Research Report No. 2007:09

Optimization Algorithms with Applications to Unicast QoS Routing in Overlay Networks

Dragos Ilie

Department of Telecommunication Systems,
School of Engineering,
Blekinge Institute of Technology,
S–371 79 Karlskrona, Sweden

Blekinge Institute of Technology
Research Report No. 2007:09
ISSN 1103-1581
Published 2007
Printed by Kaserntryckeriet AB
Karlskrona 2007
Sweden

This publication was typeset using LaTeX.

Abstract

The research report is focused on optimization algorithms with application to quality of service (QoS) routing. A brief theoretical background is provided for mathematical tools in relation to optimization theory.

The rest of the report provides a survey of different types of optimization algorithms: several numerical methods, a heuristics and a metaheuristic. In particular, we discuss basic descent methods, gradient-based methods, particle swarm optimization (PSO) and a constrained-path selection algorithm called Self-Adaptive Multiple Constraints Routing Algorithm (SAMCRA).

Contents

List of Algorithms

Chapter 1

Introduction

The path towards establishing the Internet as a platform for multimedia content distribution has shown to be a source of great challenges for the Internet research community. The transition from Internet Protocol (IP)'s *best-effort* service to a service that incorporates the notion of QoS is probably one of the biggest challenge to solve before the Internet can become a reliable platform for multimedia services.

Multimedia services have requirements on the underlaying network that go beyond what shortest-path routing can offer. In particular, live or interactive multimedia communications have stringent constraints on the path between sender and receiver. Examples of such constraints are bottleneck bandwidth, packet delay, packet delay variation and packet loss rate.

The goal of the research report is to survey several algorithms that can be used for selecting a path or sets of paths subject to multiple constraints. We focus on algorithms that can be used online, in the sense that they have the ability to react almost in real-time to relevant events in the network. Furthermore, we are also planning to implement these algorithms in an overlay network. This places additional requirements on the performance of the algorithms.

The report is organized as follows. The remainder of this chapter discusses the notion of QoS, existing QoS architectures for IP networks and the main characteristics of QoS routing. Additionally, we provide an overview of the Overlay Routing Protocol (ORP), which is a framework for unicast QoS routing under development at Blekinge Institute of Technology (BTH) in Karlskrona, Sweden.

Chapter 2 outlines the theoretical background required for the remainder of the report. The chapter begins with definitions and notations for matrix algebra, graph theory and optimization theory. We conclude the chapter with a discussion about algorithms and computation complexity.

In Chapter 3 we describe basic descent methods such as steepest descent method and Netwon's method, which are fundamental algorithms in optimization theory.

The remainder of the report allocates one chapter for each optimization method considered: conjugate gradient, simplex method, gradient projection, PSO and SAMCRA.

1.1 Quality of Service

QoS is one of the most debated topics in the areas of computer network engineering and research. It is generally understood that a network that provides QoS has the ability to

allocate resources for the purpose of implementing services better than best-effort. The major source of debate is on how to provide QoS in IP-based networks [Wan00, PD00].

At one extreme of the debate it is argued that no new mechanisms are required to provide QoS in the Internet, and simply increasing the amount of available bandwidth will suffice.

The people at the other extreme of the debate point out that it is doubtful that bandwidth over-provisioning alone takes care of QoS issues such as packet loss and delay. History has shown that whenever bandwidth has been added to the networks, new "killer" applications were developed to use most of it. Furthermore, over-provisioning may not be an economically viable solution for developing countries and in the long run it may prove to be very expensive even for developed countries.

The first proposed QoS architectures to be used on top of IP is called Integrated Services (IntServ) [BCS94]. IntServ senders allocate resources along a path using the Resource Reservation Protocol (RSVP) [ZDE+93, Wro97]. IntServ performs *per-flow* resource management. This has led to skepticism towards IntServ's ability to scale, since core routers in the Internet must handle several hundred thousands flows simultaneously [TMW97]. A newer report [FML+03] corroborates this number. However, the authors of the report argue that per-flow management is feasible in these conditions due to advances in network processors, which allow over a million concurrent flows to be handled simultaneously.

A new architecture called Differentiated Services (DiffServ) [BBC+98] was developed, due to concerns about IntServ's scalability. DiffServ attempts to solve the scalability problem by dividing the traffic into separate forwarding classes. Each forwarding class is allocated resources proportional to the service level expected by the customer (*e. g.,* bronze, silver, gold and platinum). Packets are classified and mapped to a specific forwarding class at the edge of the network. Inside the core, the routers handle the packets according to their forwarding class. Since routers do not have to store state information for every flow, but only have to peek at certain locations in the packet header, it is expected that DiffServ scales much better than IntServ. A major problem with the DiffServ architecture is that a specific service level may get different treatments among service providers. For example, the gold service level of one provider may be equivalent to the silver level of another provider.

Neither architecture has been widely deployed to date due to reasons amply discussed in [Arm03, Bel03, BDH+03]. Some important issues include the lack of a viable economical solution for network operators, poor backwards compatibility with existing technology and difficulties in the interaction between different network operators.

From a hierarchical point of view, Internet consists of several autonoumous systems (ASs). Each AS consists of a number of interconnected networks administered by the same authority. Within an AS routing is performed using intra-domain routing protocols such as Routing Information Protocol (RIP) and Open Shortest Path First (OSPF). Interconnected ASs exchange routing information using Border Gateway Protocol (BGP). An AS connects to other ASs through peering agreements. A peering agreement is typically a business contract stipulating the cost of routing traffic across an AS along with other policies to be maintained. When there are several routes to a destination the peering agreements force an AS to prefer certain routes over others. For example, given two paths to a destination where the first one is shorter (in terms of hops) and the second one is cheaper, the AS will tend to select the cheaper path. This is called *policy routing* and is one of the reasons for suboptimal routing [Hui00, KR01]. With the commercialization of the Internet it is unlikely that problems related to policy routing will disappear in the near future.

1.2 Overlay Networks

Since neither QoS architecture has seen a wide deployment in the Internet, there have been several attempts to provide some form of QoS using overlay networks. An overlay network utilizes the services of an existing network in an attempt to implement new or better services. An example of an overlay network is shown in Figure 1.1. The physical interconnections of three ASs are depicted at the bottom of the figure. The grey circles denote nodes that use the physical interconnections to construct virtual paths used by the overlay network at the top of the figure.

