem may still be insoluble, however the effort (complexity) needed to verify that indeed no feasible path is present in the graph has increased. In contrast to the case where the constraints $L_i < w_i\left(P_{s-t,i}\right)$, only computing the m Dijkstra shortest paths is not sufficient to determine that the problem is insoluble. The SAMCRA algorithm (or another exact MCP routing algorithm) must be invoked and will eventually observe that no path can obey the constraints. The larger the constraints become, the longer it will take to determine that no feasible path exists. Hence, increasing the constraints until a feasible path emerges, augments the complexity of its solution. On the other hand, when decreasing the constraints starting from the upper boundary (8.8), first many paths will obey the constraints $L_i = \max_j(w_i\left(P_{s-t,j}\right)) - \epsilon$ leading to a high probability that a feasible path will be found fast. If the values of the constraints decrease, the probability of finding a feasible path fast will also decrease. It is therefore expected that a phase transition occurs if there are only a few (if any) feasible paths present. In this case MCP $\approx$ MCOP. The steepness of the phase transition depends on the range between (8.7) and (8.8), which is heavily influenced by the correlation coefficient ρ as illustrated in Figure 8.14 (and by the computations in Appendix A.1). As discussed in Section 8.3, the correlation coefficient also impacts the level of complexity, which decreases if ρ increases.

8.4.2 Simulation results

To be able to observe a phase transition, we must choose an intractable configuration. The simulation results in the previous section suggest that the graphs should contain many paths, have a large expected hop count and the link weights should have a negative correlation. All these properties are present in the class of two-dimensional lattices with correlation $\rho = -1$. The remainder of this chapter confines to this class of lattices and tries to distinguish a phase transition via simulations and an approximate analysis. For our simulations, we have chosen to use a single two-dimensional lattice with $N = 49$ nodes and correlated uniformly distributed link weights in the range [0,1]. A worst-case scenario is obtained if the source node is positioned in the upper left corner and the destination node in the lower right corner, causing the largest minimum hop count. For each constraint L_1 and L_2, 100 different values were chosen in the NP-complete range (8.9), as discussed above, leading to a total of 10^4 iterations, all in the same lattice. Because we are examining the MCP problem, we have chosen to simulate with HAMCRA [102] instead of SAMCRA [167]. Figure 8.15 displays the maximum queue-size[5] k used by HAMCRA. The corresponding contour plot is given in Figure 8.16.

Different constraints can lead to different m-dimensional shortest paths. For instance, if L_1 is small (e.g., 5.0 in Figure 8.15) and L_2 is large (e.g., 7.0 in Figure 8.15), then a path P obeying these constraints must also have a small weight $w_1(P) \leq L_1$ and the second weight may be large as long as $w_2(P) \leq L_2$. Since L_1 is slightly larger than the weight $w_1(P_{s-t;1})$ of the shortest Dijkstra path for measure 1, the path P may closely approximate $P_{s-t;1}$, which may be easy to find as indicated by small k values in Figure 8.15. Similarly, if L_1 is large (e.g., 9.0 in Figure 8.15) and L_2 is small (e.g., 3.0 in Figure 8.15), then a path P obeying these constraints may closely approximate the Dijkstra shortest path for measure 2 ($P_{s-t;2}$), which may also be easy to find (as verified in Figure 8.15). Figure 8.15 reveals that the complexity is largest when $L_1 = 6.94$ and $L_2 = 5.06$. These values are situated near the center of the rectangle (Figure 8.14) spanned by the NP-complete range (8.9) at $L_1^* = 7.09$ and $L_2^* = 4.91$. These observations seem to connote that the complexity is largest when the constraints closely approximate the weights of the m-dimensional shortest path P, which equal $\sqrt{N} - 1$ on average (see Equation (A.10)). For two-dimensional lattices of $N = 49$ nodes, the highest complexity is therefore expected at $L_1 = L_2 = 6$. The deviation in our case is caused by only examining one single lattice, instead of the many required for statistical results.

The sharp edge/line in Figure 8.16, constituted by the different shortest paths, can be attributed to the extreme negative correlation ($\rho = -1$) as explained in Figure 8.14b and Appendix A.1. Since the link weights are chosen in the range [0,1], there holds

[5] k is different from the previously used $k_{\min}$, since k denotes the maximum queue-size used, whereas $k_{\min}$ is the queue-size that TAMCRA would have needed to attain the exact solution. k is used here, because $k_{\min} = 0$ if there is no path present.

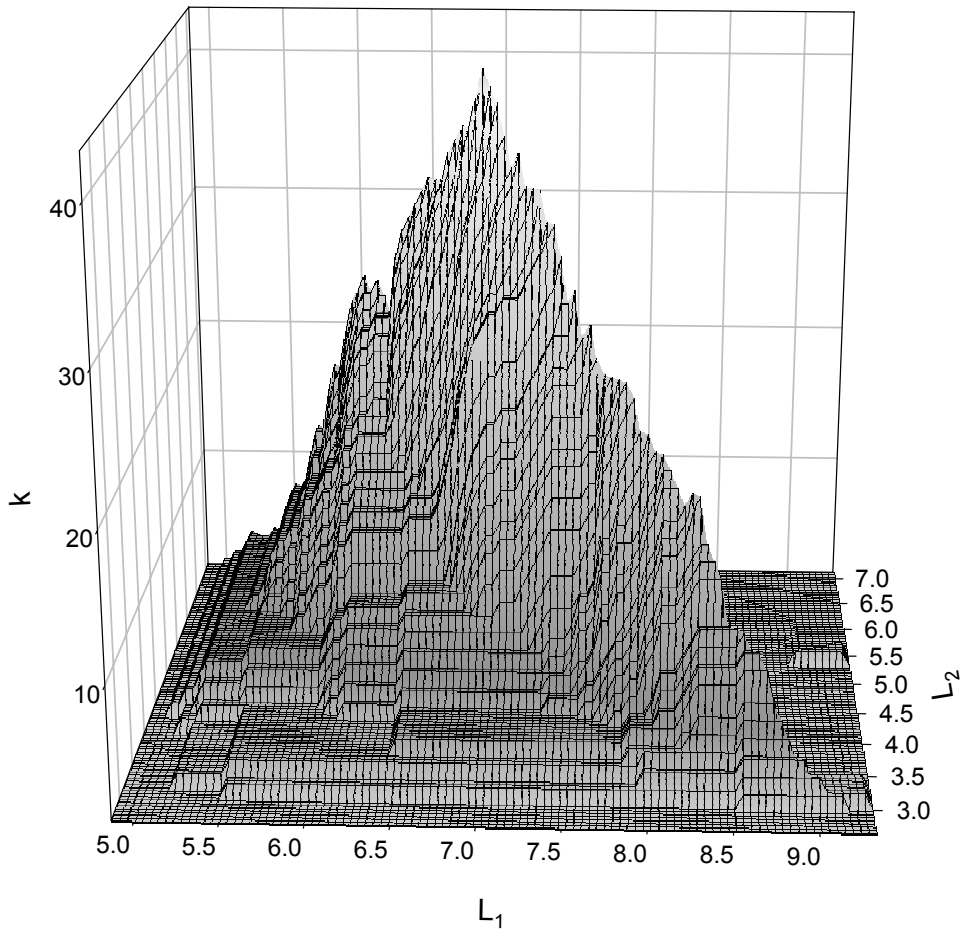


Figure 8.15: The queue-size in a two-dimensional lattice, with correlated uniformly distributed link weights, $N = 49$, $\rho = -1$, and 10^4 different constraint vectors.

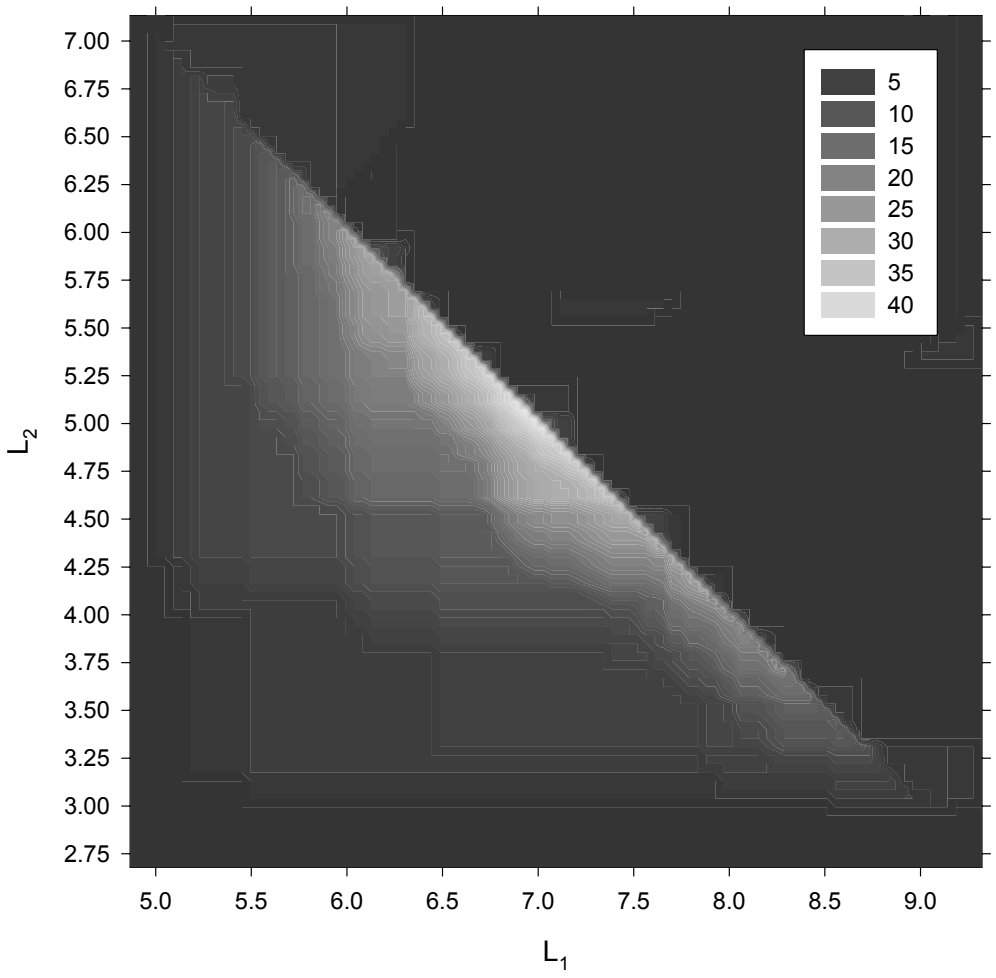


Figure 8.16: Contour plot of the queue-size in a two-dimensional lattice, with correlated uniformly distributed link weights, $N = 49$, $\rho = -1$ and 10^4 different constraint vectors.

that for $\rho = -1$, $w_1(u,v) = 1 - w_2(u,v)$, $\forall (u,v) \in E$. Hence, the path weights of any path P obey $w_1(P) = h - w_2(P)$, where $w_i(P) = \sum_{(u,v)\in P} w_i(u,v)$ and h equals the hop count of path P. If we again look at Figure 8.16, we may observe that the linear line, once continued, intersects both axes L_1 and L_2 at 12, which is precisely the minimum hop count of the two-dimensional lattice with 49 nodes. Moreover, since $w_1(P) = h - w_2(P)$, according to property 61, when $L_1 + L_2 < h$, then no feasible path exists. This means that for the class of two-dimensional lattices with correlated ($\rho = -1$) uniformly distributed link weights, the constraints must obey $L_1 + L_2 \geq h$, for a feasible path to be possible. This condition for the constraints can be checked in polynomial time and it is therefore possible to obtain a much steeper phase transition than observed in Figures 8.15 and 8.16. Finally, we have also simulated with independent uniformly distributed link weights ($\rho = 0$) in the range [0,1]. As discussed in Section 8.3, the complexity of solving the MCP and MCOP problems under independent link weights is smaller than with negatively correlated link weights. To observe a phase transition, we had to simulate with a lattice larger than $N = 49$. Figure 8.17 gives the contour plot for $N = 400$ and $\rho = 0$. The complexity is largest for $L_1 = 12.58$ and $L_2 = 15.11$.

It would be desirable to obtain an estimation of the size of the constraints that make the MCP problem critically constrained. Such an estimation would allow us to predict the location of the phase transition and hence give us an indication of the "critically constrained" region. The next subsection will provide an approximate analysis of the weights of the m-dimensional shortest path, because as seen above, choosing the constraints close to these weights may lead to a non-polynomial running time.

8.4.3 Estimation of the shortest path length in a lattice

This subsection discusses the approximate computation of the length of the m-dimensional shortest path between two corner points in a rectangular two-dimensional (2d) lattice with z_1 links vertically and z_2 links horizontally. The link weights are independent uniformly distributed in the range $(0, 1]$. The approximate analysis of the formulas presented in this subsection and some of the notation that is used, can be found in Appendix A.1. The asymptotic average weight of a $h = z_1 + z_2$ hop path in one dimension for a square lattice is given by (A.4) as $\mathbb{E}[W_1] \simeq \frac{h}{2e} \approx \frac{\sqrt{N}}{e}$. This estimate agrees reasonably well with simulations in the range $N \in [100, 1600]$, which accurately follow $\mathbb{E}[W_{sim}] \approx 0.6N^{0.48}$.

The extension to m dimensions, with independent link weight components ($\rho = 0$), for the average length $W_m = L_{eq} l_h$ is the approximation (A.8),

$$\mathbb{E}[W_m] \simeq \frac{h}{e 2^{\frac{1}{m}}}$$

The scaling $2^{-\frac{1}{m}}$ as a function of m has been observed in simulations, even for $N = 49$. This approximate analysis (A.7) shows that there is no shortest path obeying the

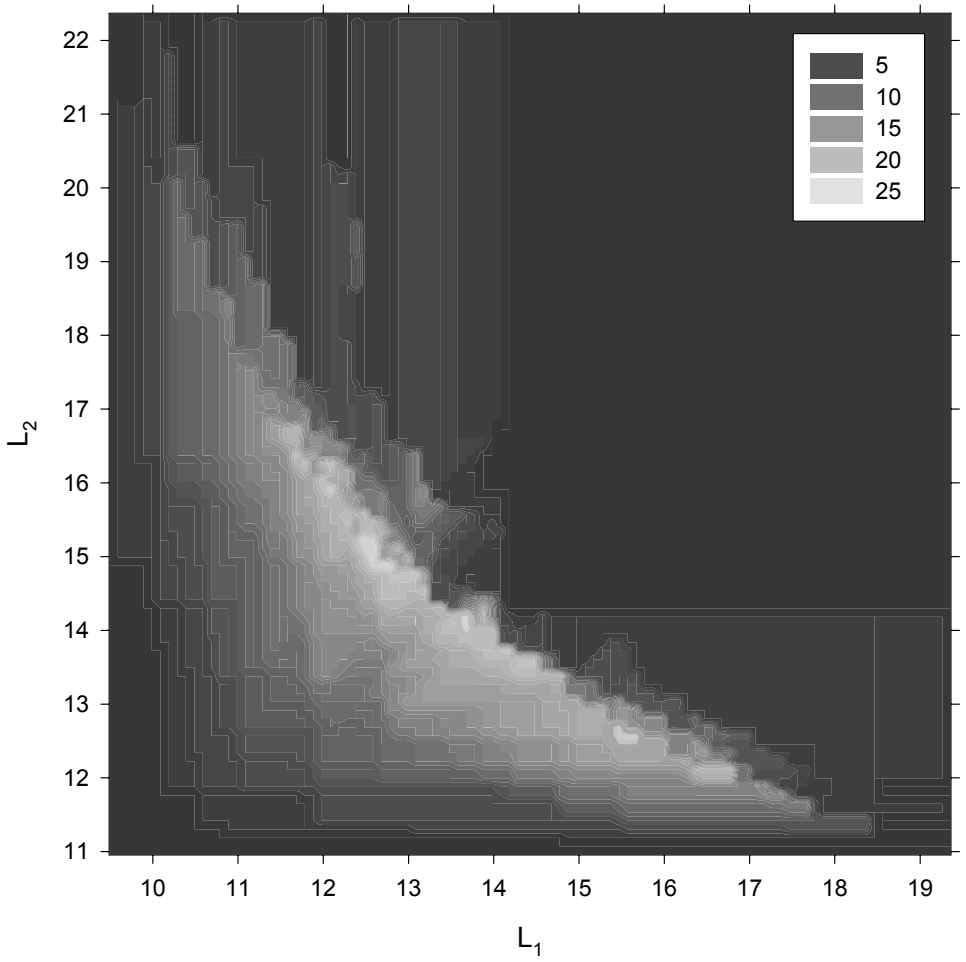


Figure 8.17: Contour plot of the queue-size in a two-dimensional lattice, with uniformly distributed link weights, $N = 400$, $\rho = 0$ and 10^4 different constraint vectors.

constraints if the length, as defined in (A.6), $l_h(P) > 1$. This event has probability

$$\Pr[l_h > 1] \approx \exp\left(-\frac{h!}{z_1!z_2!}\left(\frac{L_{eq}^h}{h!}\right)^m\right)$$

Clearly, if the lattice (i.e., z_1, z_2 and $h = z_1 + z_2$) is fixed and the constraints decrease (increase), all (no) paths violate the constraints. The fact that there exists a path within the constraints depends on the product of the constraints or equivalent constraint L_{eq}. If $\frac{L_{eq}^h}{h!} > 1$ or $L_{eq} > (h!)^{\frac{1}{h}} \approx \frac{h}{e}$ (for large h), nearly all paths obey the constraints. If $\frac{L_{eq}^h}{h!} < 1$ or $L_{eq} < (h!)^{\frac{1}{h}} \approx \frac{h}{e}$, for a large number m of constraints, no path obeys the constraints. Hence, for large m and large h, there seems to be a critical value of the equivalent constraint $L_{eq} > (h!)^{\frac{1}{h}} \approx \frac{h}{e}$ for which $\mathbb{E}[l_h] = \left(\frac{z_1!z_2!}{h!}\right)^{\frac{1}{mh}} < 1$ and specifically for the square lattice $\mathbb{E}[l_h] \approx 2^{-\frac{1}{m}}$. Below that value the shortest path behavior is clearly different than above that value, which points to a phase transition.

The result (A.9) in two dimensions ($m = 2$) with perfectly negative correlation ($\rho = -1$) even points to a more confining situation, as was readily observed by comparing Figures 8.16 and 8.17. Since $\mathbb{E}[L_{eq}l_h] \approx \frac{h}{2}$ (see (A.10)) and any random variable $L_{eq}l_h \geq \frac{h}{2}$, the average weight of the shortest path lies very close to the boundary $\frac{h}{2}$.

In summary, we have estimated the average length or weights of the shortest path for large values of h or, equivalently, the number of nodes N in the 2d-lattice. As common for extremal distributions, the variance is small, which implies a fast transition from 0 to 1 of $\Pr[L_{eq}l_h \leq y]$ around the average. The knowledge of the shortest path is important to set the constraints: if the constraints are close to $\mathbb{E}[L_{eq}l_h]$, the problem is critically constrained and more computations are needed to determine whether there exists a path obeying the constraints or not. For constraints larger or smaller than $\mathbb{E}[L_{eq}l_h]$, the problem is either under- or over-constrained and the verdict that there exist a path within the constraints is usually simple to draw with high probability. In the analysis presented in the Appendix, we have assumed that a possible overlap of h-hop paths is sufficiently weak to allow the application of the limit laws for independent random variables. Only relatively few paths will share a large number of links. We have used a heuristic argument to validate this assumption and have observed a good agreement with our simulation results. The second assumption is that the shortest path in the 2d-lattice has h hops or that $\Pr[\text{hops} > h]$ is negligibly small. This approximation is reasonable since simulations show that $\Pr[\text{hops} = h + 2k]$ is rapidly decaying in k with decay rate dependent on the size of the graph. The larger the graph, the slower the decay rate. However, for increasing m, simulations show that the shortest path tends to have h hops. Also for very negative correlation coefficients, the probability that shortest paths have h hops increases. Finally, although computed for uniformly distributed link weights, the same results hold for any distribution whose h-fold convolved distribution also behaves as $\frac{x^h}{h!}$ for small x. Any distribution in the same sphere of minimal attraction

(such as exponentially distributed link weights with mean 1) yields the same results.

8.5 Conclusions

In this chapter an evaluation of the complexity of Quality of Service (QoS) routing was presented. Finding a path based on multiple QoS constraints is proven to be an NP-complete problem. However, this Multi-Constrained Path (MCP) selection problem is not NP-complete in the strong sense, meaning that a pseudo-polynomial algorithm can exactly solve the problem. The NP-completeness of the MCP problem hinges on at least four factors, namely (1) the underlying topology, (2) link weights that can grow arbitrarily large or have an infinite granularity, (3) a very negative correlation among the link weights and (4) the values of the constraints. If the values of the constraints are very large then it is easy to find a path within the constraints. On the contrary, if the values of the constraints are very small, then it is easy to verify that there is no path within the constraints. This indicates that there will be a phase transition if the constraints are around the weights of the m-dimensional shortest path in the network. In this case, it is expected to be difficult to establish whether a feasible path exists. If the four above mentioned conditions are all necessary to induce intractability, they will allow network and service providers to properly dimension their network and to avoid intractable scenarios. Moreover, if the theory of phase transition holds for the MCP problem, then QoS requirements close to the m-dimensional shortest path will, if admitted, provide the highest possible level of QoS, but also the highest computational cost. Such information is invaluable for pricing and billing mechanisms and admission control algorithms. Finally, a proper understanding and use of the four conditions, will allow for efficient QoS routing at controlled computational costs.

Chapter 9

QoS dynamics

In this chapter a first step is made to understand the dynamics of QoS routing. QoS routing dynamics refer to the highly fluctuating QoS measures like available bandwidth. The influence of rapidly varying QoS measures on QoS routing is not yet well understood, but such an understanding is highly important for QoS routing. Therefore, although not the focus of this thesis, for completeness some attention is given to dynamic QoS routing. Our approach is twofold: first the stability of QoS-compliant paths is investigated, with the goal to obtain some feeling on "significant changes" in the link weights, i.e. how much must link weights change before the shortest QoS path is not shortest anymore. This information is invaluable in devising protocols that update the link state information in a network. Secondly, a performance evaluation of QoS algorithms in a dynamic network is conducted.

9.1 Introduction to QoS stability

To date, the Internet still lacks a (widely) working QoS architecture. Assuming the Internet infrastructure has enough resources to be able to provide QoS, then there are basically two problems that still hamper the introduction of QoS in the Internet. The first problem concerns the computation of QoS-compliant paths (the multi-constrained path (MCP) selection problem, see Chapter 1)), which is an NP-complete problem. This problem was studied in-depth in the previous chapters and we have conjectured that in practice the multi-constrained path selection problem is expected to be feasible. A second problem is that of accurately and efficiently maintaining, distributing and updating the dynamic QoS link weights. Since the number of packets are stochastically varying in a network, it is conceivable that there is a characteristic time scale for network dynamics. Monitoring any change along the Internet is simply not possible and even not desirable, because not all changes are important. Further, there is a topology range of interest: not all details of the entire global Internet are needed to determine a path

P from source s to destination t. A subnetwork encompassing s and t seems sufficient. When looking at the time-scale in a network topology as illustrated in Figure 9.1, we distinguish between changes that occur (1) infrequently and (2) frequently. The first kind reflects topology changes due to failures and the joining/leaving of nodes. In the current Internet, only this kind of topology changes is considered. Its dynamics are relatively well understood. The key point is that the time between two "first kind" topology changes is long compared to the time needed to flood this information across the whole network. Thus, the topology databases on which routing relies, converge rapidly (with respect to the frequency of updates) to the new situation. The transient period where the databases are not synchronized (which may cause routing loops and malfunctioning), is generally small.

The second type of rapidly varying changes are typically related to the consumption of resources or to the traffic flowing through the network. The link weight coupling to state information seriously complicates the dynamics of flooding, because the flooding convergence time T can be longer than the change rate Δ of some measure (such as available bandwidth).

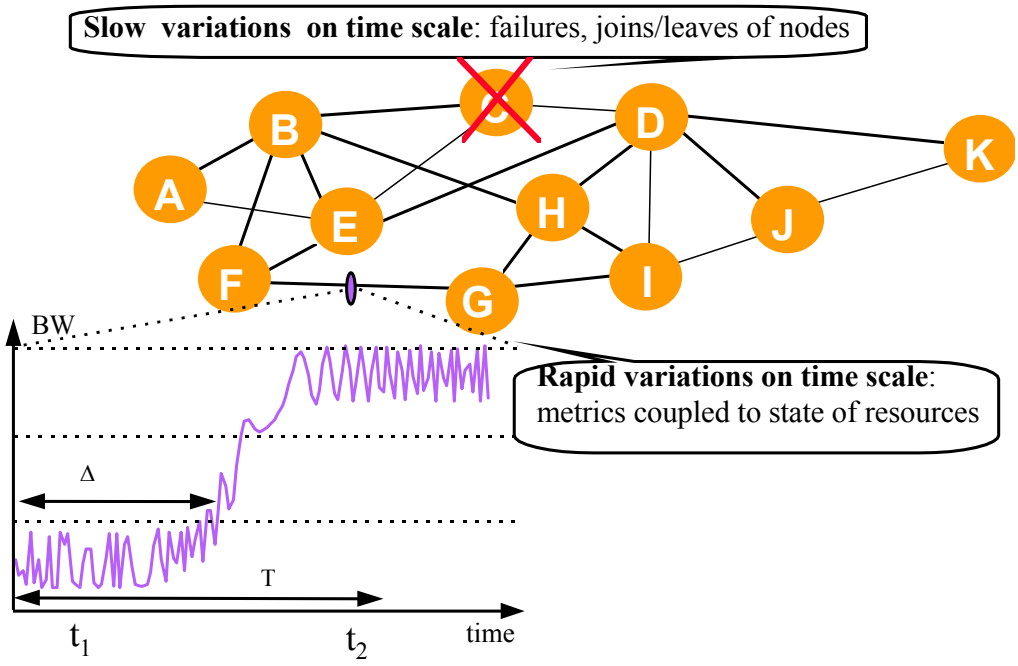


Figure 9.1: Network topology changes on different time scales. BW stands for bandwidth.

Figure 9.1 illustrates how the bandwidth BW on a link may change as a function of time. In contrast to the first kind changes where $T << \Delta$, in the second kind

changes, T can be of the same order as Δ. Apart from this, the second type changes necessitate the definition of a significant change that will trigger the process of flooding. In the first kind, every change was significant enough to start the flooding. The second kind significant change may be influenced by the flooding convergence time T and is, generally, strongly related to the traffic load in (a part of) the network. An optimal update strategy for the second type changes is highly desirable in future multi-media networks that are characterized by the broad variability in traffic profiles and QoS requirements.

Another fundamental issue is the extent of the largest subnet of the global network that is still important for the routing computation. In fact, the scope of the network consists of building a hierarchical structure (similar to PNNI) that includes the relevant region (subnet) in detail and makes abstraction (by information condensation) of the rest of the network. In this respect, the properties of a network topology (class of graphs) are very important. The Internet is shown to possess a power-law like degree distribution [48], while Ad-Hoc networks may vary from lattice structures to random graphs [76]. Since paths strongly depend on both link weight structure and graph properties, the network dynamics will dependent on these factors, even to the extent that some control strategies successful in a certain class of graphs may not work properly in other graphs.

9.2 Related work

We have made a distinction between (1) network changes that occur infrequently (topological changes) and (2) network changes that occur frequently (changes in the QoS link weights/resources). There is a wide variety of literature available that covers either the first or second kind of network changes.

The current Internet only considers the first type of infrequent topology changes and consequently the study of these changes dates back to the early days of the ARPANET. One of the topics studied is that of end-to-end Internet path stability. Paxson [130] provided us with one of the key papers in this area. He defined two types of stability, namely "prevalence," meaning the likelihood that a particular route is encountered and "persistence," the likelihood that a route remains unchanged over a long period of time. Based on measurements he found that *Internet paths are heavily dominated by a single prevalent route, but the time over which routes persist show wide variation, ranging from seconds to days.*

In the context of QoS routing, the second type of frequent changes in the network resources become a decisive factor and many problems emerge: (a) how to predict the traffic load, (b) when to update the network with new information, (c) how to update the network and (d) how to cope with inaccurate network state information. Below we briefly review the literature related to these problems.

9.2.1 Traffic prediction

Anjali *et al.* [5] proposed an algorithm to estimate the available bandwidth of a link in MPLS networks. They used a linear prediction model that is solved through Wiener-Hopf equations. Sang and Li [144] assessed the predictability of traffic by considering how far into the future a traffic rate process can be predicted with bounded error and what the minimum prediction error is over a specified prediction time interval. They used two models, namely the auto-regressive moving average and the Markov-modulated Poisson process and concluded that the applicability of traffic prediction is limited by the deteriorating prediction accuracy with increasing prediction interval. Jain and Dovrolis [86] targeted the end-to-end available bandwidth and stated that *the variability of the available bandwidth increases significantly as the utilization of the "low capacity" link increases, which makes a lightly loaded network have a more predictable and smooth throughput.* You and Chandra [178] and Basu *et al.* [14] analyzed Internet data measured at a campus and modeled this data using auto-regressive processes. Papagiannaki *et al.* [129] studied the evolution of IP backbone traffic at the larger time scale of hours and introduced a methodology to predict when and where link additions/upgrades have to take place in an IP backbone. They used mathematical tools to process historical information and extracted trends in traffic evolution at different time scales.

9.2.2 Network update triggering

QoS routing requires frequent distribution of link state information to provide routing tables with an as accurate view of the network as possible. However, frequently updating the network, through the dissemination of link state advertisements (LSA), can cause a significant overhead. Different link state update policies have therefore been proposed, which are reviewed in [105], [147], [6]. The link state update policies can be classified as either periodic based (LSA at fixed intervals) or trigger-based (LSA at a certain event) and may use either a hold-down timer or the moving-average principle [105] to reduce the number of LSA.

9.2.3 Network update distribution

The current Internet disseminates its network state through the entire network by using broadcast (flooding). In broadcast every router duplicates the network state information onto all of its outgoing links. This method may be too costly for QoS routing, where the frequency of updates is expected to be high. To reduce the overhead in broadcasting, Garcia and Spohn [55] proposed the adaptive link state protocol (ALP). A router in ALP disseminates link state updates incrementally to its neighbors for only those links along paths (trees) used to reach destinations. Huang and McKinley [79] also proposed a

tree-based protocol (T-LSR) that only constructs a single tree, shared by every router, for the dissemination of LSA and combined it with broadcast to make the protocol robust against node/link failures.

9.2.4 Inaccurate network state

The dynamics of QoS link weights, prohibits us to always obtain an accurate and up-to-date view of the network resources. The level of accuracy in state information is dependent on the choice of update strategy and can seriously impact the effectiveness (in terms of blocking) of path selection algorithms. A discussion of routing under inaccurate state information can be found in [61].

Another way of routing that avoids inaccurate state information is routing with only local information (that is up-to-date). Nelakuditi *et al.* [121] have done much work in this area. Unfortunately, the absence of global information may lead to non-optimal routing.

9.3 Stability of a path

In this section, the stability of a path in a dynamic environment is examined in a mathematical and simulative way. The simulations consisted of generating 10^4 graphs (from a particular class of graphs) with links weights according to a specific distribution. We have only assigned one weight w_e per link $e \in E$. This graph is considered to be a snapshot in time of our dynamically changing network. In this graph, first the shortest path P between a source s and a destination t was computed. Next, all the link weights[1] in the graph were perturbed by adding "link weight noise" $\widetilde{w}_{e;\alpha}$ with strength α, such that the resulting link weights equal $w_e + \widetilde{w}_{e;\alpha}$, $\forall e \in E$. This new graph represents a snapshot of the network at a later point in time and the noise therefore represents the impact of the arrival/departure of flows over time on the resources. The level of noise is related to the period of time and the size and arrival rate of the flows. We recomputed the shortest path P' between s and t in the perturbed graph and compared this path with the previously retrieved path P. The two most relevant parameters that were stored are the difference in path structure, i.e. how many links are different Δl, and the difference in path length Δw. Three classes of graphs were used, namely the class of random graphs $G_p(N)$ [21], with link density p independent of N, the class of square two-dimensional lattices and the class of Internet-like power-law graphs [48], with power $\tau = 2.4$ in the nodal degree distribution $\Pr[d = k] \sim k^{-\tau}$. However, only the results for the class of random graphs have been plotted. The results for the lattices

[1] By altering only the link weights of the links on the shortest path, we would have evaluated the influence of using a single path in the network. This study is not presented here, but is also a topic of research.

and power-law graphs can be found in [9]. The source s and destination node t were chosen randomly, except for the lattices, where they were positioned in opposite corners to achieve the largest minimum hop count. We have considered four ways of assigning the link weights:

1. $w_e = 1$ and $\widetilde{w}_{e;\alpha} = N(0, \alpha)$, $\forall e \in E$: All link weights are initially set to 1 and hence P is shortest in hop count. In the second scenario we added Gaussian noise, with mean 0 and variance α.

2. $w_e = U(0, 1)$ and $\widetilde{w}_{e;\alpha} = N(0, \alpha)$, $\forall e \in E$: Initially all link weights were assigned a value chosen from a uniform distribution in the range $(0, 1]$. In the second scenario we added Gaussian noise, with mean 0 and variance α.

3. $w_e = 1$ and $\widetilde{w}_{e;\alpha} = \alpha U(-0.5, 0.5)$, $\forall e \in E$: All link weights were initially set to 1. In the second scenario we added uniformly distributed noise in the range $\alpha[-0.5, 0.5]$.

4. $w_e = U(0, 1)$ and $\widetilde{w}_{e;\alpha} = \alpha U(-0.5, 0.5)$, $\forall e \in E$: Initially all link weights were assigned a value chosen from a uniform distribution in the range $(0, 1]$. In the second scenario we added uniformly distributed noise in the range $\alpha[-0.5, 0.5]$.

By varying α ($\alpha > 0$, corresponding to the level of noise/perturbation), we are able to evaluate the perturbation threshold that causes P and P' to differ. This threshold gives an indication of the size of the link state update thresholds that should be used. In all scenarios, especially when α gets large, it may happen that a link weight becomes negative. Negative or zero link weights are not considered realistic link weights and therefore we assume all link weights to be positive. Negative or zero link weights have been truncated at a very small value ($\varepsilon = 10^{-5}$) near zero to assure positive link weights.

The results for the four link weight scenarios were similar in behavior. This chapter only presents the results for $w_e = U(0, 1)$ and $\widetilde{w}_{e;\alpha} = \alpha U(-0.5, 0.5)$. We will first present a mathematical analysis of the path length for $w_e = U(0, 1)$ and $\widetilde{w}_{e;\alpha} = \alpha U(-0.5, 0.5)$, $\forall e \in E$, after which the simulation results follow.

9.3.1 Mathematical analysis

In this section we provide some upper bounds on the difference in path weights $\Delta w = w(P') - w(P)$, between the perturbed path P' and the unperturbed path P in any class of graphs. For an approximate calculus of the shortest path weight in the perturbed graphs, consult Appendix A.2.

By construction,

$$\begin{aligned} w(P) &= \min_{P\subset\{P_{st}\}} \left[\sum_{e\in P} w_e\right] \\ w(P';\alpha) &= \min_{P'\subset\{P_{st}\}} \left[\sum_{e'\in P'} (w_{e'} + \widetilde{w}_{e';\alpha})\right] \end{aligned}$$

where w_e are the unperturbed link weights and where $\widetilde{w}_{e;\alpha}$ is the perturbation with strength $\alpha \geq 0$. Clearly, $w(P';0) = w(P)$ and the maximum possible perturbed weight is bounded by

$$w(P';\alpha) \leq w(P) + \frac{\alpha}{2} h_P$$

where h_P is the hop count of the shortest non-perturbed path. The other extreme, in case of truncation, is $w(P';\alpha) = 0$ which occurs if there is a path from s to t with all link weights zero. Hence, denoting $\Delta w = w(P';\alpha) - w(P)$,

$$-w(P) \leq \Delta w \leq \frac{\alpha}{2} h_P$$

The relevant range for α is limited to $w_{e'} + \widetilde{w}_{e';\alpha} \geq 0$. The probability that a perturbed link weight is smaller than zero is found from (A.17), with $l = 1$ and $z = 0$ as

$$\Pr\left[w_{e'} + \widetilde{w}_{e';\alpha} \leq 0\right] = \begin{cases} \frac{\alpha}{8}, & \alpha \leq 2 \\ \frac{1}{2} - \frac{1}{2\alpha}, & \alpha > 2 \end{cases} \tag{9.1}$$

If we consider the binomial distribution with $\Pr[X = 1] = r = \Pr[w_{e'} + \widetilde{w}_{e';\alpha} \leq 0]$ and $\Pr[X = 0] = 1 - r$, then the probability that k out of the total M links are truncated equals

$$\Pr[X = k] = \binom{M}{k} r^k (1-r)^{M-k} \tag{9.2}$$

and $\mathbb{E}[X] = Mr$, $var[X] = Mr(1-r)$.

Roughly, the probability to have a zero weight path is bounded from below by,

$$\begin{aligned} \Pr\left[w(P';\alpha) = 0\right] &\geq \prod_{e\in P'} \Pr\left[w_{e'} + \widetilde{w}_{e';\alpha} \leq 0\right] \\ &\approx \begin{cases} \left(\frac{\alpha}{8}\right)^{\mathbb{E}[h]}, & \alpha \leq 2 \\ \left(\frac{1}{2} - \frac{1}{2\alpha}\right)^{\mathbb{E}[h]}, & \alpha > 2 \end{cases} \end{aligned}$$

which is only significant for large α and a small expected hop count $\mathbb{E}[h]$.

9.3.2 Simulations for Δw

We will first present our results for the difference in path weights $\Delta w = w(P';\alpha) - w(P)$ for the class of random graphs $G_p(N)$ with $N = 1000$ and link density $p = 0.2$ and $p = 0.01$. For each simulation 10^4 connected graphs were created. The link weights were assigned according to $w_e = U(0,1)$ and $\widetilde{w}_{e;\alpha} = \alpha U(-0.5, 0.5)$.

The random graph

Figure 9.2 presents $\mathbb{E}[\Delta w]$ as a function of the perturbation strength α.