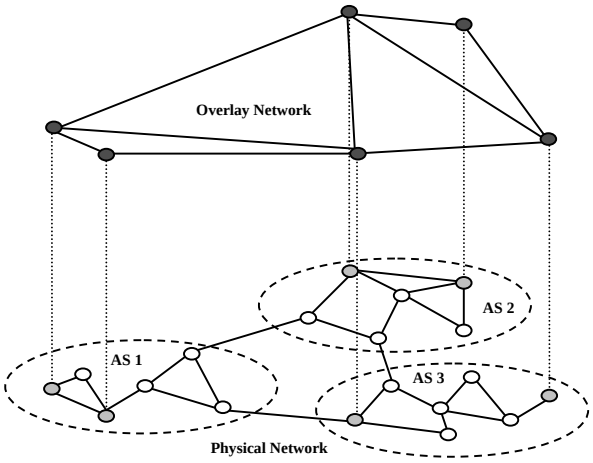

Figure 1.1: Overlay network

The nodes participating in the overlay network perform active measurements to discover the QoS metrics associated with the virtual paths. Assume that an overlay node in AS1 wishes to communicate with another overlay node in AS2. Assume further that AS1 always routes packets to AS2 by using the direct link between them, due to some policy or performance metric. The overlay node in AS1 may discover through active measurements that a path which crosses AS3 can actually provide better QoS, say smaller delay, than the direct link. In that specific case the AS1 node forwards its traffic to the AS3 node, which in turn forwards the traffic to the destination node (or to the next node on the path if multiple hops are necessary). This is the basic idea behind QoS routing in overlays. There are currently several implementations of this idea, such as [And01, LM04]).

These ideas can be further extended to multicast routing. There are other ways to improve the QoS in overlay networks without employing routing [SSBK04]. However, these topics are not within the scope of this report. Our main interest is in optimization algorithms for unicast QoS routing in overlay networks.

1.3 Overlay Routing Protocol

The project Routing in Overlay Networks (ROVER) pursued by BTH aims at developing a platform to facilitate development, testing, evaluation and performance analysis of different solutions for overlay routing, while requiring minimal changes to the applications making use of the platform [IP07]. The goal is to do this by implementing a middleware system, exposing

two set of APIs – one for application writers, and another for interfacing various overlay solutions.

The ROVER architecture is shown in Figure 1.2. The top layer represents various protocols and applications using the ROVER application programming interface (API). The middle layer is the ROVER middleware with associated API. Finally, the bottom layer represents various transport protocols that can be used by the ROVER middleware. Only the left box, denoted ORP, in the top layer in the figure, is within the scope of this report. Overlay Routing Protocol (ORP) is a framework that allows us to study various types of problems and solutions related to unicast QoS routing.

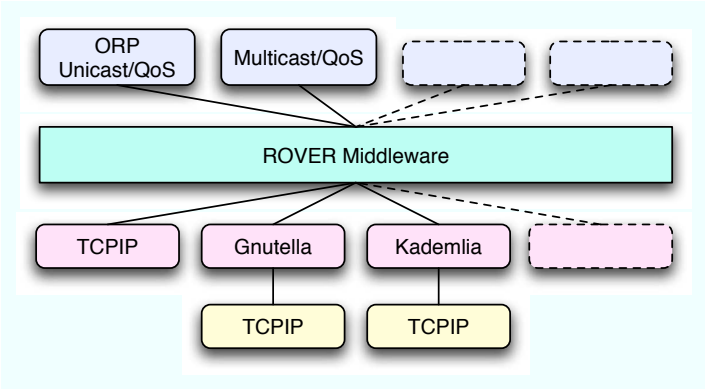

Figure 1.2: ROVER

ORP is part of a larger goal to research and develop a QoS layer on top of the transport layer. The main idea is to combine ORP together with additional QoS mechanisms, such as resource reservation and admission control, into a QoS layer. User applications that use the QoS layer can obtain soft QoS guarantees. These applications run on end-hosts without any specific privileges such as the ability to control the internals of TCP/IP stack, the operating system, or other applications that do not use the QoS layer. Nodes in the ORP overlay use the User Datagram Protocol (UDP) to transport application data, similar to the solution reported in [And01]. In terms of the OSI protocol stack, the QoS layer is a sub-layer of the application layer. Applications may choose to use it or to bypass it.

The QoS layer implements *per-flow* QoS resource management. In contrast to IP routing, we envision that it is mostly end-nodes in access networks that take part in the routing protocol. IP routers are not required to take part or be aware of the QoS routing protocol running on the end-nodes. In other words, we propose a QoS layer on top of the best-effort service provided by IP. Since a best-effort service leaves room for uncertainties regarding the resource allocation, we target only for soft QoS guarantees.

ORP requires that nodes interested in performing QoS routing form an application-layer overlay. The overlay may be structured (*i. e.* , a DHT) or unstructured. The only requirements for it are the ability to forward messages and to address individual nodes through some form of universally unique identifier (UUID) .

The type of services considered for the QoS layer are currently restricted to those that require interactive and non-interactive live unicast multimedia streams. In the future, we will consider other service types as well (*e. g.* , multicast).

By a multimedia stream, we mean a stream containing audio, video, text (*e. g.* , subtitles

or Text-TV), control data (*e. g.,* synchronization data), or a combination thereof. If an application chooses to use several media streams (*e. g.,* one stream per media type), the QoS routing protocol treats them independently of each other and assumes that the application is capable on its own of performing synchronization or any other type of stream merging processing.

The multimedia streams within the scope of ORP are of unicast type, *i. e.,* point-to-point (one-to-one). Multicast streams (one-to-many) are subject for later research. Furthermore, the streams we consider are live, which means that the receiver is not willing to wait until the whole stream data is received, but would rather start watching and listening to it as soon as enough data is available for rendering.

By interactive multimedia streams, we mean streams generated by user interaction as in a video conference or a voice over IP (VoIP) call. Conversely, non-interactive multimedia streams do not involve any interaction between users as is the case of Internet TV or music streaming.

Applications on top of the QoS layer request overlay paths to certain destinations, along with specific constraints attached to each path (*e. g.,* minimum bandwidth required, maximum delay and delay jitter tolerated). We expect the source nodes to compute the feasible path for each flow that originates from them. The path information is later communicated to the nodes on the corresponding path as part of the ORP operation. Essentially, all source nodes compete among each other for overlay resources.

We assume that each node is capable of estimating its available host resources (*e. g.,* RAM, storage) as well as link properties (*e. g.,* residual bandwidth, round-trip time (RTT)) to its one-hop neighbors in the overlay. Nodes are expected to exchange this information by using some form of link-state routing protocol implemented by ORP.

Furthermore, we assume that the QoS layer cannot interfere with the general resource usage (either in terms of host or network resources) other than those used by the QoS layer itself. In other words, the QoS routing protocol cannot perform resource reservation other than on residual resources (*i. e.,* resources not used by other applications running simultaneously on the node). Obviously, if *all* applications on a node run on top of the QoS layer, then true resource reservation can be performed for the host resources. Network resources will however always be fluctuating due to traffic streams outside the control of the QoS layer.

Some resource fluctuations may drive the node into resource starvation. During resource starvation the node is unable to honor some or all of the QoS guarantees. This type of events may lead to degradation in the quality of rendered media (*e. g.,* MPEG frames that are lost, garbled, or arrive too late).

Applications on top of the QoS layer may be able to tolerate quality degradation for very brief periods of time or even recover from brief degradation by using forward error correction (FEC) codes or retransmissions.

However, prolonged quality degradation may eventually lead to user dissatisfaction with the quality of the service. Each node must therefore carefully monitor the link properties to each of its immediate neighbors. If resource starvation is detected (or anticipated) then a new feasible path should be found and traffic re-routed on it. This mechanism must be robust enough to avoid route flapping.

We are currently designing two protocols that fit into the ORP framework: Route Discovery Protocol (RDP) and Route Management Protocol (RMP) [IP07]. RDP uses a form of selective forwarding in order to discover paths to a destination, subject to multiple QoS constraints. We have currently a prototype implementation that shows promising results [DV07].

RMP is a link-state protocol that complements RDP by addressing churn (*i. e.*, node joining and leaving the overlay). Since RMP is still in the design phase we refrain from going into the details of the protocol.

However, there are also arguments against providing QoS in overlay networks. In particular in [CHM+03] it is argued that no *quantitative* QoS (*i. e.*, other than short-term and within a local region) can be provided by overlay networks unless the underlaying layers of the network stack implement some notion of QoS. We agree in that providing QoS through overlay networks presents a set of tough challenges. However, in the current situation where demand for QoS is growing (*e. g.*, in terms of multimedia services) and service providers find little incentive to provide anything beyond best-effort service the overlay networks are the only viable short-term solution for end-users.

1.4 QoS Routing Algorithms

The QoS routing process deals with two main problems: topology information dissemination and route computation according to specific QoS constraints. The first problem is dealt with by running a routing *protocol* and the second problem requires a routing *algorithm* that acts on the topology data. Only the second problem is within the scope of this report.

Route computation can be described as an optimization problem. The formal definition of optimization problems is given in Chapter 2. Here we only mention the classes of optimization problems relevant for the ORP framework.

IP routing algorithms within an AS are used to compute routes along the shortest path to destinations. From a traffic engineering point of view it is desirable to spread out the traffic flows such as to minimize the cost of routing. The actual cost function can be either some monetary cost or it can be related to link load, average delay or any other metric of interest. Given the capacities of all links in the network, the demands between each source-destination pairs and the cost function, one can calculate the proportion of traffic that should be routed on each link. This is called *optimal routing* [BG91]. Typically, the route computations are carried out at a central location and the results are then distributed to the routers, in form of routing table updates.

This approach, in its current form, cannot be used with ORP for a number of reasons:

i) we expect the topology to be changing rather often due to node dynamics (churn).

ii) since each node can request resources due to own needs it is impractical to attempt to know at all times the demand between each source-destination pair.

iii) we prefer that the route computation is distributed for scalability purposes and in order to avoid a single point of failure.

One way to alleviate these problems is to modify ORP to use hierarchical routing. Nodes that are known to have long uptimes and enough available bandwidth are upgraded to supernodes. This is similar to the concept of ultrapeer election in Gnutella [Ili06]. Regular nodes are connected to one supernode and exchange link-state information (*i. e.*, routing metrics) with it. Supernodes aggregate the link-state information and exchange it with other supernodes. In networks using hierarchical routing node and resource dynamics are less likely to have a global impact since they can be addressed in the domain of the supernode.

However, QoS routing in hierarchical networks requires some algorithm for topology aggregation [Lee95, LNC04, TLUN07]. Topology aggregation in turn leads to inaccurate QoS information [GO97].

The alternative is to have ORP employ a form of selfish routing, where the nodes manage resources according to their needs without considering the benefit of the network as a whole [RT02]. As it turns out, selfish networks are able to perform quite well [PM04, CFSK04] from the point of view of individual nodes. In practice, this means that each node requests a path when it needs one, without considering the global state of the network. The nodes on the path perform admission control and resource reservation according to their own incentives. The problem with this approach is that it requires control mechanisms to prevent greedy nodes from starving well-behaving nodes.

There is also the possibility of a middle ground between optimal routing and selfish routing. ORP nodes can be programmed to establish several paths across the overlay. On each path, bandwidth is reserved in excess of the node's needs. The excess bandwidth is used to serve requests from other nodes that want to route their packets through the current node. The result is that each overlay node manages a subset of the overlay (in terms of nodes and bandwidth) and is responsible for the admission control and resource allocation (*i. e.*, reserved bandwidth) within the overlay subset. Theoretically, each node performs optimal routing in its subset.

To this end we see that the routing algorithm on each node acts either on a path at a time or on several paths simultaneously. This distinction becomes important in the remainder of this report since some of the optimization algorithms considered can only be used in one or the other of these scenarios.

Chapter 2

Theoretical Foundation

In this chapter we review the theoretical foundation required to investigate the route selection algorithms presented in the subsequent chapters.

The general optimization problem has the form [Lue04, BV04]:

$$\begin{array}{ll} \text{minimize} & f(\boldsymbol{x}) \\ \text{subject to} & \boldsymbol{g}(\boldsymbol{x}) \leq \boldsymbol{b} \\ & \boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{c} \\ & \boldsymbol{x} \in \Omega \end{array}$$

The real-valued function f, called *objective function*, denotes some function for which we need to find the minimum value over all $\boldsymbol{x}$ in the set Ω. In minimization problems the objective function is also called *cost* function. The n-dimensional vector $\boldsymbol{x} = (x_1, \ldots, x_n)$ is called the *optimization variable*. The optimization variable can take on values from the set Ω, which is a subset of n-dimensional space.

In *constrained* optimization, the optimization variable is limited to a subset of values in Ω by introducing *inequality constraint functions* $\boldsymbol{g}(\boldsymbol{x}) = (g_1(\boldsymbol{x}), \ldots, g_m(\boldsymbol{x}))$ and *equality constraint functions* $\boldsymbol{h}(\boldsymbol{x}) = (h_1(\boldsymbol{x}, \ldots, h_r(\boldsymbol{x}))$ where $m \leq n$ and $r \leq n$. The constants $\boldsymbol{b} = (b_1, \ldots, b_m)$ and $\boldsymbol{c} = (c_1, \ldots, c_r)$ are called bounds or limits for the constraint functions. In general, the word "constraints" is used to denote the constraint functions and bounds together.

For example, in a network dimensioning problem $\boldsymbol{g}(\boldsymbol{x})$ can be used to specify capacity constraints and $\boldsymbol{h}(\boldsymbol{x})$ to specify demand constraints [PM04]. In the specific case considered there, the capacity constraints denote the maximum capacity of the links and the demand constraints are the traffic volumes that need to be transfered between end-points. The objective function $f(\boldsymbol{x})$ is thus the sum of all reserved capacities across the network and the goal is to minimize this sum.

The constraint functions and the set Ω define the *feasible set* (also called *constraint set*), which is the set of $\boldsymbol{x}$-values considered in the optimization problem. Sometimes it is also called the *feasible space* or the *feasible region*. A x-value being part of the feasible set is a called a *feasible point*. If the feasible set contains at least one feasible point the problem is *feasible* or else it is said to be *unfeasible*.