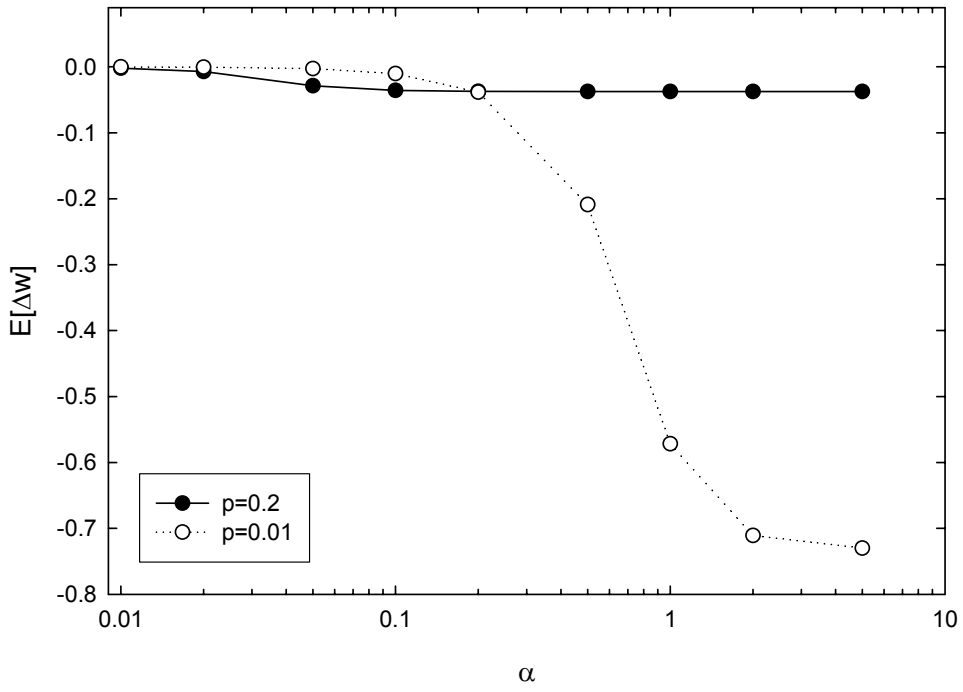


Figure 9.2: The expected difference in length ($\mathbb{E}[\Delta w] = \mathbb{E}[w(P') - w(P)]$) between P and P' for the class of random graphs ($N = 1000$), as a function of the perturbation strength α.

For the class of random graphs with $p = 0.2$, $\mathbb{E}[\Delta w]$ decreases already for very small values of α, although this decrease is only small. For large α, $\mathbb{E}[\Delta w]$ saturates at $\mathbb{E}[\Delta w] \approx -\mathbb{E}[w(P)]$. For the class of random graphs with $p = 0.01$, $\mathbb{E}[\Delta w]$ starts decreasing at larger α than with $p = 0.2$ and the decrease is steeper. However, $\mathbb{E}[\Delta w]$

again saturates at $\mathbb{E}[\Delta w] \approx -\mathbb{E}[w(P)]$. This implies that $\mathbb{E}[w(P';\alpha)] \approx 0$ for large α, irrespective of p, as was expected from our mathematical analysis.

In $G_p(N)$, a typical length is $\mathbb{E}[w(P)] \sim \frac{\ln N}{Np}$. The ratio of the two link densities considered here ($p = 0.2$ and $p = 0.01$), $\frac{0.2}{0.01} = 20$, almost precisely equals the ratio in $\mathbb{E}[\Delta w]$ at saturation, which equals $\frac{\mathbb{E}[\Delta w, p=0.01]}{\mathbb{E}[\Delta w, p=0.2]} = \frac{-0.730}{-0.037} = 19.5$.

The expected hop count of the shortest path P in $G_p(N)$, with p fixed and uniformly (or exponentially) distributed link weights, scales as $O(\log N)$ and the number of paths between source and destination is expected to be large, for N large. According to (9.2), if α is high, there is a high probability that several links have weights truncated at 0. Especially for $G_p(N)$, when p is fixed, this results in a high probability that the shortest path only consists of such zero-weight links. This behavior is verified in Figure 9.2. If the link density p is small, then the number of (truncated) links is smaller and the expected hop count larger. Hence a stronger perturbation is required before the saturation state $\mathbb{E}[\Delta w] \approx -\mathbb{E}[w(P)] \approx -\frac{\ln N}{NP}$ is reached.

The more interesting and realistic case is for α small. Our analysis indicates that if α is small, the probability that a link is truncated ($\Pr[w_{e'} + \widetilde{w}_{e';\alpha} \leq 0] = \frac{\alpha}{8},\ \alpha \leq 2$) is also small. In this case only a dense graph is likely to have a zero-weight path between source and destination. When examining Figure 9.2, we indeed see a decrease for small α in graphs with a high link density p, and zero difference for small p.

The square two-dimensional lattice

For the class of two-dimensional lattices, the minimum hop count equals $h_{\min} = 2\sqrt{N} - 2$. Due to the relatively high expected hop count not all high link weights may be circumvented, which may result in a linear increase with α (i.e., no saturation).

The power-law graph

For the class of power-law graphs, the hop count of the shortest path is expected to scale as $O(\log N)$, but most likely it is larger than the expected hop count in $G_p(N)$, which also scales as $O(\log N)$. Contrary to the class of random graphs, the number of paths in the power-law graphs between a source and destination node is believed to be relatively small. In this case, there is less choice to circumvent the links with high weights, which is manifested for large α where $\mathbb{E}[\Delta w]$ increases linearly with α.

The linear increase will be smaller for $\tau = 2.0$ than for $\tau = 2.4$, because a smaller τ leads to denser graphs and hence more possibilities to circumvent high link weights.

For the class of power-law graphs we have also simulated with link weights with a fixed granularity. We chose a granularity of 10, meaning that the link weights could only take one out of 10 values uniformly distributed in the range [0,1]. For these simulations, the paths P and P' remained the same ($\mathbb{E}[\Delta w] = 0$ and $var[\Delta w] = 0$) up to higher values of α, which confirms that choosing a larger granularity improves stability.

9.3.3 Simulations for Δl

To better evaluate the difference between P and P', we have also stored the number of different links Δl. More formally, Δl is the sum of the non-overlapping edges of P and P', and therefore (between the same source and destination nodes) Δl cannot be 1 or 2. Figures 9.3 and 9.4 display the results for Δl in the class of random graphs. The results for the lattices and power-law graphs can be found in [9]. The simulations were done with $\alpha = 0.01, 0.02, 0.05, 0.1, 0.2, 0.5, 1.0, 2.0, 5.0$, but to avoid an overload of curves, only four values of α have been plotted.

The random graph

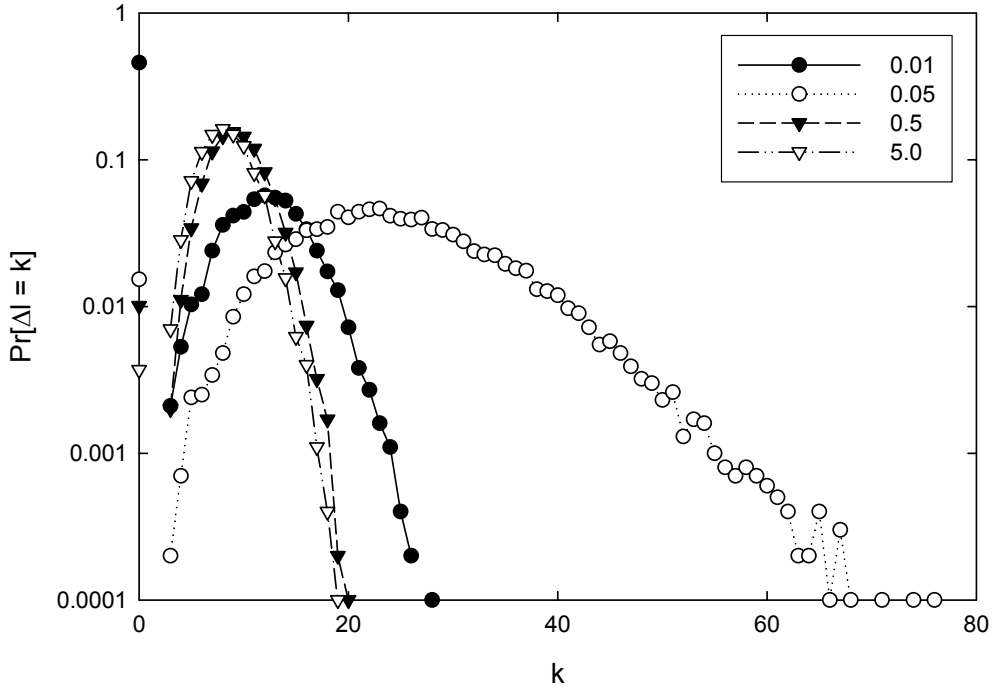


Figure 9.3: Probability that the number of different links Δl equals k in the class of random graphs $G_{0.2}(1000)$ for different values of α.

In Figures 9.3 and 9.4, for $G_{0.2}(1000)$, a small perturbation α can already trigger a large Δl. An interesting observation is that $\mathbb{E}[\Delta l]$ and $var[\Delta l]$ increase up to $\alpha = 0.05$, after which they decrease and stabilize for $\alpha > 0.1$. Note that $\alpha = 0.1$ is precisely the value at which $\mathbb{E}[\Delta w]$, for $p = 0.2$ in Figure 9.2, stabilized. Figure 9.4 also plots the

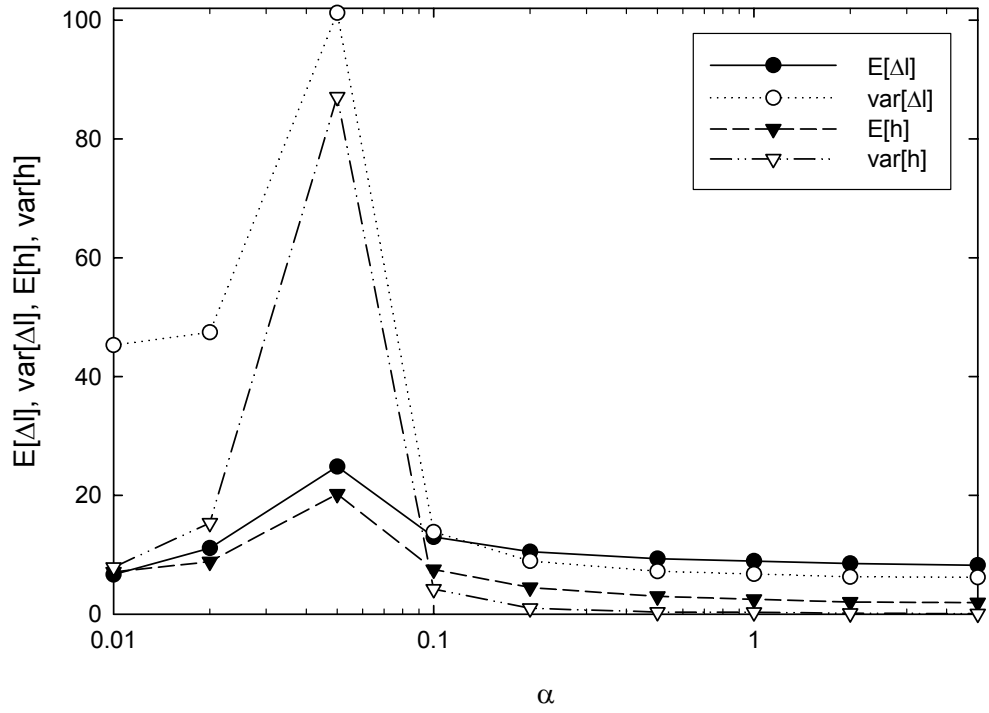


Figure 9.4: The expected difference in links $\mathbb{E}[\Delta l]$ as a function of α in the class of random graphs $G_{0.2}(1000)$.

expected hop count $\mathbb{E}[h]$ of the shortest path in the perturbed graph and its variance $var[h]$. $\mathbb{E}[h]$ and $var[h]$ are similar in shape to $\mathbb{E}[\Delta l]$ and $var[\Delta l]$. This is expected since the shortest path in the unperturbed graph is independent of α. In Figure 9.4, $\mathbb{E}[h]$ decreases after $\alpha = 0.05$ and stabilizes for $\alpha > 0.1$. The expected hop count for $\alpha = 5$ equals 1.8, which corresponds to the expected hop count of the shortest path in the random graph $G_p(N)$ with constant link weights ($\mathbb{E}[h_{\min}] \simeq 2 - p$ and $var[h_{\min}] \simeq p(1-p)$, [164]). If the number of truncated links in the perturbed random graph $G_p(N)$ with a constant (i.e., truncated) link weight ε scales as $O(N^b)$, $b \geq 1$, then the shortest path in that graph behaves as if all links were constant. In that case, the average number of truncated links per node > 1. We can consider such a graph to be "superconducting." According to (9.2), the number of truncated links equals $M\left(\frac{\alpha}{8}\right)$, where $M = \frac{pN(N-1)}{2}$ for $G_p(N)$ and which scales as $O(\alpha N^2)$ for α large enough. We therefore expect that $\mathbb{E}[h_{\min}] \simeq 2 - p$ and $var[h_{\min}] \simeq p(1-p)$, which is verified in the simulations. Moreover, if we sum the expected hop counts of the

unperturbed path P ($\mathbb{E}[h_{\min}] \simeq \ln N$) and the perturbed path P' ($\mathbb{E}[h_{\min}] \simeq 2 - p$), we find $\ln(1000) + 2 - 0.2 = 8.7$, which coincides with the peaks at saturation (for $\alpha > 0.1$) in Figure 9.3. Only if $\alpha \lesssim \frac{16}{pN}$ our scaling rule does not apply and we may consider the graph to be in "normal" regime. The phase transition from normal to superconducting is observed in Figure 9.4.

The square two-dimensional lattice

Due to the regular structure of the two-dimensional lattices, Δl can never be odd. The two-dimensional lattices are more stable than the random graphs. There is a higher probability that the paths P and P' are identical. Compared to the random graphs, the tail of Δl in the two-dimensional lattices is longer, due to the higher expected hop count.

The power-law graph

For the realistic regime of small values of α, the paths in the power-law graphs can be considered stable, but for high α there is a long tail and an irregular "distribution" function. These irregularities in the distribution suggest regularity or combinatorial confinement (in part of) the topology. The long tail suggests that sometimes the hop count of the shortest path in a power-law graph can be very large.

9.4 Conclusions on QoS stability

Network dynamics can be categorized as either slowly or frequently changing. The slow changes relate to changes in nodes or links and have been studied in the past. The frequent changes relate to changes in the link weights. Sections 9.1-9.3 have provided a preliminary evaluation of the stability of paths in a dynamically changing network. To our knowledge, this approach has never been considered in the literature and it may provide new insight into the dimensioning/tuning of future QoS architectures.

We have provided a mathematical analysis of the difference in length between the shortest path in an unperturbed graph and the shortest path in a perturbed graph. Subsequently, we have simulated the difference between these two paths. Those simulations revealed that up to a perturbation of roughly 10%, the weights of the two paths remained close to each other, while the number of different links could vary substantially, depending on the class of graphs. In practice, the link weights will have a finite granularity instead of the real values used in this section. A larger granularity will improve the stability of paths and consequently the predictability of network state. The results for the difference in links Δl displayed more volatility, as a function of the perturbation strength α, than the difference in weights Δw. We believe that the latter parameter is a more important measure to set a threshold for updating the network

state. However, the work presented here is still ongoing and therefore more simulations have to be run to substantiate our results. Future work will also focus on better understanding the influence of link weight granularity on path stability, on exploring the "slack" in a network and on multi-constrained routing in a dynamic environment. This study, once fully understood, will provide a foundation upon which efficient and accurate link state update protocols can be build. In the remainder of this chapter we assume that we have such an up-to-date knowledge of the state of the QoS weights. Given this knowledge, we will evaluate how QoS algorithms perform in a dynamically changing network.

9.5 Introduction to dynamic QoS algorithms

The goal of the remaining sections in this chapter is to evaluate the performance of SAMCRA and other algorithms in a dynamic environment. Several algorithms [96, 109, 179, 167, 38] have been proposed to find the shortest path subject to multiple constraints. The performance of these QoS routing algorithms has not yet been evaluated in a dynamic scenario. Since the paths chosen for past requests may have an influence on the admittance of future requests, the choices made by the routing algorithm have an impact on the network throughput and the number of admitted calls. Maximizing throughput and the number of admitted calls and optimizing network resource usage are goals of traffic engineering algorithms. Although proposed to work in a dynamic network, most of them disregard QoS constraints.

Ideally, one should have a routing algorithm that aims at maximizing throughput (or minimizing blocking), while satisfying the users' QoS constraints. To account for these objectives, SAMCRA is equipped with a special path length definition that guarantees the QoS constraints and accounts for the traffic engineering objectives. For clarity, this variant SAMCRA-B is named.

We have conducted many simulations to evaluate the performance of several algorithms. The performance measures are the call blocking rate, the bandwidth blocking rate, the throughput, and the time they required to compute a path.

9.6 Problem statement

First the problem under consideration is formally defined. The available bandwidth on each link $e \in E$ is denoted by $R(e)$. In addition to bandwidth, each link is characterized by a vector of m additive QoS link weights $(w_i(e), i = 1, ..., m, e \in E)$.

A flow request is characterized by a quadruple $(s, t, B, \vec{L})$, where s is the source node, t is the destination node, B is the requested bandwidth and $\vec{L}$ is a vector representing the m other additive QoS constraints. When a flow request arrives, the routing algorithm

computes a path for the flow. If enough resources are available on this path, the flow can be admitted. A path P is said to be feasible, i.e. a flow can be admitted, if:

$$\begin{cases} \min_{e \in P} R(e) \geq B \\ \sum_{e \in P} w_i(e) \leq L_i \quad i = 1, ..., m \end{cases}$$

The problem under consideration is to find the most efficient way to allocate flows, with the goal to optimize network utilization.

9.7 Traffic engineering algorithms

9.7.1 Overview

Most recently proposed traffic engineering algorithms are inspired by the work of Kar, Kodialam and Lakshman [91]. They presented a routing algorithm MIRA, based on the concept of "minimum interference." The amount of interference on a particular source-destination pair (s, t), caused by allocating a flow between some other source-destination pair, is defined as the decrease in "maxflow" [2] between s and t. Maxflow is an upper bound on the total amount of bandwidth that can be routed between a source-destination pair. The minimum interference path between a particular source-destination pair is the path that maximizes the minimum maxflow between all other source-destination pairs. The problem of finding this path is NP-hard. Therefore, Kar *et al.* proposed to determine appropriate link weights, prune links with insufficient available bandwidth and compute the shortest path in the pruned topology. MIRA considers only the QoS measure bandwidth. Wang *et al.* [169] proposed a different definition for the link weights in MIRA. We denote this variant of MIRA as "New MIRA."

Banerjee and Sidhu [11] proposed two algorithms: one (TE-B) takes into account only bandwidth, while the other (TE-DB) considers also delay. The authors introduced three objectives for traffic engineering: (1) reducing the blocking of requests, (2) minimizing network cost, and (3) distributing network load. This formulation has three objective functions (plus the delay constraint in the case of TE-DB) and is proven to be NP-complete [11]. Banerjee and Sidhu presented another formulation in which the first two objective functions are transformed into constraints. Both TE-B and TE-DB make use of TAMCRA (see Chapter 5), to find a candidate set of k paths satisfying the set of constraints. TE-B and TE-DB are computationally more expensive than MIRA, since they require a couple of shortest path computations and a TAMCRA execution, in addition to the maxflow computations.

Iliadis and Bauer [81] introduced a new class of minimum-interference routing algorithms, called SMIRA (Simple Minimum-Interference Routing Algorithms). These algorithms evaluate the interference on a source-destination pair by means of a k-shortest-path-like computation, instead of a maxflow computation. The set of k paths between a

source-destination pair (s, t) is determined by first computing the widest-shortest path [64] between s and t using static link costs. Then, the bottleneck bandwidth of this path is determined and all the links along the path that have a residual capacity equal to the bottleneck bandwidth are pruned. The second path is found by computing the widest-shortest path in the pruned topology. This procedure is repeated until either k paths are found or no more paths are available. Iliadis and Bauer [81] proposed two SMIRA algorithms, MI-BLA (Minimum-Interference Bottleneck-Link-Avoidance) and MI-PA (Minimum-Interference Path Avoidance). In the simulations in [81], MI-PA achieves a better performance than MI-BLA.