The output of the $f(\boldsymbol{x})$ at a feasible point is called *feasible value*. The $\boldsymbol{x}$-value where $f(\boldsymbol{x})$ attains its minimum value is called the *optimal point* or *optimal solution* and is denoted by $\boldsymbol{x}^*$. The value of $f(\boldsymbol{x})$ at the point $\boldsymbol{x} = \boldsymbol{x}^*$ is called *optimal value* and is denoted by $f^*(\boldsymbol{x})$.

In the case of *unconstrained* optimization there are no constraint functions and the feasible set consists of all elements of the set Ω. It is also possible to have a problem with constraint functions but without an explicit objective function. Such a problem has the form [BV04]

$$\begin{array}{ll} \text{find} & \boldsymbol{x} \\ \text{subject to} & \boldsymbol{g}(\boldsymbol{x}) \leq \boldsymbol{b} \\ & \boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{c} \\ & \boldsymbol{x} \in \Omega \end{array}$$

and is called a *feasibility problem*. The goal is to find $\boldsymbol{x}$-values that satisfy the constraints.

For example, in case of routing, the feasibility problem may be used to describe a constrained path selection problem. We assume in this example that the variable $\boldsymbol{x}$ denotes a sequence of links $x_1, \ldots, x_n$ building up the path between a source node and a destination node. Each link x_i is associated with a delay d_i and a delay jitter j_i. We use the shorthand notation $\boldsymbol{d} \leftrightarrow \boldsymbol{x}$ and $\boldsymbol{j} \leftrightarrow \boldsymbol{x}$ to denote delay and delay jitter associated with a path. There could be several paths between the two nodes. Therefore we use the symbol $\mathcal{P}$ to denote the union of the paths $\boldsymbol{x}$ between source and destination. We use the inequality constraint functions $g_1(\boldsymbol{x})$ and $g_2(\boldsymbol{x})$ to constrain the path selection to those with a delay less than $b_1 = 100\,\text{ms}$ and delay jitter less than $b_2 = 10\,\text{ms}$. We are not using any equality constraint function in this case. To express the path selection problem as a feasibility problem we write

$$\begin{array}{ll} \text{find} & \boldsymbol{x} \\ \text{subject to} & \sum_{i=1}^{n} d_i \leq 100 \text{ for } \boldsymbol{d} \leftrightarrow \boldsymbol{x} \\ & \sum_{i=1}^{n} j_i \leq 10 \text{ for } \boldsymbol{j} \leftrightarrow \boldsymbol{x} \\ & \boldsymbol{x} \in \mathcal{P}. \end{array}$$

So far we have assumed that the objective function is single-valued. In a similar way, it is possible to have a vector-valued objective function $\boldsymbol{f}(\boldsymbol{x}) = (f_1(\boldsymbol{x}), \ldots f_n(\boldsymbol{x}))$, where $\boldsymbol{x} = (x_1, \ldots, x_m)$. We have in this case what is a called a *multi-objective* optimization problem. This type of problems is outside the scope of the current report. Multi-objective optimization in computer networks is amply discussed in [DF07].

Similarly, the optimization variable can be either vector-valued $\boldsymbol{x}$ (which is the typical case for us) but can be single-valued as well. Furthermore, the components of $\boldsymbol{x}$, $(x_1, x_2, \ldots, x_n)$, may take on continuous values, can be restricted to integer values (so called *integer problems*) or a mixture where some x_i components take on continuous values and the others are restricted to integer values (so called *mixed-integer problems*). Solutions to integer and mixed-integer problems are generally more computationally demanding than solutions to problems with continuous variables. Our discussion focuses on problems with continuous variables and when necessary we explicitly mention what elements of the discussion apply or do not apply to integer or mixed-integer problems.

2.1 Mathematical Background

The aim in this section is to provide a minimal mathematical background that is required for discussing optimization theory and related algorithms. Due to space reasons we do not provide any proofs, but rather point to relevant references. The main interest lies in establishing definitions and notation.

2.1.1 Sets

Much of the modern mathematical theory is based on the concept of *set*, which is defined as a collection of elements. We denote a set by a capital letter and specify its elements inside curly brackets, with elements separated by comma (*e. g.* , $A = \{0, 1, 2, 3\}$). We can also write $B = \{x : 0 \leq x < 4\}$ to specify a range. The expression for B should be read as "the set B contains all elements x such that x is greater or equal to 0 and less than 4". Note that in the example given here A and B have the exact same elements only if the set elements are known to be integers. To show that a specific element x belongs to a set we write for example $x \in A$. If that is not the case we write $x \notin A$. The set containing no elements is called the *empty set* and is denoted by $\emptyset$.

The basic operation applicable on sets are *union* $A \cup B$, *intersection* $A \cap B$ and *complementation* A^C. We assume they are familiar to the reader and refrain from additional explanations.

Some of the important sets used here are the set of natural numbers $\mathbb{N}$, the set of all integers $\mathbb{Z}$ and the set of real numbers $\mathbb{R}$.

Given two sets A and B if every element of the set A is also an element of the set B we call A a subset of B and write $A \subseteq B$. If the set B has additional elements that do not exist in A, then A is a called a *proper subset* of B and we write $A \subset B$. If $A \subseteq B$ and also $B \subseteq A$ this relation is denoted by $A = B$. This is also true for proper subsets.

It turns out that it is very useful if the elements of a set can be ordered. Such a set is called an *ordered* set.

Definition 2.1. An order on a set A, denoted by $<$, satisfies the following conditions [Rud76]. Given three elements $x, y, z \in A$, then

i) either $x < y$, or $x > y$, or $x = y$ (*i. e.* , only one statement can be true)

ii) $x < y$ and $y < z$ implies $x < z$

Ordered sets can have bounds.

Definition 2.2 (Bounds)**.** Given an ordered set A and and a set $B \subset A$, the set B is said to be *bounded (from) above* if there exists an element $y \in A$ such that $x \leq y$ for all $x \in B$. We call y an *upper bound* of B. An upper bound y_0 of the set B is called the *least upper bound* or *supremum* if y_0 is less than or equal to any other upper bound of B. Lower bounds are defined in the same way [Rud76, Shi96]. The *greatest lower bound* is also called *infimum*.

Definition 2.3 (Fields)**.** A *(number) field* is a set F with addition and multiplication operations, which satisfies the following field axioms [Rud76, Shi96]: .

i) $x + y \in F$ if and only if $x, y \in F$

ii) $x + y = y + x$ for all $x, y \in F$

iii) $(x + y) + z = x + (y + z)$ for all $x, y \in F$

iv) $0 + x = x$ for all x and 0 inF

v) $x + (-x) = 0$ for all $x \in F$

vi) $xy \in F$ if and only if $x, y \in F$

vii) $xy = yx$ for all $x, y \in F$

viii) $(xy)z = x(yz)$ for all $x, y \in F$

ix) $1x = x$ for all $x \in F$ and $1 \neq 0$

x) $x\,(1/x) = 1$ such that $x, 1/x \in F$ and $x \neq 0$

xi) $x(y + z) = xy + xz$ for all $x, y, z \in F$.

An *ordered field* is an ordered set F that in addition to the field axioms satisfies [Rud76]:

i) assuming $x, y, z \in F$ and $y < z$ then $x + y < x + z$

ii) assuming $x, y \in F$ and $x > 0, y > 0$ then $xy > 0$.

It can be shown that given the concept of an ordered field and additional properties of numbers, one arrives at the set of of all real numbers, the real field $\mathbb{R}$ [Rud76]. It is common practice to add the symbols $+\infty$ and $-\infty$ to $\mathbb{R}$ such that $-\infty < x < +\infty$ for all $x \in \mathbb{R}$. This way the original order in $\mathbb{R}$ is preserved. Henceforth we assume that $\mathbb{R}$ contains the infinity symbols, unless stated otherwise.

2.1.2 Spaces

The field $\mathbb{R}$ allows us to generalize the concept of a set to a metric space. The main feature of a metric space is the notion of distance between elements.

Definition 2.4 (Metric space)**.** Given a set X and a *distance* metric $\rho(x, y) \in \mathbb{R}$ defined for all $x, y \in X$, the pair $\langle X, \rho \rangle$ denotes a metric space with the following properties [KF75, Rud76]:

i) $\rho(x, y) \geq 0$

ii) $\rho(x, y) = 0$ if and only if $x = y$

iii) $\rho(x, y) = \rho(y, x)$

iv) $\rho(x, z) \leq \rho(x, y) + \rho(y, z)$.

Definition 2.5 ($\mathbb{R}^1$)**.** The set of all real numbers equiped with the distance metric $\rho(x, y) = |x - y|$ is a metric space denoted by $\mathbb{R}^1$.

It is necessary to expand the definition of a metric space to several dimensions. In order to do so we define the k-dimensional vector $\boldsymbol{x}$ to be the ordered set $\{x_1, \ldots, x_k\}$, where $x_i \in \mathbb{R}$ for $i = 1, \ldots, k$. We adopt the standard notation with parenthesis for vectors and write $\boldsymbol{x} = (x_1, \ldots, x_k)$. Also, given two k-dimensional vectors $\boldsymbol{x}$, $\boldsymbol{y}$ and a number $a \in \mathbb{R}$ we define the following vector operations [Rud76]:

i) $a\boldsymbol{x} = (ax_1, \ldots, ax_2)$ (scalar multiplication)

ii) $\boldsymbol{x} + \boldsymbol{y} = (x_1 + y_1, \ldots, x_k + y_k)$ (vector addition)

iii) $\boldsymbol{x} \cdot \boldsymbol{y} = (x_1y_1 + x_2y_2 +, \ldots, +x_{k-1}y_{k-1} + x_ky_k)$ (inner product)

iv) $\|\boldsymbol{x}\| = \sqrt{\boldsymbol{x} \cdot \boldsymbol{x}} = \sqrt{x_1^2 +, \ldots, +x_k^2}$ (Euclidean norm)

Operation i) and ii) enable vectors to satisfy the field axioms. For a positive integer k, the set of all k-dimensional vectors satisfying the field axioms together with a suitable distance metric, ρ, define a *vector space* $\mathbb{R}^k$ over the real field. The zero vector in a vector space is defined as $\mathbf{0} = (0, \ldots, 0)$. A vector in $\mathbb{R}^k$ is also called a *point* and its x_i elements, for $i = 1, \ldots, k$, can be thought of as coordinates in the $\mathbb{R}^k$ space.

The inner product associates a number in $\mathbb{R}$ to every pair of vectors in $\mathbb{R}^k$. The Euclidean norm uses the inner product to implement a notion of length or size [Edw94]. The Euclidean norm belongs to the class of p-norms

$$\|\boldsymbol{x}\|_p = \left(\sum_{i=1}^{n} |x_i| \right)^{\frac{1}{p}}, \tag{2.1}$$

where we have assumed that the vector $\boldsymbol{x}$ has n elements [GVL96]. Another important member of this class is obtained when $p \to \infty$

$$\|\boldsymbol{x}\|_\infty = \max_{1 \leq i \leq n} |x_i|. \tag{2.2}$$

We are mainly interested in the Euclidean norm, but the ∞-norm is useful for SAMCRA (Chapter 8).

Definition 2.6 (Euclidean $\mathbb{R}^k$ space)**.** The *Euclidean* $\mathbb{R}^k$ *space* is the set endowed with distance metric $\rho = \|\boldsymbol{x} - \boldsymbol{y}\|$ consisting of all k-dimensional vectors, with k being an integer greater than zero, that satisfy the field axioms. We denote the Euclidean $\mathbb{R}^k$ space by $\mathbb{E}^k$. $\mathbb{E}^k$ is a *linear* space [KF75].

In the reminder of this report when we mention that certain definitions or properties apply to the space $\mathbb{R}^n$ we imply that they also apply to the space $\mathbb{E}^n$, unless stated otherwise. On the other hand, if we mention the space $\mathbb{E}^n$, then we do not imply the space $\mathbb{R}^n$, unless explicitly stated otherwise. Clearly, the main difference between the two spaces is related to the Euclidean norm, which in some cases may prevent more general definitions.

It is sometimes useful to work with a subset of a space called *subspace*.

Definition 2.7 (Subspace)**.** A set A of elements of a space R^n is called a subspace if and only if

i) for all $\boldsymbol{x}, \boldsymbol{y} \in A$, $\boldsymbol{x} + \boldsymbol{y} \in A$

ii) for all $c \in \mathbb{R}$ and all $\boldsymbol{x} \in A$, $c\boldsymbol{x} \in A$.

In other words the set A is a subspace of $\mathbb{R}^n$ if it is closed under the operations of vector addition and scalar multiplication.

2.1.3 Topology

The field of topology is concerned with the study of mathematical structures [Men90]. Its focus is on *topological spaces*. Topological spaces are described mainly in terms of *open* sets, a concept that is to be defined shortly. Our interest in topology is restricted to certain definitions and results that appear in the context of optimization theory only.

Definition 2.8 (Open and closed intervals)**.** We define an *open interval* (a, b) to be the set of numbers $x \in \mathbb{R}$ such that $a < x < b$. Open intervals are also called *segments* [Rud76]. If the set of numbers $x \in \mathbb{R}$ satisfies $a \leq x \leq b$ we call it a *closed interval* $[a, b]$. Half-open intervals, such as $(a, b]$ and $[a, b)$, exist as well.

Definition 2.9 (Open and closed balls)**.** Assume a metric space $\langle X, \rho \rangle$, a point $a \in X$ and any radius $\delta \in \mathbb{R}$ such that $\delta > 0$. An open ball of size δ is a set consisting of all elements $x \in X$ that satisfy $\rho(a, x) < \delta$ [Men90]. A *closed ball* of size δ is defined in similar way with the exception that it must satisfy $\rho(a, x) \leq \delta$ instead. If we allow X and a to be vectors, this definition extends to n-dimensional metric spaces.