9.7.2 Limitations

The algorithms described above all have some drawbacks. The first drawback is their complexity. A maxflow computation requires a time-complexity of $O(N^2\sqrt{M})$ and it must be executed for each source-destination pair. Consequently, the time complexity increases prohibitively as the network grows.

Another important disadvantage is that the algorithms, with the exception of TE-DB, do not explicitly take into account QoS constraints. Some of the authors claim that such constraints can be converted into an effective bandwidth constraint. However, the theory of effective bandwidth is complex and not generally applicable. Therefore, in order to actually satisfy the additive QoS constraints, it is necessary to explicitly take them into account. This is partly by TE-DB, and fully by SAMCRA and its new variant SAMCRA-B, described in the following section.

9.8 SAMCRA-B

For each link, the QoS link weights are constant upper bounds, such that the QoS values perceived by packets crossing the link do not exceed these QoS weights. This assures that the additive QoS constraints of admitted flows remain satisfied even after new flows join. Only bandwidth is considered to be an adaptive measure that changes with the joining and leaving of nodes. In order to obtain an algorithm capable of guaranteeing QoS constraints and optimizing network throughput, SAMCRA (see Section 4.6) has been equipped with a new path length function to improve its behavior in a dynamic network. As discussed in Section 4.1, SAMCRA can be used with any path length function. Length function (4.3) is a function of the (constant) QoS link weights $(w_i(e), i = 1, ..., m)$ and of the QoS constraints $(L_i, i = 1, ..., m)$.

Instead of (4.3), we propose a different function for the length of a path P in

SAMCRA:

$$\begin{aligned} l(P) &= \sum_{e \in P} w_0(e), \text{ if } \vec{w}(P) \leq \vec{L} \\ &= \infty, \text{ else} \end{aligned} \tag{9.3}$$

where $w_0(e)$ is a link weight that depends on dynamic information. We have assigned $w_0(e) = \frac{1}{R(e)}$, where $R(e)$ represents the available bandwidth on link e. We are going to use SAMCRA with the two different length functions (4.3) and (9.3). The latter variant is named SAMCRA-B. SAMCRA-B selects the shortest path according to (9.3) among those satisfying the QoS constraints. The constraints make length (9.3) non-linear. Note that SAMCRA-B has an additional measure w_0 as compared to SAMCRA.

9.9 Performance evaluation

In this section, the performance of several algorithms is evaluated. The algorithms under consideration are: Dijkstra with weights $\frac{1}{R(e)}$ (labeled as "SP-Inv"), widest-shortest path [64] (labeled as "WidShort"), New MIRA, MI-PA, TE-DB, SAMCRA and SAMCRA-B. The objective was to compare the traffic engineering algorithms (New MIRA, TE-DB, MI-PA) to SAMCRA and to evaluate the gain that is possibly achieved by using the new path length function of SAMCRA-B.

The experiments were carried out using various topologies varying from 18 nodes (see Figure 9.5) to 100 nodes. All simulations on the different types of graphs displayed similar results (i.e., ranking among the algorithms) as for the topology in Figure 9.5. Therefore only the results for this topology are presented. In Figure 9.5, the dark shaded nodes represent the edge routers, the entry and exit points for the network traffic, while the other nodes represent core routers, which carry transit traffic only. The topology consists of 18 nodes (7 edge nodes) and 31 links. The capacity of the light links is 1550 units and that of the dark links is 6220 units. All links are symmetric, with respect to both capacity and QoS link weights. In our simulations, we considered two additive QoS constraints ($m = 2$) and refer to them as delay and packet loss. Figure 9.5 shows for each link a two-component vector of QoS link weights $(w_i(e), i = 1, 2)$.

The flow-level simulator NetSim++ [105] (developed at TU Delft) has been used to analyze and compare the performance of the different routing algorithms in a dynamic scenario. NetSim++ makes use of several random variables, which specify the characteristics of the flows that load the network. Throughout all simulations, the source and destination nodes were chosen uniformly among the set of edge nodes.

Several scenarios have been studied. Each scenario consisted of some tests and each test was iterated 20 times. For each of these 20 iterations, the algorithms under evaluation faced the same scenario and the same set of flow requests. Each iteration

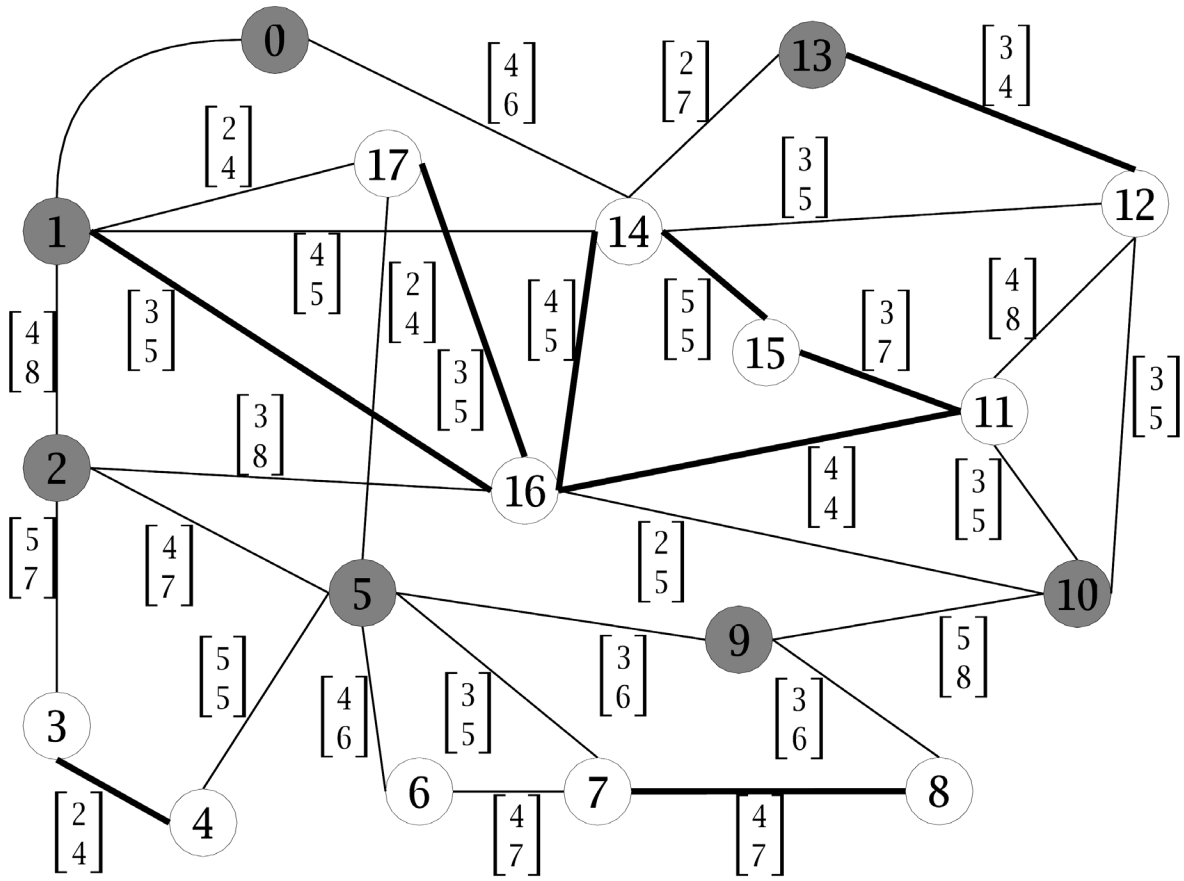


Figure 9.5: Simulation network topology.

involved the generation of 120000 flows. The first 20000 were not considered, as they represent a warm-up period needed to load the network.

Each iteration stored the call blocking rate (CBR) and the bandwidth blocking rate (BBR) of each algorithm. These performance measures are defined as follows:

$$CBR = \frac{\text{number of rejected flows}}{\text{total number of flows}}, BBR = \frac{\sum_{\text{rejected flows}} \text{requested BW}}{\sum_{\text{all the flows}} \text{requested BW}}$$

After the processing of each new flow request, the throughput was computed as the sum of the bandwidth requested by the flows crossing the network at that moment. Finally, for each test we measured the average processor time used by each algorithm to select a path. The simulations were run under Linux on a Pentium III 866MHz processor.

In the following subsections the three investigated scenarios and their results are discussed. The first is intended to compare all the implemented algorithms, while

the other two only compare the algorithms which explicitly take into account QoS constraints. For the first scenario, the additive QoS constraints are chosen very large, such that no algorithms will reject any request due to a violation of the delay or packet loss constraint.

9.9.1 Scenario 1: influence of bandwidth constraint

The purpose of this set of simulations is to study the behavior of the implemented algorithms under different bandwidth constraints, while keeping the load offered to the network constant. The product of mean number of flow arrivals per time unit, mean requested bandwidth and mean flow duration equals the average load per time unit.

Scenario 1 comprises two tests, which differ in the distribution of the bandwidth requirement. The bandwidth requirement is uniformly distributed between 1 and 10 units for test 1a, which is very small compared to the link capacities (1550 or 6220 units). The bandwidth requirement for test 1b is taken from a uniform distribution U(1,10) with probability 0.7 and from a uniform distribution U(80,100) with probability 0.3.

The results for tests 1a and 1b are summarized in Table 9.1. Table 9.1 shows the proportional increase between the minimum mean CBR (BBR) μ_{min} and the mean CBR (BBR) μ of each of the other algorithms:

$$\Delta(test) = \frac{\mu - \mu_{min}}{\mu_{min}} \times 100 \tag{9.4}$$

In the case of test 1a, the minimum call blocking rate was achieved by SAMCRA-B and SP-Inv. Since the additive QoS constraints are chosen sufficiently large, SAMCRA-B reduces to SP-Inv. The small difference in behavior can be explained by considering that the two algorithms may choose a different path when multiple equal-length paths are present. The call blocking rate achieved by New MIRA was higher (about 14%) than the minimum, while the CBR of WidShort, TE-DB and SAMCRA was far greater (above 45%). The highest call blocking rate was achieved by MI-PA. The same conclusions can be drawn, when analyzing the performance in terms of bandwidth blocking rate. However, the BBR of New MIRA is closer to the minimum achieved by SAMCRA-B and SP-Inv. This suggests that New MIRA accepts less flows than SAMCRA-B and SP-Inv, but accepts those with greater bandwidth requirements. Also the BBR of TE-DB, which makes use of the maxflow concept to define link weights, is now similar to that of WidShort. The maximum throughput was reached by SAMCRA-B and SP-Inv, which were closely followed by New MIRA. The time required by WidShort, SAMCRA and SP-Inv is of the same order of magnitude, while SAMCRA-B requires a time double to that of SAMCRA. The difference in path computation time between SAMCRA-B and SAMCRA is due to the additional QoS measure (bandwidth) in SAMCRA-B. Also the difference in length function is likely to be of influence. The time required by New MIRA, TE-DB and MI-PA is two orders of magnitude bigger than that of SAMCRA.

Table 9.1: Scenario 1: proportional increase

	$\Delta_{CBR}(1a)$	$\Delta_{BBR}(1a)$	$\Delta_{CBR}(1b)$	$\Delta_{BBR}(1b)$
WidShort	46.08	43.49	6.06	6.45
New MIRA	14.28	6.62	min	min
TE-DB	53.2	45.12	6.62	6.03
MI-PA	105.9	100.67	34.27	34.75
SAMCRA	66.2	62.36	10.92	11.26
SAMCRA-B	min	min	0.37	0.95
SP-Inv	0.07	0.13	0.06	0.75

MI-PA is slightly faster than New MIRA and TE-DB, as expected since it does not require maxflow computations.

Test 1b differs from test 1a in the characterization of the bandwidth requirement. Table 9.1 reveals that there is a very small difference between the call blocking rate of New MIRA (the minimum) and those of SAMCRA-B (0.37% greater) and SP-Inv (0.06% greater). Moreover, the CBR of TE-DB, WidShort and SAMCRA is now closer to the minimum CBR (less than 11% greater). MI-PA again achieved poor performance. The same was observed for the bandwidth blocking rate and throughput. We observed that the CBR of all the algorithms is greater with respect to test 1a (this is especially true for the algorithms achieving the minimum), even though the load offered to the network was the same. This means that the algorithms are able to accept more calls when the bandwidth requirements are small. This can be explained by considering that small requests fit the residual bandwidth of links better than large requests. The algorithms required almost the same time to compute a path as for test 1a.

Finally, from the results obtained in different topologies, we saw a similar classification among the algorithms, which seems independent of the topology size. However, more simulations are needed to substantiate this observation.

9.9.2 Scenario 2: influence of one QoS constraint

In the last two scenarios, we have only compared the algorithms that explicitly take into account QoS constraints. SAMCRA, SAMCRA-B and TE-DB fall into this category. In addition, we have included SP-Inv, which behaves like SAMCRA-B when the QoS constraints are sufficiently large. In scenario 2, four tests were simulated, which differed only in the delay constraint. The topology and the parameters of the other random variables did not change throughout these tests and were equal to those used for test 1a.

Table 9.2 shows the difference Γ (scaled by 100) between the mean CBR (BBR) of SAMCRA-B and that of each of the other algorithms. The delay constraint becomes more stringent going from test 2a to test 2d. Table 9.2 affirms that SAMCRA-B achieved

Table 9.2: Scenario 2: Gap between SAMCRA-B and the other algorithms

	SAMCRA		TE-DB		SP-Inv	
Test	Γ_{CBR}	Γ_{BBR}	Γ_{CBR}	Γ_{BBR}	Γ_{CBR}	Γ_{BBR}
2a	1.35	1.78	1.17	1.41	1.08	0.62
2b	2.24	2.69	2.29	2.64	8.42	7.81
2c	3.61	4.34	3.67	4.28	12.84	12.63
2d	3.46	4.19	3.55	4.12	14.48	14.48

the minimum CBR in all the tests. When the constraints become very strict, only a few (if any) feasible paths are available. In this case, all the algorithms are likely to make the same choice. Furthermore, the CBR of SAMCRA is initially greater than that of TE-DB, but eventually it becomes smaller. This suggests that SAMCRA, due to its exactness, is less sensitive to the tightening of QoS constraints than TE-DB. Finally, Table 9.2 shows that the performance of SP-Inv (in terms of both CBR and BBR) rapidly degrades as the delay constraint becomes more stringent. The path computation times were similar to those measured for test 1a.

9.9.3 Scenario 3: influence of both QoS constraints

In the last scenario we have decreased both constraints. The minimum CBR (BBR) was again achieved by SAMCRA-B. The gap between the average CBRs (BBRs) of SAMCRA-B and TE-DB and the gap between the average CBRs (BBRs) of SAMCRA-B and SAMCRA continuously increase with decreasing constraints. This means that SAMCRA-B is less sensitive to the tightening of QoS constraints. We came to the same conclusions from the results of scenario 2. Again, the performance of SP-Inv (in terms of both CBR and BBR) rapidly degraded as the QoS constraints become more stringent.

9.10 Conclusions on dynamic QoS algorithms

Dynamic QoS algorithms should aim to guarantee QoS constraints, while making efficient use of the network's resources. Moreover, these objectives should be met in a network with dynamically changing link weights. A new length function for SAMCRA was used to target these objectives and this variant was named SAMCRA-B. To analyze the extent to which several QoS and traffic engineering algorithms meet the objectives of an optimal dynamic algorithm, a simulative study has been conducted. The simulations were done on different networks, under the assumption that the bandwidth on a link could dynamically change and that the link weights referring to delay and packet loss remained constant. The performance indicators were: call blocking rate, bandwidth

blocking rate, throughput, and path computation time. Several scenarios have been analyzed, either under loose QoS constraints or by tightening the QoS constraints. For the first type of loosely-constrained scenarios SAMCRA-B and SP-Inv displayed the best performance. The algorithms based on the maxflow concept (New MIRA and TE-DB) performed better in terms of bandwidth blocking rate and throughput, than in terms of call blocking rate. Nonetheless, in every test the bandwidth blocking rate of SAMCRA-B was either the minimum or very close to the minimum. The simulations also revealed that the path length function of SAMCRA-B (based on the current bandwidth availability) allows a considerable advantage over SAMCRA with a static length function.

In the last type of tightly-constrained scenarios SAMCRA-B performed the best. Moreover, the gap between SAMCRA-B and the other algorithms (SAMCRA and TE-DB) increased as the QoS constraints became more stringent.

In all the tests, the average path computation time of SAMCRA and SAMCRA-B was two orders of magnitude smaller than New MIRA and TE-DB. From these scenarios, we can conclude that SAMCRA-B achieves the best performance at a low computational cost. However, all the simulations were based on a dozen topologies and therefore only indicate a potential for SAMCRA with a properly chosen path length function. Further simulations are needed to confirm our claim.

Chapter 10

Conclusions

The continuously demand for using multimedia applications over the Internet has spurred research on how to satisfy the quality of service (QoS) requirements of these applications, e.g., requirements regarding bandwidth, delay, jitter, packet loss, and reliability. One of the key issues in providing QoS guarantees is

how to determine paths that satisfy QoS constraints.

Solving this problem is known as QoS routing or constraint-based routing and is the main topic of this thesis.

Routing in general involves two entities, namely the routing protocol and the routing algorithm. The routing protocol manages the dynamics of the routing process: it captures the state of the network and its available network resources and distributes this information throughout the network. The routing algorithm uses this information to compute paths that optimize a criterion and/or obey constraints. In QoS routing, the goal is to find paths that obey multiple user-desired QoS constraints, also referred to as the multi-constrained path (MCP) problem. Paths that obey the constraints are called feasible paths. The multiple constraints make the MCP problem difficult. To facilitate exact QoS routing, we used four concepts: (1) a non-linear length function, (2) a k-shortest paths approach, (3) the concept of non-dominance and (4) the look-ahead concept. The non-linear length function is necessary, because

multiple constraints make the MCP problem non-linear.

If QoS routing is performed via a linear length function, then exactness cannot be guaranteed. Motivated by the constraints surface, we have proposed the non-linear length function $l(P) = \max_{i=1,\ldots,m}\left(\frac{w_i(P)}{L_i}\right)$. A problem of a non-linear length function is that subsections of shortest paths are not necessarily shortest themselves. In this case, examining only shortest sub-paths at a node may lead to an end-to-end path that is not the shortest one. Therefore,

multiple paths at a node may have to be examined.

This is accomplished by a k-shortest paths approach, where not only examine the shortest paths are examined, but also the second shortest up to the k-th shortest path. In the worst case, this k-shortest paths approach could evaluate all possible paths and therefore it requires efficient search-space-reducing techniques.

One technique to reduce the search space is the concept of non-dominance, which excludes paths P' for which a path P is already known that has weights $w_i(P) \leq w_i(P')$, for $i = 1, ..., m$.

Another concept to reduce the search space is that of look-ahead.

Look-ahead computes lower-bound vectors to assist in determining whether a path can become feasible or not. Based on the above-mentioned four concepts, we proposed the exact algorithm SAMCRA. Naturally, besides the four concepts, others may exist, e.g., a preprocessing of the graph that prunes links that cannot be on the shortest path, or a bi-directional search. Based on such new concepts SAMCRA could evolve over time and keep the leading position it occupies today. SAMCRA is not the only algorithm that has been proposed to solve the MCP problem, although it is one of the few exact algorithms.