Definition 2.10 (Neighborhood)**.** In a metric space $\langle X, \rho \rangle$, the set defined by the open ball of radius δ center at a is a called the δ-neighborhood of a. If the radius is obvious from the context, then the δ symbol is typically omitted [Men90].

The concept of a neighborhood is useful in describing properties associated with points in a set.

Definition 2.11. Assume a metric space $\langle X, \rho \rangle$ with a subset A. A point a in this space is called a [Rud76]

limit point of A if every neighborhood of a contains a point $b \neq a$ such that $b \in A$.

isolated point of A if $a \in A$ and a is not a limit point of A.

interior point of A if there exists a neighborhood of a, which is a subset of A.

The classification of points in a set helps us in defining different types of sets.

Definition 2.12. Assume a metric space $\langle X, \rho \rangle$ with a subset A. The set A is said to be [Rud76]

closed if every limit point of A is as point of A.

open if every point of A is an interior point of A.

perfect if A is closed and if every point of A is a limit point of A.

bounded if there is a point $b \in X$ such that $\rho(a, b) < R$ for all points $a \in A$ and $R > 0$.

dense in X if every point of X is a point of A, a limit point of A or both.

convex if $A \subset \mathbb{R}^n$ and every combination $\lambda \boldsymbol{x} + (1 - \lambda)\boldsymbol{y}$ yields a point also in A, for all $\boldsymbol{x}, \boldsymbol{y} \in A$ and $0 < \lambda < 1$.

compact if it is both closed and bounded [Lue04].

The *interior* of a set A is the set consisting of all interior points of A. The *closure* of a set A is the smallest set that can contain A (*i. e.*, the closed ball containing the points of the set A). The *boundary* of a set A is the set containing the points of A's closure, which are are not interior points.

2.1.4 Linear Algebra

Since we work with linear spaces we often perform linear combinations of a several vectors. A *linear combination* of n vectors in $\mathbb{R}^k$ has the form $c_1\boldsymbol{x}_1 + c_2\boldsymbol{x}_2 + \cdots + c_m\boldsymbol{x}_m + c_n\boldsymbol{x}_n$, where $c_1, \ldots, c_n \in \mathbb{R}$ are called the *coefficients* of the linear combination [Shi77, Edw94].

Definition 2.13 (Linear independence)**.** The vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ are called *linearly independent* if the the following expression based on their linear combination

$$c_1\boldsymbol{x}_1 + c_2\boldsymbol{x}_2 + \cdots + c_m\boldsymbol{x}_m + c_n\boldsymbol{x}_n = 0 \tag{2.3}$$

holds only if $c_1 = ... = c_n = 0$. Otherwise, if the expression holds for one or more nonzero coefficients, the vectors are said to be *linearly dependent*. Linear dependence implies that one of the vectors can be written as a linear combination of the others [Shi77].

The dimension of a space is given by the maximum number of independent vectors that can exist in that space. For example, in a $\mathbb{R}^n$ space there can be n independent vectors, whereas $n + 1$ or more vectors are linearly dependent. Thus the space $\mathbb{R}^n$ has dimension n (which motivates the notation).

Definition 2.14 (Basis)**.** Given n linearly independent vectors $\boldsymbol{x}_i$ in a space $\mathbb{R}^n$, if any vector $\boldsymbol{y} \in \mathbb{R}^n$ can be uniquely determined through a linear combination

$$\boldsymbol{y} = c_1\boldsymbol{x}_1 + c_2\boldsymbol{x}_2 + \cdots + c_m\boldsymbol{x}_m + c_n\boldsymbol{x}_n \tag{2.4}$$

then the vectors x_i are said to be a basis of the space $\mathbb{R}^n$. In this case the coefficients $c_1, \ldots, c_n$ are said to be the *components* or *coordinates* of the vector $\boldsymbol{y}$ with respect to the basis $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ and we can write $\boldsymbol{y} = (c_1, \ldots, c_n)$.

In defining the space $\mathbb{E}^n$ we have used vectors of the form $\boldsymbol{x} = (x_1, \ldots, x_n)$ without mentioning any basis. In a Euclidean space it is convenient to define vectors with respect to a *orthonormal* basis. Such basis rests on the definition of an angle between vectors and that of a normalized vector.

Definition 2.15. The angle θ between two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ in a space $\mathbb{R}^k$ with a Euclidean norm is defined as

$$\cos\theta = \frac{\boldsymbol{x} \cdot \boldsymbol{y}}{\|\boldsymbol{x}\| \, \|\boldsymbol{y}\|} \tag{2.5}$$

The *Cauchy-Schwarz inequality*, presented below, ensures that $0 \leq \cos\theta \leq 1$ or equivalently $0 \leq \theta \leq 180$ degrees.

Definition 2.16 (Cauchy-Schwarz inequality)**.** All vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ defined on a space $\mathbb{R}^k$ with Euclidean norm satisfy the Cauchy-Schwarz inequality [Rud76, Shi77]

$$\|\boldsymbol{x} \cdot \boldsymbol{y}\| \leq \|\boldsymbol{x}\| \, \|\boldsymbol{y}\| \, . \tag{2.6}$$

Definition 2.17 (Triangle inequality)**.** It follows from the Cauchy-Schwarz inequality that all vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ defined on a space $\mathbb{R}^k$ with Euclidean norm satisfy also the *triangle inequality* [Rud76]

$$\|\boldsymbol{x} + \boldsymbol{y}\| \leq \|\boldsymbol{x}\| + \|\boldsymbol{x}\| \, . \tag{2.7}$$

Definition 2.18 (Normalized vector)**.** A vector $\boldsymbol{x}$ can be *normalized* through division by its norm:

$$\boldsymbol{e} = \frac{\boldsymbol{x}}{\|\boldsymbol{x}\|} \tag{2.8}$$

A normalized vector is a *unit vector* (*i. e.*, $\|\boldsymbol{e}\| = 1$) [Shi77].

Two vectors are said to be *orthogonal* if their inner product is zero. By Definition 2.15 orthogonal vectors in a Euclidean space are separated by an angle of 90 degrees (hence the name orthogonal). In addition to that, nonzero orthogonal vectors are independent.

Definition 2.19 (Orthonormal basis)**.** If the n vectors used for the basis of a space $\mathbb{R}^n$ are both normalized and mutually orthogonal, then the space is said to have an *orthonormal basis*.