Many researchers have investigated the constraint-based path selection problem and proposed various algorithms, mostly heuristics. This wealth of algorithms grew so large that they complicated a good perception of the possibilities. Moreover, based on very limited simulations, each claimed superiority over the others. To gain more insight in these algorithms, this thesis has provided a large-scale and fair performance evaluation of the lion's share of QoS algorithms. The simulations were conducted using particular distributions, different types of correlation and different classes of graphs. To obtain confident results, we used simulation runs that consist of 10^4 iterations. In general, the simulation results indicated that

better performance is obtained with SAMCRA-like algorithms that use a k-shortest path algorithm with a non-linear length function that eliminate paths via the non-dominance and look-ahead concepts.

for the problems considered (RSP, MCP, MCOP). The performance and complexity of these algorithms can easily be adjusted by controlling the value of k. When k is not restricted, the SAMCRA-like algorithms lead to exact solutions. In the class of Waxman or random graphs, with uniformly distributed link weights, simulation results

suggest that the execution times of such exact algorithms increase linearly with the number of nodes. Therefore,

> *in "easy" cases, the execution time of exact algorithms is comparable to those of heuristics, and the success rate of heuristics approaches that of exact algorithms.*

In contrast,

> *in "hard" cases, the success rate of heuristics drops to zero, while the execution time of exact algorithms grows exponentially.*

This was observed for the class of lattices with negatively correlated link weights. In our simulations, the polynomial-time ϵ-approximation schemes displayed an extensive execution time and therefore seem unsuitable for practical purposes. More research is necessary to indicate whether these algorithms might be a good alternative for exact algorithms in large and hard topologies.

QoS routing does not only find a feasible path between a source and a destination, it also relates to efficiency and security. We have proposed an algorithm to find an efficient multicast graph in a network that obeys a set of QoS-constraints and have shown that

> *a multicast tree may not always guarantee the requested QoS-constraints, not even if multiple unicast QoS sessions will.*

This property increases the complexity of constrained multicast routing, since we have to maintain a set of paths/trees and we need to check if no min/max constraints are violated (mere topology filtering may be insufficient). In multicast QoS routing, a trade-off has to be made between efficient use of resources and QoS, which resulted in the proposed algorithm MAMCRA. MAMCRA computes the set S of shortest paths from source s to all other nodes, and then reduces this set to an efficient set of multicast routes, without compromising the requested level of QoS. Simulations with MAMCRA indicate that it often returns trees or sub-graphs that closely resemble a tree. It was shown that

> *it is desirable to always construct (to strive to construct) a multicast tree*

either by fine-tuning MAMCRA or by renegotiating the constraints.

For secure and reliable QoS routing, a single feasible path between source and destination may be too vulnerable. It is better to simultaneously identify a back-up path

that can be used when the primary path fails. To avoid that both primary and backup paths fail, they should be link-disjoint. This leads to the problem of finding two link-disjoint paths that both obey multiple constraints. In one dimension this problem is easily solved, but the extension to multiple dimensions is less trivial. Again, the complicating factor is that subsections of shortest (feasible) paths are not necessarily shortest themselves. A heuristic algorithm DIMCRA is proposed to find link-disjoint multi-constrained paths between a pair of source and destination nodes. If DIMCRA returns a link-disjoint pair of paths, they will always obey the constraints. However, DIMCRA's solution is not necessarily optimal in the sense that it minimizes the total length of the returned paths or guarantees that it always finds a feasible set. Its performance, however, is better than (the intuitive method of) simply finding a feasible path, removing it from the graph and then computing the backup path. Some open issues remain, namely: how to make DIMCRA exact whilst still efficient, how to allow maximally disjoint paths or bridges, and evaluating the performance of other new link-disjoint QoS algorithms appear.

After this extensive algorithmic study, we investigated the complexity of exact QoS routing. Finding a path based on multiple QoS constraints was proven to be an NP-complete problem. However,

> *the Multi-Constrained Path (MCP) selection problem is not NP-complete in the strong sense,*

meaning that a pseudo-polynomial algorithm can exactly solve the problem.

> *The NP-completeness of the MCP problem hinges on at least four factors that must hold simultaneously, namely (1) the underlying topology, (2) link weights that can grow arbitrarily large or have an infinite granularity, (3) a very negative correlation among the link weights, and (4) the values of the constraints.*

If the values of the constraints are very large, then it will be easy to find a path within the constraints. In contrast, if the values of the constraints are very small, then it will be easy to verify that there is no path within the constraints. This indicates that

> *the complexity of QoS routing has a phase transition as a function of the value of the constraints.*

There will be a phase transition if the constraints are around the weights of the m-dimensional shortest path in the network. In this case, it will probably be difficult to establish whether a feasible path exists. If the four above-mentioned conditions are all necessary to induce intractability, they will allow network and service providers

to properly dimension their network and to avoid intractable scenarios. Moreover, if the theory of phase transition holds for the MCP problem, then we know that QoS requirements close to the m-dimensional shortest path will, if admitted, provide the highest possible level of QoS, but will also involve the highest computational cost. Such information is invaluable for pricing and billing mechanisms and admission control algorithms. Finally, a proper understanding and use of the four conditions will allow for efficient QoS routing at controlled computational costs.

This thesis has identified and discussed the theory and key concepts of QoS algorithms and their complexity. As such the main goal of this thesis has been met. However, the theory of QoS routing reaches beyond QoS algorithms and also involves the dynamics of QoS routing. A key question here is "how can we get an accurate view of the state of the network, or keep it up to date, without introducing too much complexity?" Therefore, for completeness, the last chapter of this thesis discusses the main research questions in dynamic QoS routing and presents some pioneering work in this direction. We have provided a preliminary evaluation of the stability of paths in a dynamically changing network.

Network dynamics can be categorized as either slowly or frequently changing.

Slow changes relate to changes in nodes or links and have been studied in the past. Frequent changes relate to changes in the link weights. We have focussed on the influence that dynamic link weights have on path stability. We have provided a mathematical analysis of the difference in length between the shortest path in an unperturbed graph and the shortest path in a perturbed one. Subsequently, we have simulated the difference between these two paths. In practice, the link weights will have a finite granularity instead of the real values used for the simulations. We expect that

a larger granularity will improve the stability of paths and consequently the predictability of network state.

Our results for the difference in links Δl displayed more volatility as a function of the perturbation strength α, than the difference in weights Δw. We believe that the latter parameter is a more important measure to set a threshold for updating the network state.

Finally, we have evaluated several algorithms for dynamically routing flows that have a bandwidth requirement and a number of constraints on additive QoS measures. We have carried out some simulation studies in order to compare SAMCRA with previous proposals, using call and bandwidth blocking rates, throughput and path computation time as performance indicators. Several scenarios have been analyzed, and two versions of SAMCRA were used: SAMCRA and SAMCRA-B, where SAMCRA-B incorporates

the available bandwidth into its length function. In every test the call/bandwidth blocking rate of SAMCRA-B was the minimum or very close to the minimum. The simulations also revealed that the path length function of SAMCRA-B performed much better than SAMCRA with a static length function.

In all the tests, the average path computation time of SAMCRA and SAMCRA-B were among the shortest. From these scenarios, we can conclude that SAMCRA-B performs best and has a low computational cost. The results indicate that SAMCRA, with a properly chosen path length function, is not only an exact and fast QoS routing algorithm, but that it also serves as an effective traffic engineering algorithm that optimizes network throughput. However, more simulations are needed to confirm this claim.

Appendix A

Approximate analysis

A.1 Approximate analysis of QoS complexity

We present an approximate analysis of the length of the m-dimensional shortest path in a two-dimensional lattice.

A.1.1 Analysis for a single link weight ($m = 1$)

Consider a rectangular 2d-lattice with size z_1 and z_2 and with independent and identical, uniformly distributed link weights on $(0, 1]$. The shortest-hop path between two diagonal corner points consists of $h = z_1 + z_2$ hops. The weight W_h of such an h hop path is the sum of h independent uniform random variables u_j and $W_h = \sum_{j=1}^{h} u_j$ has distribution,

$$F(x) = \Pr[W_h \leq x] = \frac{1}{h!} \sum_{j=0}^{h} \binom{h}{j} (-1)^j (x-j)^h 1_{j \leq x} \tag{A.1}$$

In particular, $\Pr[W_h \leq h] = 1$ and for small $x < 1$ holds that $F(x) = \frac{x^h}{h!}$. We assume that the number $l = \binom{z_1+z_2}{z_1} = \frac{h!}{z_1! z_2!}$ of those h-hop paths is large. Although these paths can possibly overlap, we ignore this dependence for the moment and assume that the minimum weight among all h-hop paths is well approximated by the limit law for the minimum of a set of independent random variables X_k with identical distribution F. In particular, if $\lim_{l\to\infty} l(F(x_l)) = \zeta$,

$$\lim_{l\to\infty} \Pr\left[\min_{1\leq k\leq l} X_k > x_l\right] = e^{-\zeta} \tag{A.2}$$

The limit sequence must obey $l(F(x_l)) \to \zeta$ for sufficiently large l, which implies that $F(x_l)$ must be small or, equivalently, x_l must be small. Hence, $l\frac{x_l^h}{h!} = \zeta$ or $x_l = \left(\frac{h!\zeta}{l}\right)^{\frac{1}{h}}$.

The limit law (A.2) for the minimum weight $W = \min_{1\le k\le l} W_{h,k}$ of the shortest-hop path between two corner points in a rectangular 2d-lattice is

$$\lim_{l\to\infty} \Pr\left[\min_{1\le k\le l} W_{h,k} > \left(\frac{h!x}{l}\right)^{\frac{1}{h}}\right] = e^{-x}$$

In other words, the random variable $\frac{lW^h}{h!}$ tends to an exponential random variable with mean 1 for large $l = \frac{h!}{z_1!z_2!}$ or

$$\Pr[W \le y] \approx 1 - \exp\left(-\frac{y^h}{z_1!z_2!}\right)$$

The mean shortest weight of an h hop path equals

$$\begin{aligned}\mathbb{E}[W] &= \int_0^\infty (1 - F_W(x))\,dx \approx \int_0^\infty \exp\left(-\frac{x^h}{z_1!z_2!}\right)dx \\ &= \Gamma\left(1+\frac{1}{h}\right)(z_1!z_2!)^{\frac{1}{h}} \end{aligned} \tag{A.3}$$

For a square 2d-lattice where $z_1 = z_2 = \frac{h}{2}$,

$$\mathbb{E}[W] = \Gamma\left(1+\frac{1}{h}\right)\left(\left(\frac{h}{2}\right)!\right)^{\frac{2}{h}}$$

Using Stirling's formula [1, 6.1.38] for the factorial $h! = \sqrt{2\pi}h^{h+\frac{1}{2}}e^{-h+\frac{\theta}{12h}}$, where $0 < \theta < 1$, we finally arrive for large h at

$$\mathbb{E}[W] \simeq \left(\frac{h}{2e}\right)\left(\sqrt{\pi h}e^{\frac{\theta}{6h}}\right)^{\frac{2}{h}} \approx \frac{h}{2e} \tag{A.4}$$

We now provide a heuristic argument why, for large h, the neglect of the dependence between h-hop paths is justified. Denote by Γ_h the set of all h-hop paths in the 2d-lattice between corner points, with the number of those paths $|\Gamma_h| = \binom{h}{z_1}$. A particular path of the set Γ_h is denoted by γ_h. We denote the weight of γ by $w(\gamma)$. Let w_N be the (random) weight of the shortest path between two diagonal corner points in the 2d-lattice with independent uniformly distributed link weights. The event $\{h_N = h, w_N \le z\}$ implies that there is an h-hop path γ_h with weight $w(\gamma_h) \le z$ and, therefore,

$$\begin{aligned}\Pr[h_N &= h, w_N \le z] \le \Pr[\cup_{\gamma\in\Gamma_h}\{w(\gamma) \le z\}] \\ &\le \sum_{all\ \gamma} \Pr[\gamma \in \Gamma_h, w(\gamma) \le z] \end{aligned} \tag{A.5}$$

where the second inequality follows from Boole's inequality ($\Pr[\cup A_j] \leq \sum \Pr[A_j]$). Using the independence of the link and the link weights,

$$\begin{aligned}\Pr[h_N &= h, w_N \leq z] \leq \sum_{all\ \gamma} \Pr[\gamma \in \Gamma_h] \Pr[w(\gamma) \leq z] \\ &= \mathbb{E}\left[|\Gamma_h|\right] \Pr[w(\gamma_h) \leq z]\end{aligned}$$

or since $\Pr[w(\gamma_h) \leq z] = \Pr\left[W_h \leq z\right]$, given by (A.1),

$$\Pr[h_N = h, w_N \leq z] \leq \binom{h}{z_1} F\left(z\right)$$

From this rigorous inequality we infer the heuristic argument $\Pr[h_N = h, w_N \leq z] \simeq \binom{h}{z_1} F\left(z\right)$. For a typical value of z, the probabilities should sum to 1, yielding,

$$1 = \sum_{j=0}^{\infty} \Pr[h_N = h + 2j, w_N \leq z] \simeq F\left(z\right) \binom{h}{z_1}$$

where the assumption is that $\sum_{j=1}^{\infty} \Pr[h_N = h + 2j, w_N \leq z] << \Pr[h_N = h, w_N \leq z]$. Hence, a *typical* value for the weight of the shortest path is the solution of $F\left(z\right) = \frac{1}{\binom{h}{z_1}}$. For small z, we have $F\left(z\right) = \frac{z^h}{h!}$ such that

$$z \sim \left(z_1! z_2!\right)^{\frac{1}{h}}$$

which agrees with $\mathbb{E}\left[W\right]$ in (A.3).

A.1.2 Analysis for multiple link weights ($m > 1$)

Consider a 2d-lattice where each link is specified by a link weight vector $\vec{w} = (w_1, w_2, \ldots, w_m)$. We confine to the case where all link weight components are independent and uniformly distributed. Using the non-linear length of SAMCRA, the length of an h-hop path is computed as

$$l_h(P) = \max_{1 \leq j \leq m} \left\lceil \frac{W_{h,j}}{L_j} \right\rceil \tag{A.6}$$

where each weight per component j is $W_{h,j} = \sum_{n=1}^{h} u_{n,j}$, with distribution F given in (A.1). Since all link weight components are independent,

$$\Pr\left[l_h(P) \leq x\right] = \prod_{j=1}^{m} F\left(L_j x\right)$$

For small x, $\prod_{j=1}^{m} F(L_j x) \approx \prod_{j=1}^{m} \frac{(L_j x)^h}{h!} = \left(\frac{x^h}{h!}\right)^m \prod_{j=1}^{m} L_j^h$. We define an equivalent constraint $L_{eq} = \left(\prod_{j=1}^{m} L_j\right)^{\frac{1}{m}}$. Neglecting the dependence of h-hop paths, due to possible overlap as above, and applying the limit law for the minimum length with $\lim_{l\to\infty} l\left(\prod_{j=1}^{m} F(L_j x_l)\right) = \zeta$ results in

$$\lim_{l\to\infty} \Pr\left[\min_{1\le k\le l} l_{h,k}(P) > \left(\frac{x(h!)^m}{l\,(L_{eq})^{mh}}\right)^{\frac{1}{mh}}\right] = e^{-x}$$

For large $l = \frac{h!}{z_1!z_2!}$, we obtain the approximate distribution of the minimum length, $l_h = \lim_{l\to\infty} \min_{1\le k\le l} l_{h,k}(P)$, of an h-hop path,

$$\Pr[l_h \le y] = 1 - \exp\left(-\frac{h!}{z_1!z_2!}\left(\frac{(L_{eq}y)^h}{h!}\right)^m\right) \tag{A.7}$$

The average length of the shortest h-hop path is, with $(h!)^{\frac{1}{h}} \approx \frac{h}{e}(2\pi h)^{\frac{1}{2h}} \approx \frac{h}{e}$,

$$\begin{aligned}
\mathbb{E}[l_h] &= \int_0^\infty \Pr[l_h > y]\,dy \\
&= \Gamma\left(1+\frac{1}{mh}\right)\frac{(h!)^{\frac{1}{h}}\left(\frac{z_1!z_2!}{h!}\right)^{\frac{1}{mh}}}{L_{eq}} \\
&\approx \frac{h}{eL_{eq}}\left(\frac{z_1!z_2!}{h!}\right)^{\frac{1}{mh}}
\end{aligned}$$

Since all link weight components are independent and equal in distribution, we can interpret $\mathbb{E}[L_{eq}l_h]$ as the weight of the shortest path in m dimensions. For a square lattice, using [1, 6.1.49] $\binom{2z}{z} \approx \frac{2^{2z}}{\sqrt{\pi z}}$, the formula

$$\mathbb{E}[L_{eq}l_h] \approx \frac{h}{e2^{\frac{1}{m}}} \tag{A.8}$$

shows that the weight of the shortest path very slowly increases with m as $2^{-\frac{1}{m}}$ and that for any dimension m, $\frac{h}{e2} \le \mathbb{E}[L_{eq}l_h] \le \frac{h}{e}$.

The variance equals

$$\begin{aligned}
var[l_h] &= \int_0^\infty (y - \mathbb{E}[l_h])^2 d\Pr[l_h \le y] \\
&= \frac{(h!)^{\frac{2}{h}}\left(\frac{z_1!z_2!}{h!}\right)^{\frac{2}{mh}}}{(L_{eq})^2}\left(\Gamma\left(1+\frac{2}{mh}\right) - \Gamma^2\left(1+\frac{1}{mh}\right)\right)
\end{aligned}$$

For large h,

$$\Gamma\left(1+\frac{2}{mh}\right)-\Gamma^2\left(1+\frac{1}{mh}\right)=\frac{\pi^2}{6}\frac{1}{(mh)^2}+O\left(\frac{1}{(mh)^3}\right)$$

Hence,

$$var\left[l_h\right]\approx\frac{\pi^2}{6}\frac{\left(\mathbb{E}\left[l_h(P)\right]\right)^2}{(mh)^2}\rightarrow\frac{\pi^2}{6}\frac{1}{em^2L_{eq}}$$

which is rather small and independent of h, as is common for extremal distributions.