For the space $\mathbb{R}^n$ the orthonormal basis typically employed consists of the vectors

$$\begin{aligned}
\boldsymbol{e}_1 &= (1, 0, \ldots, 0, 0) \\
\boldsymbol{e}_2 &= (0, 1, \ldots, 0, 0) \\
&\vdots \\
\boldsymbol{e}_{n-1} &= (0, 0, \ldots, 1, 0) \\
\boldsymbol{e}_n &= (0, 0, \ldots, 0, 1).
\end{aligned}$$

This is called the *canonical basis* or *standard basis* and has the following property that simplifies calculations:

$$\boldsymbol{e}_i \cdot \boldsymbol{e}_j = \begin{cases} 1 & \text{if } i = j, \\ 0 & \text{if } i \neq j. \end{cases} \tag{2.9}$$

Since we can write any vector in R^n as a linear combination of the basis vectors, the inner product of the vectors $\boldsymbol{x} = x_1\boldsymbol{e}_1 + \cdots + x_n\boldsymbol{e}_n$ and $\boldsymbol{y} = y_1\boldsymbol{e}_1 + \cdots + y_n\boldsymbol{e}_n$ is equal to $x_1y_1 + \cdots + x_ny_n$, which motivates the definition of inner product $\boldsymbol{x} \cdot \boldsymbol{y}$ used in Section 2.1.2.

Definition 2.20 (Linear operator)**.** Given a space $\mathbb{R}^m$ and a space $\mathbb{R}^n$, a *linear operator* $\mathcal{A}$ is a rule (mapping) that assigns a vector in $\mathbb{R}^m$ to each vector in $\mathbb{R}^n$ such that the following conditions are satisfied:

i) $\boldsymbol{v} = \mathcal{A}(\boldsymbol{x} + \boldsymbol{y}) = \mathcal{A}\boldsymbol{x} + \mathcal{A}\boldsymbol{y}$ for all $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$ and so that $\boldsymbol{v} \in \mathbb{R}^m$,

ii) $\boldsymbol{v} = \mathcal{A}(c\boldsymbol{x}) = c\mathcal{A}\boldsymbol{x}$ for every $c \in \mathbb{R}$ and every $\boldsymbol{x} \in \mathbb{R}^n$ so that $\boldsymbol{v} \in \mathbb{R}^m$.

Note that it is allowed that $m = n$, in which case the spaces $\mathbb{R}^m$ and $\mathbb{R}^n$ coincide [Shi77].

Assume that the vectors $\boldsymbol{e}_1, \ldots, \boldsymbol{e}_n$ constitute the canonical basis of the space $\mathbb{R}^n$, the vectors $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_m$ are the canonical basis for the space $\mathbb{R}^m$ and that $\mathcal{A}$ is the linear operator that provides a mapping between $\mathbb{R}^n$ and $\mathbb{R}^m$. We are interested in the form of the mapping of the canonical basis of $\mathbb{R}^n$ to the space $\mathbb{R}^m$. We focus our attention on $\boldsymbol{e}_1$ first. In the space $\mathbb{R}^m$, the vector $\boldsymbol{e}_1$ is described as a linear combination of the vectors $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_m$, which are the space basis. This linear combination is provided by the linear operator $\boldsymbol{A}$

$$\mathcal{A}\boldsymbol{e_1} = a_{11}\boldsymbol{u}_1 + a_{21}\boldsymbol{u}_2 + \cdots + a_{m1}\boldsymbol{u}_m = (a_{11}, a_{21}, \ldots, a_{m1}) \tag{2.10}$$

where a_{i1} are the coordinates in the space $\mathbb{R}^m$ associated with the unit vector $\boldsymbol{e}_1$ [Edw94, Shi77]. Similarly,

$$\begin{aligned} \mathcal{A}\boldsymbol{e_2} &= a_{12}\boldsymbol{u}_1 + a_{22}\boldsymbol{u}_2 + \cdots + a_{m2}\boldsymbol{u}_m = (a_{12}, a_{22}, \ldots, a_{m2}) \\ &\vdots \\ \mathcal{A}\boldsymbol{e_n} &= a_{1n}\boldsymbol{u}_1 + a_{2n}\boldsymbol{u}_2 + \cdots + a_{mn}\boldsymbol{u}_m = (a_{1n}, a_{2n}, \ldots, a_{mn}) \end{aligned} \tag{2.11}$$

for the the remaining vectors $\boldsymbol{e}_1, \ldots, \boldsymbol{e}_n$. This motivates the use of *matrix* notation. We define the $m \times n$ matrix $\boldsymbol{A}$ as the rectangle block with m rows and n columns

$$\boldsymbol{A} = \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \ldots & a_{nm} \end{bmatrix} \tag{2.12}$$

where the columns contain the coefficient for the linear combinations $\mathcal{A}\boldsymbol{e}_1 \ldots \mathcal{A}\boldsymbol{e_n}$. We use capital bold letter for matrices. In compact notation we write $\boldsymbol{A} = [a_{ij}]$ when the size of the matrix is implied from the context. The first index, i in $[a_{ij}]$, denotes the i-th row of the matrix, while the second index, j, denotes the j-th column of the matrix A. Following convention for matrix notation, vectors are by default $n \times 1$ matrices (so called column vectors). We use the superscript ^{T} to denote the vector transpose operation (which turns a column vector into a row vector and vice versa). For example $\boldsymbol{e_1}$ appears as

$$\boldsymbol{e} = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & \ldots 0 \end{bmatrix}^T. \tag{2.13}$$

We define also shorthand notation for selecting a specific column or row from a matrix. For example,

$$\boldsymbol{A}_2 = \begin{bmatrix} a_{21} & a_{22} & \ldots & a_{2n} \end{bmatrix} \quad \text{and} \quad \boldsymbol{A}^2 = \begin{bmatrix} a_{12} \\ a_{22} \\ \vdots \\ a_{m2} \end{bmatrix} \tag{2.14}$$

are the second row and second column respectively of the matrix $\boldsymbol{A}$.

Definition 2.21 (Matrix operations)**.** We define the following operations for the matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ and the scalar c [Edw94, Shi77]:

i) $\boldsymbol{A} + \boldsymbol{B} = \begin{bmatrix} a_{ij} + b_{ij} \end{bmatrix}$ if $\boldsymbol{A}$ and $\boldsymbol{B}$ have the same number of rows and columns (matrix addition)

ii) $c\boldsymbol{A} = \begin{bmatrix} ca_{ij} \end{bmatrix}$ (scalar multiplication)

iii) $\boldsymbol{AB} = \boldsymbol{C} = \begin{bmatrix} c_{ij} \end{bmatrix} = \sum_{k=1}^{n} a_{ik}b_{kj}$ if the number of columns in $\boldsymbol{A}$ is equal to the number of rows in $\boldsymbol{B}$ (matrix multiplication)

iv) $\boldsymbol{A}^T = \begin{bmatrix} a_{ij} \end{bmatrix}^T = \begin{bmatrix} a_{ji} \end{bmatrix}$ (matrix transpose)

The matrix operations i)–iii) transform the set of all $m \times n$ matrices into a vector space. The transpose operation transforms columns into rows and vice versa.