A.1.3 Perfect negative correlation ($m = 2$)

In case of $m = 2$ and perfect negative correlation, the first path weight is $W_{h,1} = \sum_{j=1}^{h} u_j$ and the second is $W_{h,2} = h - \sum_{j=0}^{h} u_j = h - W_{h,1}$. Then,

$$l_h(P)=\max\left[\frac{W_{h,1}}{L_1},\frac{W_{h,2}}{L_2}\right]=\max\left[\frac{W_{h,1}}{L_1},\frac{h-W_{h,1}}{L_2}\right]$$

If $L_1 = L_2 = L_{eq}$, then $L_{eq}l_h(P) \geq \frac{h}{2}$ and if $W_{h,1} \leq x \leq \frac{h}{2}$, then $L_{eq}l_h(P) \geq h - x$, else $\frac{h}{2} \leq L_{eq}l_h(P) \leq x$. Thus, $\Pr\left[\frac{h}{2} \leq L_{eq}l_h(P) \leq z\right]$ equals

$$\begin{aligned}&\Pr\left[\frac{h}{2}\leq W_{h,1}\leq z\right]+\Pr\left[h-z\leq W_{h,1}\leq\frac{h}{2}\right]\\ =\;&F(z)-F\left(\frac{h}{2}\right)+F\left(\frac{h}{2}\right)-F(h-z)\\ =\;&F(z)-F(h-z)\end{aligned}$$

Assuming as before independence of paths, then for the minimum length path holds,

$$\Pr\left[\frac{h}{2}\leq\min_{1\leq k\leq l}L_{eq}l_{h,k}(P)\leq z\right]=1-\prod_l\Pr\left[L_{eq}l_h(P)>z\right]$$

With $\Pr[L_{eq}l_h] = \Pr\left[\frac{h}{2} \leq \min_{1\leq k\leq l} L_{eq}l_{h,k}(P) \leq z_l\right]$,

$$\begin{aligned}1-\Pr\left[L_{eq}l_h\right]&=\exp\left[l\log\left(1-\left[F(z_l)-F(h-z_l)\right]\right)\right]\\&=\exp\left[-l\left[F(z_l)-F(h-z_h)\right]\right]\\&\quad\times\left(1+o\left[hF(z_l)-F(h-z_l)\right]\right)\end{aligned}$$

If $\lim_{l\to\infty} l\left[F(z_l) - F(h - z_l)\right] = \xi$, then $1 - \Pr\left[\frac{h}{2} \leq L_{eq} l_h \leq z_l\right] = e^{-\xi}$. It remains to find z_l in terms of ξ. We rewrite $z_l = \frac{h}{2} + x_l$. For small x_l and with $f(x) = \frac{dF(x)}{dx}$,

$$\begin{aligned}
\xi &= l\left[F\left(\frac{h}{2} + x_l\right) - F\left(\frac{h}{2} - x_l\right)\right] \\
&= l\left[F\left(\frac{h}{2}\right) + f\left(\frac{h}{2}\right) x_l + \right. \\
&\quad \left. -F\left(\frac{h}{2}\right) + f\left(\frac{h}{2}\right) x_l + O\left(x_l^3\right)\right] \\
&= 2lf\left(\frac{h}{2}\right) x_l + o\left(x_l^3\right)
\end{aligned}$$

such that, with the Gaussian approximation for $f\left(\frac{h}{2}\right) \simeq \frac{1}{\sqrt{\frac{\pi h}{6}}}$, and $l = \frac{h!}{z_1! z_2!}$

$$x_h = \frac{\xi}{2lf\left(\frac{h}{2}\right)} = \frac{\xi(z_1! z_2!)}{2h!}\sqrt{\frac{\pi h}{6}}$$

Finally,

$$\Pr\left[\frac{h}{2} \leq L_{eq} l_h \leq \frac{h}{2} + y\right] = 1 - \exp\left(-2\frac{h!}{z_1! z_2! \sqrt{\frac{\pi h}{6}}} y\right) \tag{A.9}$$

from which

$$\begin{aligned}
\mathbb{E}\left[L_{eq} l_h\right] &= \frac{h}{2} + \int_0^\infty \exp\left(-2\frac{h!}{z_1! z_2! \sqrt{\frac{\pi h}{6}}} y\right) dy \\
&= \frac{h}{2} + \frac{(z_1! z_2!)}{2h!}\sqrt{\frac{\pi h}{6}}
\end{aligned} \tag{A.10}$$

Hence, for large h, the average $\mathbb{E}\left[L_{eq} l_h\right]$ rapidly tends to $\frac{h}{2}$, as has been verified through simulations.

A.2 Approximate analysis of path stability

We provide an approximate calculus of the length of the shortest path in a class of graphs with link weight distribution $w_e + \widetilde{w}_{e;\alpha}$, where $w_e = U(0,1)$ and $\widetilde{w}_{e;\alpha} = \alpha U(-0.5, 0.5)$, $\forall e \in E$. We base our calculus on the mathematical framework provided in [164] and use the same notation: we fix the source node s and destination node t and denote by Γ_l the set of all paths in $G = (V, E)$ from s to t with l links or hops. A particular path

of the set Γ_l is denoted by γ_l. The weight of γ is denoted by $w(\gamma)$. h_N denotes the number of hops and w_N the weight of the shortest path between s and t with i.i.d. link weights. The event $X = \{h_N \leq k; w_N \leq z\}$ implies that there is a path $\gamma_l \in \Gamma_l$, $l \leq k$, with weight $w(\gamma_l) \leq z$ and therefore [164]:

$$\begin{aligned} \Pr[X] &\leq \sum_{l=1}^{k} \Pr\left[\cup_{\gamma\in\Gamma_l} w(\gamma) \leq z\right] \\ &\leq \sum_{l=1}^{k} \mathbb{E}\left[|\Gamma_l|\right] \Pr\left[w(\gamma) \leq z\right] \end{aligned} \tag{A.11}$$

where $\mathbb{E}\left[|\Gamma_l|\right]$ refers to the average number of paths from s to t with l hops. If we let k equal the maximum hop count $N-1$, we can rewrite (A.11) as

$$\Pr\left[w_N \leq z\right] \leq \sum_{l=1}^{N-1} \mathbb{E}\left[|\Gamma_l|\right] F_w^{l*}(z) \tag{A.12}$$

where the distribution function $F_w^{l*}(z)$ is the probability that a sum of l independent random variables, each with cdf F_w, is at most z and is given by the l-fold convolution:

$$F_w^{l*}(z) = \int_0^z F_w^{(l-1)*}(z-y) f_w(y)\, dy, \quad l \geq 2$$

and where $F_w^{1*} = F_w$. From the rigorous inequality (A.11) we also infer the heuristic statement

$$\Pr[h_N = k, w_N \leq z] \simeq \mathbb{E}\left[|\Gamma_k|\right] F_w^{k*}(z) \tag{A.13}$$

For a typical value of z, the probabilities in (A.13) should sum to 1, yielding

$$1 \simeq \sum_{k=1}^{N-1} \mathbb{E}\left[|\Gamma_k|\right] F_w^{k*}(z) \tag{A.14}$$

and this equation determines the typical value $z = \mathbb{E}\left[w_N\right]$. Substitution of this result in (A.13) shows that the hop count probability should satisfy

$$\Pr\left[h_N = k\right] \simeq \mathbb{E}\left[|\Gamma_k|\right] F_w^{k*}(\mathbb{E}[w_N]) \tag{A.15}$$

where the event $\{w_N \leq z\}$ is deleted because we substitute the *typical* value of z, so that $\Pr[w_N \leq z]$ is close to 1.

If $F_x(z) = z 1_{0\leq z\leq 1}$ corresponding to $U(0,1)$, then using $(x+y)^n = \sum_{j=0}^{n} \binom{n}{j} x^{n-j} y^j$ and the inverse Laplace transform $\frac{1}{2\pi i}\int_{c-i\infty}^{c+i\infty} \frac{e^{sa}}{s^{n+1}} ds = \frac{a^n}{n!} 1_{\mathrm{Re}(a)>0}$, we find

$$F_x^{l*}(z) = \sum_{j=0}^{\lfloor z \rfloor} \frac{(-1)^j}{j!(l-j)!} (z-j)^l, \quad 0 \leq z \leq l \tag{A.16}$$

while the perturbed situation corresponding to $F_w(z) = U(0,1) + \alpha U(-0.5, 0.5)$ is

$$\begin{aligned} F_w^{l*}(z) &= \frac{1}{\alpha^l} \sum_{j=0}^{l} \frac{l!(-1)^j}{j!(l-j)!} \sum_{k=0}^{l} \frac{l!(-1)^k}{k!(l-k)!} \\ &\quad \cdot \frac{(z - j + \frac{\alpha l}{2} - \alpha k)^{2l}}{(2l)!} 1_{z-j+\frac{\alpha l}{2}-\alpha k>0} \end{aligned} \tag{A.17}$$

Since the link weights are independent and also w_e and $\widetilde{w}_{e;\alpha}$ are independent, the expected weight of a path P in the perturbed graph equals $\mathbb{E}[w(P)] = \sum_{i=1}^{l} \mathbb{E}[w_e + \widetilde{w}_{e;\alpha}] = \frac{l}{2}$ and the variance $var[w(P)] = \sum_{i=1}^{l} var[w_e + \widetilde{w}_{e;\alpha}] = \frac{l(1+\alpha^2)}{12}$. By the central limit theorem, provided l is large enough, we can approximate the distribution (A.17) with the Gaussian distribution $\Phi\left(\frac{z-\mu}{\sigma}\right)$, with $\mu = \mathbb{E}[w(P)] = \frac{l}{2}$ and $\sigma = \sqrt{var[w(P)]} = \sqrt{\frac{l(1+\alpha^2)}{12}}$.

In (QoS) routing it is uncommon that link weights become negative and therefore the negative weights are truncated at 0 (in the simulations we used a small value $\varepsilon = 10^{-5}$). In this case, we want to know $\Pr[w \leq z | w \geq 0] = \frac{\Pr[w \leq z \cap w \geq 0]}{\Pr[w \geq 0]} = \frac{\Pr[w \leq z] - \Pr[w \leq 0]}{\Pr[w \geq 0]} = \frac{F_w^{l*}(z) - F_w^{l*}(0)}{1 - F_w^{l*}(0)}$.

First consider

$$\begin{aligned} \Pr[w \leq z | w \geq 0] &= \frac{F_w^{l*}(z) - F_w^{l*}(0)}{1 - F_w^{l*}(0)} \\ &= \frac{1}{1 - F_w^{l*}(0)} \int_0^z f_w^{(l*)}(u) du \end{aligned}$$

The Taylor expansion around $z = 0$ is

$$\Pr[w \leq z | w \geq 0] = \frac{1}{1 - F_w^{l*}(0)} \sum_{k=1}^{\infty} \frac{d^{k-1}}{dx^{k-1}} f_w^{(l*)}(x) \bigg|_{x=0} \frac{z^k}{k!}$$

If $f_w^{(l*)}(0) > 0$, the approach is as follows. First, $\Pr[w \leq z | w \geq 0] = \frac{f_w^{(l*)}(0)}{1 - F_w^{l*}(0)} z + O(z^2)$. To be able to solve the confining equation (A.14) for a typical value $z \approx \mathbb{E}[w_N]$, we need to know $\mathbb{E}[|\Gamma_l|]$. For the class of random graphs $G_p(N)$ holds that $\mathbb{E}[|\Gamma_l|] = \frac{(N-2)!}{(N-l-1)!} p^l \sim N^{l-1} p^l$ [162], for large N. The weight of a shortest path for random graphs can then be solved from,

$$1 \simeq \frac{z}{N} \sum_{l=1}^{N-1} \frac{f_w^{(l*)}(0)}{1 - F_w^{l*}(0)} (Np)^l$$

such that

$$z \approx \mathbb{E}[w_N] \sim \frac{N}{\sum_{l=1}^{N-1} \frac{f_w^{(l*)}(0)}{1 - F_w^{l*}(0)} (Np)^l} \approx 0$$

This regime corresponds to saturation (α large).

If $\frac{d^{k-1}}{dx^{k-1}} f_w^{(l*)}(x)\Big|_{x=0} > 0$ for some $k > 1$, then $\Pr[w \leq z | w \geq 0] = \beta_l z^k$ and a similar approach as above can be followed, unless $k = l$, in which case we have

$$1 \simeq \frac{1}{N} \sum_{l=1}^{N-1} \frac{\beta_l}{1 - F_w^{l*}(0)} (zNp)^l$$

and the evaluation of the sum is crucial to find z. In that case, $\frac{\beta_l}{1-F_w^{l*}(0)} = O\left(\frac{1}{l!}\right)$ (at least) for the sum to converge. This region corresponds to small α such that $\Pr[w \leq z | w \geq 0] \sim \frac{z^l}{l!}$ nearly independent of α. This region then will be close to $z \sim \frac{\ln(N)}{Np}$, which corresponds to the shortest path weight in the unperturbed random graph.

Appendix B

Abbreviations

ATM	Asynchronous Transfer Mode
BBR	Bandwidth Blocking Rate
BDS	Bi-Directional Search
BFS	Breadth-First Search
BW	Bandwidth
CBR	Call Blocking Rate
CLM	ConnectionLess Multicast
CR-LDP	Constraint-based Routing Label Distribution Protocol
DiffServ	Differentiated Services
DIMCRA	link-DIsjoint Multiple Constraints Routing Algorithm
DVMRP	Distance Vector Multicast Routing Protocol
FIFO	First In First Out
FPTAS	Fully Polynomial Time Approximation Scheme
FttH	Fiber to the Home
GMPLS	Generalized MPLS
HAMCRA	Hybrid Auguring Multiple Constraints Routing Algorithm
HCMB	Hop-Constrained Maximum Bandwidth
IntServ	Integrated Services
IP	Internet Protocol
LBA	Link-disjoint Bhandari Algorithm
LIFO	Last In First Out
LP	Linear Programming
LPP	Link-disjoint Path Pair
LSA	link state Advertisment
LSU	link state Update
MAMCRA	Multicast Adaptive Multiple Constraints Routing Algorithm
MCLPP	Multi-Constrained Link-disjoint Path Pair

MCM	Multiple Constrained Multicast
MCMWM	Multiple Constrained Minimum Weight Multicast
MCOP	Multi-Constrained Optimal Path
MCP	Multi-Constrained Path
MLBA	Multi-constrained LBA
MOSPF	Multicast OSPF
MPLS	MultiProtocol Label Switching
MPST	Multiple Parameter Steiner Tree
NP	Nondeterministic Polynomial
OSPF	Open Shortest Path First
PIM	Protocol Independent Multicast
PNNI	Private Network-to-Network Interface
QoS	Quality of Service
RF	Remove-Find
RIP	Routing Information Protocol
RSP	Restricted Shortest Path
RSVP	Resource reSerVation Protocol
SAMCRA	Self-Adaptive Multiple Constraints Routing Algorithm
SDH	Synchronous Digital Hierarchy
SONET	Synchronous Optical NETwork
SP	Shortest Path
TAMCRA	Tunable Accuracy Multiple Constraints Routing Algorithm
TCP	Transmission Control Protocol
TSpec	Traffic Specification
URT	Uniform Recursive Tree
WDM	Wavelength Division Multiplexing
WFQ	Weighted Fair Queueing

Bibliography

[1] M. Abramowitz and I.A. Stegun. *Handbook of Mathematical Functions.* Dover Publications, Inc, New York, 1968.

[2] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network Flows: Theory, Algorithms, and Applications.* Prentice Hall Inc., 1993.

[3] R. Albert and A.L. Barabasi. Statistical mechanics of complex networks. *Reviews of modern physics*, 74, January 2002.

[4] L.H. Andrew and A.A.N. Kusuma. Generalized analysis of a QoS-aware routing algorithm. *Proc. of IEEE Globecom'98, Piscatawav, N.J. USA*, 1:1–6, 1998.

[5] T. Anjali, C. Scoglio, J.C. de Oliveira, L.C. Chen, I.F. Akyildiz, J.A. Smith, G. Uhl, and A. Sciuto. A new path selection algorithm for MPLS networks based on available bandwidth estimation. *Proc. of QofIS/ICQT 2002, LNCS 2511, Zurich, Switzerland*, pages 205–214, October 16-18 2002.

[6] G. Apostolopoulos, R. Guerin, S. Kamat, and S.K. Tripathi. Quality of service based routing: A performance perspective. *Proc. of the ACM Sigcomm'98 Conference, Vancouver, Britsh Columbia, Canada*, August/September 1998.

[7] G. Apostolopoulos, D. Williams, S. Kamat, R. Guerin, A. Orda, and T. Przygienda. QoS routing mechanisms and OSPF extensions. *IETF RFC 2676*, August 1999.

[8] T. Asano. An O(NloglogN) time algorithm for constructing a graph of maximum connectivity with prescribed degrees. *Journal of Computer and System Sciences*, (51):507–510, 1995.

[9] E. Astiz. *Stability of paths in a dynamic QoS Environment.* MSc thesis, Delft University of Technology (mentor F.A. Kuipers), April 2003.

[10] A. Banerjee, J. Drake, J.P. Lang, B. Turner, K. Kompella, and Y. Rekhter. Generalized multiprotocol label switching: An overview of routing and management enhancements. *IEEE Communications Magazine*, pages 114–150, January 2001.

[11] G. Banerjee and D. Sidhu. Comparative analysis of path computation techniques for MPLS traffic engineering. *Computer Networks*, 40:149–165, 2002.

[12] A.L. Barabasi and R. Albert. Emergence of scaling in random networks. *Science*, (286):509–512, 1999.

[13] O. Barndorff-Nielsen and M. Sobel. On the distribution of the number of admissible points in a vector random sample. *Theory of Probability and its Applications*, 11(2), 1966.

[14] S. Basu, A. Mukherjee, and S. Klivansky. Time series models for internet traffic. *Proc. of IEEE INFOCOM'96*, pages 611–620, 1996.

[15] Y. Bejerano, Y. Breitbart, A. Orda, R. Rastogi, and A. Sprintson. Algorithms for computing QoS paths with restoration. *Proc. of IEEE INFOCOM'03, San Francisco, USA*, April 1-3 2003.

[16] R. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, New Jersey, 1957.

[17] R. Bellman. On a routing problem. *Quarterly of Applied Mathematics*, (16):87–90, 1958.

[18] J.L. Bentley, H.T. Kung, M. Schkolnick, and C.D. Thompson. On the average number of maxima in a set of vectors and applications. *Journal of ACM*, 25(4):536–543, 1978.

[19] R. Bhandari. Optimal diverse routing in telecommunication fiber networks. *Proc. of IEEE INFOCOM'94, Toronto, Ontario, Canada*, 3:1498–1508, June 1994.

[20] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An architecture for differentiated services. *IETF RFC 2475*, December 1998.

[21] B. Bollobás. *Random Graphs*. Cambridge University Press, second edition, 2001.

[22] R. Braden, D. Clark, and S. Shenker. Integrated services in the internet architecture: an overview. *IETF RFC 1633*, June 1994.

[23] K. Carlberg and J. Crowcroft. Building shared trees using a one-to-many joining mechanism. *Computer Communications*, pages 5–11, 1997.

[24] D.A. Castanon. Efficient algorithms for finding the k best paths through a trellis. *IEEE Transactions on Aerospace and Electronic Systems*, 26(2):405–410, March 1990.

[25] I. Castineyra, N. Chiapp, and M. Steenstrup. The nimrod routing architecture. *IETF RFC 1992*, August 1996.

[26] P. Cheeseman, B. Kanefsky, and W.M. Taylor. Where the really hard problems are. *Proc. of IJCAI-91, San Mateo, CA*, pages 331–337, 1991.

[27] S. Chen and K. Nahrstedt. On finding multi-constrained paths. *Proc. of ICC'98, New York*, pages 874–879, 1998.

[28] S. Chen, K. Nahrstedt, and Y. Shavitt. A QoS-aware multicast routing protocol. *Proc. of IEEE INFOCOM'2000, Tel-Aviv, Israel*, March 2000.

[29] C. Cheng, S.P.R. Kumar, and J.J. Garcia-Luna-Aceves. A distributed algorithm for finding k disjoint paths of minimal total length. *Proc. of 28th Annual Allerton Conference on Communication, Control, and Computing, Urbana, Illinois*, October 1990.

[30] B.V. Cherkassky, A.V. Goldberg, and T. Radizik. Shortest paths algorithms: theory and experimental evaluation. *Mathematical Programming, Series A*, (73):129–174, 1996.

[31] B.V. Cherkassky, A.V. Goldberg, and C. Silverstein. Buckets, heaps, lists and monotone priority queues. *Siam Journal on Computing*, (28):1326–1346, 1999.

[32] E.I. Chong, S. Maddila, and S. Morley. On finding single-source single-destination k shortest paths. *Journal of Computing and Information, special issue ICCI'95*, pages 40–47, July 1995.

[33] S.A. Cook. The complexity of theorem-proving procedures. *Proc. of third annual ACM symposium on Theory of Computing (STOC), Shaker Height, Ohio*, 1971.

[34] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *An Introduction to Algorithms*. MIT Press, Boston, 2000.

[35] G.B. Dantzig. On the shortest route through a network. *Mgmt. Sci.*, 6:187–190, 1960.

[36] G.B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, Princeton, New Jersey, 1963.

[37] H. De Neve and P. Van Mieghem. A multiple quality of service routing algorithm for PNNI. *Proc. of IEEE ATM workshop, Fairfax, USA*, pages 324–328, May 26-29 1998.

[38] H. De Neve and P. Van Mieghem. TAMCRA: a tunable accuracy multiple constraints routing algorithm. *Computer Communications*, 23:667–679, 2000.

[39] R. Diestel. *Graph Theory*. Graduate Texts in Mathematics. Springer-Verlag, New York, 1997.

[40] E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, (1):269–271, 1959.

[41] P.E. Dunne, A. Gibbons, and M. Zito. Complexity-theoretic models of phase transition in search problems. *Theoretic Computer Science*, (249):243–263, 2000.

[42] D. Eppstein. Finding the k shortest paths. *SIAM Journal on Computing*, 28(2):652–673, 1998.

[43] P. Erdös and A. Rényi. On random graphs I. *Publ. Math. Debrecen.*, 6:290–297, 1959.

[44] P. Erdös and A. Rényi. On the evolution of random graphs. *Magyar Tud. Akad. Mat. Kutato Int. Kozl.*, 5:17–61, 1960.

[45] F. Ergun, R. Sinha, and L. Zhang. QoS routing with performance-dependent costs. *Proc. of IEEE INFOCOM'2000, Tel-Aviv, Israel*, March 2000.

[46] D. Estrin, D. Farinacci, A. Helmy, D. Thaler, S. Deering, M. Handley, V. Jacobson, C. Liu, P. Sharma, and L. Wei. Protocol independent multicast-sparse mode (pim-sm): Protocol specification. *IETF RFC 2362*, June 1998.

[47] M. Faloutsos, A. Benerjea, and R. Pankaj. QoSMIC: Quality of service sensitive multicast internet protocol. *Proc. of SIGCOMM'98*, September 1998.

[48] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. *Proc. of ACM SIGCOMM'99, Cambridge, Massachusetts*, pages 251–262, 1999.

[49] R.W. Floyd. Algorithm 97 (shortest path). *Communications of the ACM*, (5):345, 1962.

[50] S. Floyd and V. Paxson. Difficulties in simulating the internet. *IEEE/ACM Transactions on Networking*, 9(4):392–403, August 2001.

[51] L.R. Ford and D.R. Fulkerson. *Flows in Networks*. Princeton University Press, Princeton, New Jersey, 1962.

[52] B. Fortz and M. Thorup. Internet traffic engineering by optimizing OSPF weights. *Proc. of IEEE INFOCOM 2000, Tel-Aviv, Israel*, March 2000.

[53] M.L. Fredman and R.E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. Assoc. Comput. Mach.*, (34):596–615, 1987.

[54] G. Gallo and S. Pallottino. Shortest paths algorithms. *Annals of Operations Research*, (13):3–79, 1988.

[55] J.J. Garcia-Luna-Aceves and M. Spohn. Scalable link-state internet routing. *Proc. of IEEE International Conference on Network Protocols (ICNP), Austin, Texas, USA*, October 14-16 1998.

[56] M.R. Garey and D.S. Johnson. Strong NP-completeness results: Motivation, examples and implications. *Journal of the ACM*, 25(3):499–508, July 1978.

[57] M.R. Garey and D.S. Johnson. *Computers and Intractability: A guide to the Theory of NP-Completeness*. ISBN 0-7167-1044-7. W.H. Freeman and Company, San Francisco, 1979.

[58] I.P. Gent and T. Walsh. Analysis of heuristics for number partitioning. *Computational Intelligence*, 14(3):430–452, 1998.

[59] A. Goel, K.G. Ramakrishnan, D. Kataria, and D. Logothesis. Efficient computation of delay-sensitive routes from one source to all destinations. *Proc. of IEEE INFOCOM'01, Anchorage, Alaska*, 2:854–858, April 2001.

[60] G.H. Golub and C.F. Van Loan. *Matrix Computations*. North Oxford Academic, Oxford, 1st edition, 1983.

[61] R. Guerin and A. Orda. QoS routing in networks with inaccurate information: Theory and algorithms. *IEEE/ACM Transactions on Networking*, 7(3):350–364, June 1999.

[62] R. Guerin and A. Orda. Networks with advance reservations: the routing perspective. *Proc. of IEEE INFOCOM'2000, Tel-Aviv, Israel*, March 2000.

[63] R. Guerin and A. Orda. Computing shortest paths for any number of hops. *IEEE/ACM Transactions on Networking*, 10(5):613–620, October 2002.

[64] R. Guerin, D. Williams, and A. Orda. QoS routing mechanisms and OSPF extensions. *Proc. of IEEE Globecom*, 1997.

[65] K.P. Gummadi, M.J. Pradeep, and C.S.R. Murthy. An efficient primary-segmented backup scheme for dependable real-time communication in multihop networks. *IEEE/ACM Transactions on Networking*, 11(1):81–94, February 2003.

[66] L. Guo and I. Matta. Search space reduction in QoS routing. *Proc. of ICDCS'99, Austin, Texas, USA*, June 1999.

[67] L. Guo and I. Matta. Search space reduction in QoS routing. *Computer Networks*, 41:73–88, 2003.

[68] Y. Guo, F.A. Kuipers, and P. Van Mieghem. A link-disjoint paths algorithm for reliable QoS routing. *International Journal of Communication Systems*, 16(9):779–798, November 2003.

[69] F. Harary. *Graph Theory.* Addison-Wesly Publishing Company, Reading, Massachusetts, 1969.

[70] G.H. Hardy, J.E. Littlewood, and G. Polya. *Inequalities.* Cambridge University Press, 2nd edition, 1973.

[71] P.E. Hart, N.J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 2(2):100–107, 1968.

[72] P.E. Hart, N.J. Nilsson, and B. Raphael. Correction to a formal basis for the heuristic determination of minimum cost paths. *SIGART Newsletter*, (37):28–29, 1972.

[73] R. Hassin. Approximation schemes for the restricted shortest path problem. *Mathematics of Operations Research*, 17(1):36–42, February 1992.

[74] B. Hayes. Can't get no satisfaction. *American Scientist*, 85(2):108–112, March-April 1997.

[75] C. Hedrick. Routing information protocol. *IETF RFC 1058*, June 1988.

[76] R. Hekmat and P. Van Mieghem. Degree distribution and hopcount in wireless ad-hoc networks. *Proc. of IEEE ICON-03, Sydney, Australia*, Sept. 28 - Oct. 3 2003.

[77] M.I. Henig. The shortest path problem with two objective functions. *European Journal of Operational Research*, 25:281–291, 1985.

[78] P-H Ho and H.T. Mouftah. Issues on diverse routing for WDM mesh networks with survivability. *Proc. of tenth International Conference on Computer Communications and Networks*, pages 61–66, 1997.

[79] Y. Huang and P.K. McKinley. Tree-based link-state routing in the presence of routing information corruption. *Computer Communications*, 26:691–699, 2003.

[80] F.K. Hwang, D.S. Richards, and P. Winter. *The Steiner Tree Problem*. North-Holland, Amsterdam, 1992.

[81] I. Iliadis and D. Bauer. A new class of online minimum-interference routing algorithms. *Proc. of Networking'02*, LNCS 2345:957–971, 2002.

[82] G. Istrate. Computational complexity and phase transitions. *Proc. of the 15th anual Conference on Computational Complexity - Florence, Italy*, pages 104–114, July 4-7 2000.

[83] G.F. Italiana, R. Rastogi, and B. Yener. Restoration algorithms for virtual bandwidth guaranteed tunnels with restoration. *Proc. of IEEE INFOCOM'02*, 2002.

[84] A. Iwata, R. Izmailov, D. Lee, B. Sengupta, G. Ramamurthy, and H. Suzuki. ATM routing algorithms with multiple QoS requirements for multimedia internetworking. *IEICE Trans. Commun.*, E79-B(8):999–1007, Aug. 1996.

[85] J.M. Jaffe. Algorithms for finding paths with multiple constraints. *Networks*, 14:95–116, 1984.

[86] M. Jain and C. Dovrolis. End-to-end available bandwidth: Measurement methodology, dynamics, and relation with TCP throughput. *Proc. of SIGCOMM-02, Pittsburgh, Pennsylvania, USA*, August 19-23 2002.

[87] A. Jamakovic. *Influence of Path Correlation on the Complexity of QoS Routing*. MSc thesis, Delft University of Technology (mentor F.A. Kuipers), January 2004.

[88] D.B. Johnson. A note on dijkstra's shortest path algorithm. *Journal of the ACM*, 20(3):385–388, 1973.

[89] L.R. Ford (Jr). Network flow theory. *The RAND Corporation, Santa Monica, California*, page 293, August 14 1956.

[90] A. Juttner, B. Szviatovszki, I. Mecs, and Z. Rajko. Lagrange relaxation based method for the QoS routing problem. *Proc. of IEEE INFOCOM'01, Anchorage, Alaska*, 2:859–868, April 2001.

[91] K. Kar, M. Kodialam, and T.V. Lakshman. Minimum interference routing of bandwidth guaranteed tunnels with MPLS traffic engineering applications. *IEEE Journal on Selected Areas in Communications*, 18(12):2566–2579, December 2000.

[92] K. Kar, M. Kodialam, and T.V. Lakshman. Routing restorable bandwidth guaranteed connections using maximum 2-route flows. *Proc. of IEEE INFOCOM'02*, 2002.

[93] R. Karp. Reducability among combinatorial problems. *Complexity of Computer Communications, R. Miller and J. Thatcher (eds.), Plenum Press, New York*, pages 85–103, 1972.

[94] M. Kodialam and T.V. Lakshman. Dynamic routing of bandwidth guaranteed tunnels with restoration. *Proc. of IEEE INFOCOM 2000, Tel-Aviv, Israel*, pages 902–911, 2000.

[95] M. Kodialam and T.V. Lakshman. Restorable dynamic quality of service routing. *IEEE Communications Magazine*, pages 72–81, June 2002.

[96] T. Korkmaz and M. Krunz. Multi-constrained optimal path selection. *Proc. of IEEE INFOCOM'01, Anchorage, Alaska*, pages 834–843, April 2001.

[97] T. Korkmaz and M. Krunz. A randomized algorithm for finding a path subject to multiple QoS requirements. *Computer Networks*, 36:251–268, 2001.

[98] T. Korkmaz and M. Krunz. Routing multimedia traffic with QoS guarantees. *IEEE Transactions on Multimedia*, 5(3):429–443, September 2003.

[99] F.A. Kuipers, T. Korkmaz, M. Krunz, and P. Van Mieghem. Performance evaluation of constraint-based path selection algorithms. *IEEE Network*, to appear.

[100] F.A. Kuipers and P. Van Mieghem. QoS routing: Average complexity and hopcount in m dimensions. *Proc. of QofIS 2001, Quality of future Internet Services: second COST 263 international workshop, Coimbra, Portugal*, pages 110–126, September 24-26 2001.

[101] F.A. Kuipers and P. Van Mieghem. MAMCRA: a constrained-based multicast routing algorithm. *Computer Communications*, 25(8):801–810, May 2002.

[102] F.A. Kuipers and P. Van Mieghem. Bi-directional search in QoS routing. *Proc. Of the fourth COST 263 International Workshop on Quality of Future Internet Services, QofIS2003, edited by G. Karlsson and M.I. Smirnov in Springer LNCS2811, KTH, Stockholm, Sweden*, pages 102–111, October 1-3 2003.

[103] S.W. Lee and C.S. Wu. A k-best path algorithm for highly reliable communication networks. *IEICE Trans. on Commun.*, E82-B(4):580–585, April 1999.

[104] W.C. Lee, M.G. Hluchyi, and P.A. Humblet. Routing subject to quality of service constraints in integrated communications networks. *IEEE Network*, pages 46–55, July/Aug. 1995.

[105] B. Lekovic and P. Van Mieghem. Link state update policies for quality of service routing. *Proc. of IEEE Eight Symposium on Communications and Vehicular Technology in the Benelux (SCVT2001), Delft, The Netherlands*, pages 123–128, October 18 2001.

[106] L.A. Levin. Average case complete problems. *SIAM J. Comput*, 15(1):285–286, 1986.

[107] C-L Li, S.T. McCornick, and D. Simchi-Levi. The complexity of finding two disjoint paths with minmax objective function. *Discrete Applied Mathematics*, 26(1):105–115, January 1990.

[108] W. Liang. Robust routing in wide-area WDM networks. *Proc. of 15th Int. Parallel and Distributed Processing Symposium, San Francisco*, April 2001.

[109] G. Liu and K.G. Ramakrishnan. A*prune: An algorithm for finding K shortest paths subject to multiple constraints. *Proc. of IEEE INFOCOM'01, Anchorage, Alaska*, April 2001.

[110] C-C Lo and B-W Chuang. A novel approach of backup path reservation for survivable high-speed networks. *IEEE Communications Magazine*, March 2003.

[111] D.H. Lorenz. A simple efficient approximation scheme for the restricted shortest path problem. *Operations Research Letter*, 28(5):213–219, June 2001.

[112] D.H. Lorenz, A. Orda, D. Raz, and Y. Shavit. Efficient QoS partition and routing of unicast and multicast. *Proc. of IWQoS 2000*, pages 75–83, June 2000.

[113] Q. Ma and P. Steenkiste. On path selection for traffic with bandwidth guarantees. *Proc. of the IEEE International Conference on Network Protocols (ICNP97), Atlanta, Georgia*, pages 191–202, October 1997.

[114] Q. Ma and P. Steenkiste. Quality-of-service routing for traffic with performance guarantees. *Proc. of IFIP fifth International Workshop on Quality of Service, New York*, pages 115–126, May 1997.

[115] G. Malkin. RIP version 2. *IETF RFC 2453*, November 1998.

[116] R. Monasson, R. Zecchina, S. Kirkpatrick, B. Selman, and L. Troyansky. Determining computational complexity from characteristic phase transitions. *Nature*, 400:133–137, July 8 1999.

[117] E.F. Moore. The shortest path through a maze. *Proceeding of an international Symposium on the Theory of Switching, April 2-5, 1957, Part II, [The Annals of the Computation Laboratory of Harvard University, volume XXX] H. Aihen (ed), Havard University Press, Cambridge, Massachusetts*, pages 285–292, 1959.

[118] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[119] J. Moy. Multicast extensions to OSPF. *IETF RFC 1584*, March 1994.

[120] E.F. Mykytka and C.Y. Cheng. Generating correlated random variates based on an analogy between correlation and force. *Proc. of the 1994 Winter Simulation Conference*, pages 1413–1416, 1994.

[121] S. Nelakuditi, Z-L Zhang, R.P. Tsang, and D.H.C. Du. Adaptive proportional routing: a localized QoS routing approach. *IEEE/ACM Transactions on Networking*, 10(6):790–804, 2002.

[122] T. Nicholson. Finding the shortest route between two points in a network. *The computer journal*, 9:275–280, 1966.

[123] R.G. Ogier, V. Rutenburg, and N. Shacham. Distributed algorithms for computing shortest pairs of disjoint paths. *IEEE Transactions on Information Theory*, 39(2):443–445, March 1993.

[124] E. Oki and N. Yamanaka. A recursive matrix calculation method for disjoint path search with hop link number constraints. *IEICE Trans. Commun.*, E78-B(5):769–774, May 1995.

[125] A. Orda. Routing with end-to-end QoS guarantees in broadband networks. *IEEE/ACM Transactions on Networking*, 7(3):365–374, June 1999.

[126] A. Orda and A. Sprintson. QoS routing: the precomputation perspective. *Proc. of IEEE INFOCOM'2000, Tel-Aviv, Israel*, pages 128–136, March 2000.

[127] A. Orda and A. Sprintson. Efficient algorithms for computing disjoint QoS paths. *Proceedings of IEEE INFOCOM, Hong Kong*, March 2004.

[128] C.H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization - Algorithms and Complexity*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1982.

[129] K. Papagiannaki, N. Taft, Z-L Zhang, and C. Diot. Long-term forecasting of internet backbone traffic: Observations and initial models. *Proc. of IEEE INFOCOM 2003, San Francisco, USA*, March 30 - April 3 2003.

[130] V. Paxson. End-to-end routing behavior in the internet. *IEEE/ACM Transactions on Networking*, 5(5):601–615, October 1997.

[131] J. Pearl. *Heuristics - Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley Publishing Company, Reading, Massacchusetts, 1984.

[132] M. Peyravian and R. Onvural. Algorithm for efficient generation of link-state updates in ATM. *Computer Networks and ISDN Systems*, 29:237–247, 1997.

[133] C.A. Phillips. The network inhibition problem. *Proc. of the 25th Annual ACM Symposium on the Theory of Computing (STOC)*, pages 776–785, May 1993.

[134] D. Pisinger. Algorithms for knapsack problems. *Ph.D. thesis, Dept. Of Computer Science, University of Copenhagen, Denmark*, February 1995.

[135] I. Pohl. Bi-directional search. *Machine Intelligence 6, eds. B. Meltzer and D. Michie, American Elsevier, New York*, pages 127–140, 1971.

[136] A. Puri and S. Tripakis. Algorithms for routing with multiple constraints. *Proc. of AIPS'02, Toulouse, France*, April 23 2002.

[137] S.S. Rao. *Optimization - Theory and Algorithms*. Wiley Eastern Limited, New Delhi, 2nd edition, 1984.

[138] D.S. Reeves and H.F. Salama. A distributed algorithm for delay-constrained unicast routing. *IEEE/ACM Transactions on Networking*, 8(2):239–250, April 2000.

[139] C. Roos, T. Terlaky, and J.-Ph. Vial. *Theory and Algorithms for Linear Optimization: An Interior Point Approach*. John Wiley & Sons, Chichester, UK, 1997.

[140] E. Rosen, A. Viswanathan, and R. Callon. Multiprotocol label switching architecture. *IETF RFC 3031*, January 2001.

[141] G.N. Rouskas and I. Daldine. Multicast routing with end-to-end delay and delay variation constraints. *IEEE Journal on Selected Areas in Communications*, 15(3):346–356, April 1997.

[142] H.L. Royden. *Real Analysis*. Macmillan Publishing Company, New York, 3rd edition, 1988.

[143] H.F. Salama, D.S. Reeves, and Y. Viniotis. Evaluation of multicast routing algorithms for real-time communication on high-speed networks. *IEEE Journal on Selected Areas in Communications*, 15(3):332–345, April 1997.

[144] A. Sang and S-Q Li. A predictability analysis of network traffic. *Computer Networks*, 39:329–345, 2002.

[145] A. Schrijver. *Combinatorial Optimization-Polyhedra and Efficiency*, volume 1-3. Springer-Verlag, Berlin, 2003.

[146] A. Sen, B.H. Shen, S. Bandyopadhyay, and J.M. Capone. Survivability of lightware networks - path lengths in WDM protection scheme. *Journal of High Speed Networks*, 10(4):303–315, 2001.

[147] A. Shaikh, J. Rexford, and K.G. Shin. Evaluating the impact of stale link state on quality-of-service routing. *IEEE/ACM Transactions on Networking*, 9(2), April 2001.

[148] S.Z. Shaikh. Span-disjoint paths for physical diversity in networks. *Proc. of IEEE Symposium on Computers and Communications*, pages 127–133, 1995.

[149] S. Shenker, C. Partridge, and R. Guerin. Specifications of guaranteed quality of service. *IETF RFC 2212*, September 1997.

[150] M-K Shin, Y-J Kim, K-S Park, and S-H Kim. Explicit multicast extension (xcast+) for efficient multicast delivery. *ETRI Journal*, 23(4), December 2001.

[151] D. Sidhu, R. Nair, and S. Abdallah. Finding disjoints paths in networks. *ACM SIGCOMM Computer communication Review, Proc. of the conference on Communications architecture & protocols*, 21(4), August 1991.

[152] R. Solomonoff and A. Rapoport. Connectivity of random nets. *Bull. Math. Biophys.*, (13):107–117, 1951.

[153] W.R. Stevens. *TCP/IP Illustrated*, volume 1, The Protocols. Addison-Wesley, Reading, Massachusetts, 1994.

[154] J.W. Suurballe. Disjoint paths in a network. *Networks*, 4:125–145, 1974.

[155] J.W. Suurballe and R.E. Tarjan. A quick method for finding shortest pairs of disjoint paths. *Networks*, 14:325–333, 1984.

[156] N. Taft-Plotkin, B. Bellur, and R. Ogier. Quality-of-service routing using maximally disjoint paths. *The Seventh International Workshop on Quality of Service (IWQoS99), London, England*, pages 119–128, May/June 1999.

[157] Y. Tanaka, F. Rue-xue, and M. Akiyama. Design method of highly reliable communication networks by use of matrix calculation. *IEICE Trans.*, J70-B(5):551–556, 1987.

[158] The ATM Forum. *Private Network-Network Interface*. af-pnni-0055.000, pnni 1.0 edition, March 1996.

[159] K-C Tsai and C. Chen. Two algorithms for multi-constrained optimal multicast routing. *International Journal of Communication Systems*, 16:951–973, 2003.

[160] A.M. Turing. On computable numbers, with application to the entscheidungsproblem. *Proc. of the London Mathematical Society*, 2(42):230–265, 1937.

[161] P. Van Mieghem. A lower bound for the end-to-end delay in networks: application to voice over IP. *Proc. of IEEE Globecom'98, Sydney (Australia)*, pages 2508–2513, November 8-12 1998.

[162] P. Van Mieghem. Paths in the simple random graph and the waxman graph. *Probability in the Engineering and Informational Sciences (PEIS)*, 15:535–555, 2001.

[163] P. Van Mieghem, H. De Neve, and F.A. Kuipers. Hop-by-hop quality of service routing. *Computer Networks*, 37(3-4):407–423, November 2001.

[164] P. Van Mieghem, G. Hooghiemstra, and R. van der Hofstad. A scaling law for the hopcount in internet. *Technical Report 2000125, Delft University of Technology, http://www.nas.ewi.tudelft.nl/people/piet/telconference.html*, 2000.

[165] P. Van Mieghem, G. Hooghiemstra, and R. van der Hofstad. On the efficiency of multicast. *IEEE/ACM Transactions on Networking*, 9(6):719–732, 2001.

[166] P. Van Mieghem and F.A. Kuipers. On the complexity of QoS routing. *Computer Communications*, 26(4):376–387, March 2003.

[167] P. Van Mieghem and F.A. Kuipers. Concepts of exact quality of service algorithms. *IEEE/ACM Transactions on Networking*, to appear, 2004.

[168] D. Waitzman, C. Partridge, and S. Deering. Distance vector multicast routing protocol. *IETF RFC 1075*, November 1988.

[169] B. Wang, X. Su, and C. Chen. A new bandwidth guaranteed routing algorithm for MPLS traffic engineering. *Proc. of IEEE International Conference on Communications, ICC'02*, 2002.

[170] Z. Wang. On the complexity of quality of service routing. *Information Processing Letters*, 69:111–114, 1999.

[171] Z. Wang and J. Crowcroft. Quality of service routing for supporting multimedia applications. *IEEE Journal on Selected Areas in Communications*, 14(7):1228–1234, September 1996.

[172] A. Warburton. Approximation of pareto optima in multi-objective, shortest path problems. *Operations Research*, 1:70–79, 1987.

[173] S. Warshall. A theorem on matrices. *Journal of the ACM*, 9(1):11–12, 1962.

[174] J.K. Wolf, A.M. Viterbi, and G.S. Dixon. Finding the best set of k paths through a trellis with application to multitarget tracking. *IEEE Transactions on Aerospace and Electronic Systems*, 25(2):287–296, March 1989.

[175] J. Wroclawski. Specification of the controlled-load network element service. *IETF RFC 2211*, September 1997.

[176] X. Xiao and L.M. Ni. Internet QoS: A big picture. *IEEE Network*, 13(2):8–18, March-April 1999.

[177] G. Xue, A. Sen, and R. Banka. Routing with many additive QoS constraints. *Proc. of ICC 2003, Anchorage, Alaska, USA*, pages 223–227, May 11-15 2003.

[178] C. You and K. Chandra. Time series models for internet data traffic. *Proc. of 24th conference on local computer networks*, pages 164–171, October 1999.

[179] X. Yuan and X. Liu. Heuristic algorithms for multi-constrained quality of service routing. *Proc. of IEEE INFOCOM'01, Anchorage, Alaska*, April 2001.

[180] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala. RSVP: A new resource reservation protocol. *IEEE Network*, 7(5):8–18, September 1993.

Samenvatting (Summary in Dutch)

Titel: Routeren in het Internet met kwaliteitsgaranties: Theorie, Complexiteit en Algoritmes

(Quality of Service Routing in the Internet: Theory, Complexity and Algorithms)

Dagelijks vinden enorm veel data pakketjes hun weg over het Internet naar hun bestemming. Het Internet bestaat uit vele netwerkelementen die als taak hebben om deze pakketjes het correcte pad naar hun bestemming te wijzen. Het vinden en volgen van het pad naar de bestemming wordt 'routeren' genoemd. Het proces van routeren is niet feilloos, waardoor er pakketjes verloren kunnen gaan. Het huidige Internet kan geen garanties geven aan de pakketjes die het transporteert. Zo kunnen er geen garanties worden gegeven omtrent de vertraging die de pakketjes ondervinden of over de variantie in de vertraging, de verlieskans van pakketjes en de beschikbare bandbreedte van het te volgen pad. Tegenwoordig zijn er echter vele nieuwe multimedia applicaties, welke zonder dergelijke garanties niet goed kunnen functioneren. Bijvoorbeeld, om een gesprek over het Internet te kunnen voeren, moet de vertraging begrensd worden. Het vinden van paden die verschillende garanties kunnen geven, leidt tot een nieuwe vorm van routeren: 'Quality of Service (QoS)', oftewel kwaliteit van de dienstverlening, routeren genoemd.

De beoogde doelen van deze thesis zijn:

1) het analyseren van de algoritmische concepten die aan QoS routeren ten grondslag liggen,

2) het onderzoeken van de complexiteit van QoS routeren,

3) het bespreken van de dynamica in QoS routeren.

De eerste drie hoofdstukken geven een formele definitie van de problemen die in ogenschouw worden genomen. Ze leggen de gebruikte notatie vast en geven het achtergrondmateriaal dat noodzakelijk is om de thesis volledig te kunnen begrijpen. de volgende twee definities (Hoofdstuk 2) zijn onderdeel van dit achtergrondmateriaal:

Algoritme: *Een algoritme voert een berekening uit volgens een duidelijk omlijnde procedure met als invoer een waarde of een set van waarden en als uitkomst ook een waarde of set van waarden. Een algoritme is dus een sequentie van tussenberekeningen die de invoer transformeren tot een uitkomst.*

Complexiteit: *Complexiteit verwijst naar het intrinsiek minimum aantal hulpbronnen die aangewend moeten worden voor het oplossen van een probleem of het uitvoeren van een algoritme.*

De theorie van NP-complexiteit wordt ook uitgelegd. QoS routeren is bewezen NP-compleet te zijn, hetgeen betekent dat exacte algoritmes in het ergste geval een complexiteit nodig hebben die niet kan worden begrensd door een polynomiale functie.

Om QoS algoritmes volledig te kunnen begrijpen, is het nodig om bekend te zijn met de simpele (één-dimensionale) kortste pad algoritmes. Hoofdstuk 3 levert deze kennis door breadth-first search, depth-first search, het Bellman-Ford algoritme, het Dijkstra algoritme, bi-directional search, het A* algoritme en mathematisch programmeren te bespreken. Een belangrijke eigenschap van dergelijke kortste pad algoritmes is dat subpaden van kortste paden in één dimensie ook zelf kortste paden zijn. Na de introducerende hoofdstukken komen we (in Hoofdstuk 4) tot de kern van de thesis, namelijk de concepten die aan QoS routeren ten grondslag liggen. QoS routeren op basis van meerdere eisen zorgt ervoor dat subpaden van kortste paden in meerdere dimensies niet noodzakelijkerwijs zelf ook kortste paden zijn. Het kan daarom nodig zijn om gedurende de berekening van QoS paden verscheidene subpaden in acht te nemen. Dit heeft duidelijk consequenties voor de zoekruimte, die nu exponentieel in omvang kan toenemen. Om deze zoekruimte in te perken zijn er twee technieken, non-dominance (niet-gedomineerd) en look-ahead (vooruit kijken), in het SAMCRA algoritme verwerkt. SAMCRA is een exact QoS algoritme dat door ons ontwikkeld is. Naast SAMCRA bestaan er nog veel meer QoS algoritmes, waarvan het merendeel een heuristiek is. Hoofdstuk 5 bespreekt deze algoritmes en levert de eerste grootschalige evaluatie naar hun prestaties. Op basis van deze studie wordt geconcludeerd dat het SAMCRA algoritme (of de algoritmes die hierop lijken) het best presteert.

Hoofdstukken 6 en 7 wijken enigszins af van de rode draad van de thesis. Ze bespreken een aanvulling op QoS routeren. Allereerst wordt multicast QoS routeren behandeld. Multicast routeren is bedoeld voor de communicatie tussen één zender en meerdere ontvangers. Bij multicast routeren worden pakketjes alleen op de noodzakelijke punten vermenigvuldigd, wat ervoor zorgt dat er minder pakketjes worden verzonden dan als er tussen elk zender-ontvanger paar afzonderlijk een communicatie sessie wordt opgezet. Multicast QoS routeren berust ook op dit principe, maar de winst in efficiëntie kan minder groot zijn dan in één dimensie. We hebben het algoritme MAMCRA ontwikkeld voor multicast QoS routeren.

In Hoofdstuk 7 is QoS routeren middels gescheiden verbindingen het onderwerp. Het probleem is om twee paden te vinden die geen enkele verbinding (communicatielijn) gemeen hebben. Het vinden van deze twee paden is belangrijk voor een betrouwbare communicatie. Als bijvoorbeeld het eerste pad zou komen te vervallen, dan kan onmiddellijk worden overgeschakeld naar het tweede pad. De twee paden zouden tevens gebruikt kunnen worden om de netwerkbelasting te verdelen. Net als in Hoofdstuk 6 (multicast QoS routeren) bespreken we de problemen rond QoS routeren middels

gescheiden verbindingen en hebben we een algoritme DIMCRA voor het probleem ontwikkeld.

Hoofdstukken 4 t/m 7 hebben uitvoerig bijgedragen aan het eerste doel van deze thesis, namelijk het analyseren van de concepten die aan QoS routeren ten grondslag liggen. Het tweede doel is het onderzoeken van de complexiteit van QoS routeren, welke wordt behandeld in Hoofdstuk 8. Hoofdstuk 8 beargumenteert dat de complexiteit van QoS routeren behapbaar is in de praktijk en dat moeilijke scenario's alleen voorkomen als het netwerk aan vier condities voldoet ten aanzien van: (1) de onderliggende topologie, (2) de grootte van de QoS gewichten, (3) de (negatieve) correlatie tussen de QoS gewichten en (4) de waarden van de QoS eisen.

Het derde en laatste doel van de thesis is het bespreken van de dynamische aspecten in QoS routeren. Hoofdstuk 9 bespreekt de dynamica van QoS routeren en komt met enkele nieuwe resultaten op dit gebied. De belangrijkste onderzoeksvragen worden duidelijk gekenschetst en komen in essentie allemaal neer op de vraag hoe we het netwerk geïnformeerd kunnen houden over de staat van zijn QoS gewichten. Het werk en de simulaties die worden gepresenteerd geven enig inzicht in de stabiliteit van paden en in de prestatie van SAMCRA in een dynamisch netwerk. Wederom blijkt SAMCRA de andere algoritmes te overtreffen. De conclusies van Hoofdstuk 9 moeten echter worden gezien als indicatoren. Meer simulaties zijn nodig om de conclusies te bevestigen.

Auteur: Fernando A. Kuipers

Acknowledgements

I would like to thank everybody who has positively contributed to, or supported, my work. Their impact on my life has helped to form me into who I am today and accordingly has enriched my thesis.

In particular, I am very grateful to my supervisor and mentor Prof. Piet Van Mieghem. Under his watchful eye and thanks to his expert guidance I was able to complete both my M.Sc. thesis and Ph.D. thesis. His involvement, support, enthusiasm, cleaver insights, and our discussions have been invaluable. I feel privileged an honoured to have worked under his guidance.

I would also like to thank the members of my Ph.D.-committee: Prof. Ignas Niemegeers, Prof. Nico Baken, Prof. Cees Roos, Prof. Jordi Domingo-Pascual, Prof. Giorgio Ventre and dr. Hans De Neve, for reading my manuscript. The knowledge they combine is huge and I feel blessed to have been able to sample from this great pool of wisdom.

As a tree flourishes in fertile soil, similarly, a Ph.D. thesis benefits from a healthy work environment. In the NAS (and WMC) group I found such an environment. I have enjoyed the contact with my colleagues and friends from these groups.

During my Ph.D. I had the opportunity to work with many people. I am obliged to all of them. My gratitude especially extends to Turgay Korkmaz, Marwan Krunz, Yuchun Guo and Stefano Avallone, because the joint work with them has been partly incorporated in this thesis. Also the work of the nine students that I have guided, is equally appreciated. From those students, the theses of Sergi Calvet Ceballos, Selma Begtasevic, Eguzki Astiz Lezaun and Almerima Jamakovic contain many simulation results relevant to the contents of this thesis.

Finally, I would like to thank my family. Malcolm X once said "Education is our passport to the future, for tomorrow belongs only to the people who prepare for it today." I am indebted to my parents, for they have provided me the means to obtain my passport. I feel confident now to take on my travel into the future.

Thank you, Bedankt, Obrigado!

Curriculum Vitae

Personal details

Title(s), name	Ir Fernando Antonio Kuipers
Male/female	male
Date and place of birth	May 16, 1977, The Hague
Nationality	Dutch and Brazilian

Brief summary of research over the period 2000-2004

Fernando A. Kuipers received the M.Sc. degree in Electrical Engineering at Delft University of Technology in June, 2000. The M.Sc. thesis was entitled "Hop-by-hop destination based routing with Quality of Service constraints." He has been enrolled for three months at Alcatel CRC in Antwerp, Belgium to gain "industrial experience." There he worked on balancing the load in a network by means of a genetic algorithm. He was also a member of the DIOC (interdisciplinary research center, now speerpunt) on the Design and Management of Infrastructures, headed by Prof. Margot Weijnen, where he participated in the Telecommunications project. His Ph.D. work mainly focused on the algorithmic aspects, theory and complexity of Quality of Service (QoS) routing. During his Ph.D. he supervised 9 students.

International activities

Program committee member of The First International Workshop on QoS Routing (WQoSR), October 1, 2004 (co-located with the Fifth International Workshop on Quality of Future Internet Services (QoFIS'04)).

Other academic activities

- Reviewer for many journals and conferences, among which: IEEE/ACM Transactions on Networking, IEEE Communications Magazine, Computer Networks, Computer Communications, IEEE INFOCOM and Qofis.
- Lecturer of a class on QoS Routing, and of a Masterclass (organized by Prof. N. Baken) on the topic QoS Routing. Coordinator of the home-work exercises for a class on Performance Evaluation.

- Attended four IETF Masterclasses on: "Challenge and response: towards tomorrow's Internet" by Brian Carpenter, "Internet Architectural Philosophy and the new business reality" by Scott Bradner, "Security" by Jeff Schiller, "IPv6" by Steve Deering.

- Participated in the COST 279 Second European Summer School on "Routing and Multi-Layer Traffic Engineering in Next Generation IP Networks" at Darmstadt, Germany, September 8-12, 2003.

- Followed the EIDMA minicourse on "Approximation Schemes for NP-hard Geometric Problems," by Prof. Sanjeev Arora at the Euler Institute for Discrete Mathematics and its Applications (EIDMA), Technical University of Eindhoven, The Netherlands, September 1-5, 2003.

- Attended the ATHENS (Advanced Technological Higher Education Network Socrates) course in Paris, titled "Discovering the datacommunication networks of the information society of the future," November 12-21, 1999.

Nominations, scholarships and prizes

- Nominated for best paper award at INFOCOM 2003, San Francisco, April 2003.

- Nominated for the Runner-Up award 2001. This is an award for upcoming talent in the telecommunications, media and Internet sector, organized by Telecombrief at the 7th Telecongres in the Netherlands, January 15, 2002.

Publications

Books, or contributions to books:

- F.A. Kuipers, *Hop-by-hop routing with QoS constraints*, M.Sc. thesis in Electrical Engineering at Delft University of Technology, June 2000.

- P. Van Mieghem (ed.), F.A. Kuipers, T. Korkmaz, M. Krunz, M. Curado, E. Monteiro, X. Masip-Bruin, J. Solé-Pareta, and S. Sánchez-López, *Quality of Service Routing*, Chapter 3 in Quality of Future Internet Services, EU-COST 263 Final Report, edited by Smirnov et al. in Springer LNCS 2856, pp. 80-117, 2003.

International journals:

- P. Van Mieghem, H. De Neve and F.A. Kuipers, "Hop-by-hop Quality of Service Routing," Computer Networks, vol. 37/3-4, pp. 407-423, November 2001.

- F.A. Kuipers and P. Van Mieghem, "MAMCRA: a constrained-based multicast routing algorithm," Computer Communications, vol. 25/8, pp. 801-810, May 2002.

- F.A. Kuipers, T. Korkmaz, M. Krunz and P. Van Mieghem, "An Overview of Constraint-Based Path Selection Algorithms for QoS Routing," IEEE Communications Magazine, vol. 40, no. 12, December 2002.

- P. Van Mieghem and F.A. Kuipers, "On the Complexity of QoS Routing," Computer Communications, special issue on QofIS'01, vol. 26, no. 4, pp. 376-387, March 2003.

- Y. Guo, F.A. Kuipers and P. Van Mieghem, "A Link-Disjoint Paths Algorithm for Reliable QoS Routing," International Journal of Communication Systems, vol. 16, no. 9, pp. 779-798, November 2003.

- P. Van Mieghem and F.A. Kuipers, "Concepts of Exact Quality of Service Algorithms," to appear in IEEE/ACM Transaction on Networking.

- F.A. Kuipers, T. Korkmaz, M. Krunz and P. Van Mieghem, "Performance Evaluation of Constraint-Based Path Selection Algorithms," to appear in IEEE Network.

Refereed conference proceedings:

- F.A. Kuipers and P. Van Mieghem, "QoS Routing: Average Complexity and hop count in m dimensions," Proc. of Second COST 263 International Workshop, QofIS'01, Coimbra, Portugal, pp. 110-126, September 24-26, 2001.

- M. Janic, F.A. Kuipers, X. Zhou and P. Van Mieghem, "Implications for QoS Provisioning based on Traceroute Measurements," Proc. of 3rd International Workshop on Quality of Future Internet Services, QofIS'02, Zurich, Switzerland, October 16-18, 2002.

- F.A. Kuipers and P. Van Mieghem, "The Impact of Correlated Link Weights on QoS Routing," Proc. of IEEE INFOCOM 2003, San Francisco, USA, March 30 - April 3, 2003 (nominated for best paper award).

- F.A. Kuipers and P. Van Mieghem, "Bi-directional Search in QoS Routing," Proc. of the fourth COST 263 International Workshop on Quality of Future Internet Services, QofIS'03, edited by G. Karlsson and M. I. Smirnov in Springer LNCS2811, KTH, Stockholm, Sweden, pp. 102-111, October 1-3, 2003.