Matrix multiplication can be applied only if the number of columns in $\boldsymbol{A}$ equals the number of rows in $\boldsymbol{B}$. When $\boldsymbol{A}$ is a $1 \times m$ matrix (row vector) and $\boldsymbol{B}$ is a $m \times 1$ matrix (column vector) the operation $\boldsymbol{AB}$ yields the inner product of the two vectors. This means that for two vectors $\boldsymbol{x} = [x_1 \dots x_n]$ and $\boldsymbol{y} = [y_1 \dots y_n]$

$$\boldsymbol{x} \cdot \boldsymbol{y} = \boldsymbol{x}^T \boldsymbol{y} = \boldsymbol{y}^T \boldsymbol{x} \tag{2.15}$$

The matrix operations are endowed with the following properties:

i) $(\boldsymbol{AB})\boldsymbol{C} = \boldsymbol{A}(\boldsymbol{BC})$

ii) $(\boldsymbol{A} + \boldsymbol{B})\boldsymbol{C} = \boldsymbol{AC} + \boldsymbol{BC}$

iii) $\boldsymbol{A}(\boldsymbol{B} + \boldsymbol{C} = \boldsymbol{AC} + \boldsymbol{BC}$

iv) $c\boldsymbol{A} = \boldsymbol{A}c$

v) $c(\boldsymbol{AB}) = (c\boldsymbol{A})\boldsymbol{B} = \boldsymbol{A}(c\boldsymbol{B})$

It is important to note that in general the matrix multiplication is not commutative (*i. e.*, $\boldsymbol{AB} \neq \boldsymbol{BA}$.)

Definition 2.22 (Rank)**.** A $m \times n$ matrix $\boldsymbol{A}$ with a maximum number c of linearly independent columns is said to be of *rank* c. In that case we write $\operatorname{rank}(\boldsymbol{A}) = c$. In a matrix of rank c the maximum number of linearly independent rows is also equal to c.

If $\operatorname{rank}(\boldsymbol{A}) = \min(m, n)$ then matrix $\boldsymbol{A}$ is said to be of *full rank*.

Definition 2.23 (Zero matrix $\mathbf{0}$)**.** For convenience, we use the symbol $\mathbf{0}$ to denote the *zero matrix*, which is the $m \times n$ matrix with all elements equal to zero.

Returning to the linear mapping in Eq. 2.12 it can be shown [Edw94, Shi77] that any vector $\boldsymbol{x}$ in the space $\mathbb{R}^n$ can be mapped to a vector $\boldsymbol{y}$ in the space $\mathbb{R}^m$ by multiplication with the matrix $\boldsymbol{A}$

$$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}, \tag{2.16}$$

which can be verified by letting $\boldsymbol{x} = \boldsymbol{e_1} = \begin{bmatrix} 1 & 0 & \dots & 0 \end{bmatrix}^T$.

We turn the attention now towards *square* matrices with n rows and n columns. A square matrix A is said to be *symmetric* if $\boldsymbol{A} = \boldsymbol{A}^T$.

Definition 2.24 (Unit matrix $\boldsymbol{I}$)**.** We define the *unit matrix* $\boldsymbol{I}$

$$\boldsymbol{I} = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ \vdots & 0 & \ddots & \dots & 0 \\ 0 & \dots & 0 & 1 & 0 \\ 0 & 0 & \dots & 0 & 1 \end{bmatrix} \tag{2.17}$$

as the $n \times n$ matrix with elements equal to 1 along the diagonal and elements equal to 0 otherwise, such that $\boldsymbol{AI} = \boldsymbol{IA} = \boldsymbol{A}$. The matrix $\boldsymbol{I}$ is clearly symmetric.

If $\boldsymbol{AB} = \boldsymbol{I}$ then we call $\boldsymbol{B}$ the inverse of $\boldsymbol{A}$ and denote it by $\boldsymbol{A}^{-1}$ (alternatively it can be said that $\boldsymbol{A}$ is the inverse of $\boldsymbol{B}$, in which case $\boldsymbol{A} = \boldsymbol{B}^{-1}$.

Determinants are a useful tool to calculate, among other things, the inverse of a square matrix $\boldsymbol{A}$.

Definition 2.25. The determinant of a $n \times n$ matrix $\boldsymbol{A}$, denoted $\det \boldsymbol{A}$, is a real number associated with the matrix $\boldsymbol{A}$. The determinant of the $n \times n$ matrix $\boldsymbol{A}$ can be computed recursively by *Laplace expansion* applied to rows

$$\det \boldsymbol{A} = \sum_{j=1}^{n} (-1)^{i+j} a_{ij} \det \boldsymbol{A}_{ij}, \tag{2.18}$$

or to columns

$$\det \boldsymbol{A} = \sum_{i=1}^{n} (-1)^{i+j} a_{ij} \det \boldsymbol{A}_{ij}, \tag{2.19}$$

where $\boldsymbol{A}_{ij}$ denotes the $(n-1) \times (n-1)$ matrix obtained by removing the i-th row and the j-th column of the matrix $\boldsymbol{A}$ [Edw94]. If $\det \boldsymbol{A} = 0$, the matrix $\boldsymbol{A}$ is called *singular* and its inverse $\boldsymbol{A}^{-1}$ does not exist or is not uniquely determined. Otherwise, if $\det \boldsymbol{A} \neq 0$, matrix $\boldsymbol{A}$ is said to be *nonsingular* or *invertible* [BT97].

The term $\det \boldsymbol{A}_{ij}$ in the Laplace expansion in Eq. 2.18 and Eq 2.19 is called the (i,j)-th *minor* of the matrix $\boldsymbol{A}$. The number $c_{ij} = (-1)^{i+j} \det \boldsymbol{A}_{ij}$ (*i. e.* , the sign corrected minor) is called (i,j)-th cofactor of $\boldsymbol{A}$. The *adjugate matrix* of $\boldsymbol{A}$ is $\operatorname{adj} \boldsymbol{A} = \left[c_{ij}\right]^T$ [KW98].

Using the determinant and adjugate, the inverse of a square matrix $\boldsymbol{A}$ is

$$\boldsymbol{A}^{-1} = \frac{\operatorname{adj} \boldsymbol{A}}{\det \boldsymbol{A}} \tag{2.20}$$

provided that $\det \boldsymbol{A} \neq 0$.

A nonsingular $n \times n$ matrix $\boldsymbol{A}$ enjoys the following equivalent properties [BT97, KW98]:

i) $\boldsymbol{A}$ has an inverse $\boldsymbol{A}^{-1}$.

ii) $\det \boldsymbol{A} \neq 0$.

iii) the columns of $\boldsymbol{A}$ are linearly independent.

iv) $\boldsymbol{A}^T$ is nonsingular as well.

v) $\boldsymbol{Ax} \neq \boldsymbol{0}$ for all $\boldsymbol{x} \in \mathbb{R}^n$ such that $\boldsymbol{x} \neq \boldsymbol{0}$.

vi) $\boldsymbol{Ax} = \boldsymbol{y}$ has a unique solution $\boldsymbol{x} \neq \boldsymbol{0}$ for all $\boldsymbol{y} \in \mathbb{R}^n$.

Definition 2.26 (Eigenvectors and eigenvalues)**.** Given an equation of the type

$$\boldsymbol{Ax} = \lambda \boldsymbol{x} \tag{2.21}$$

where $\boldsymbol{A}$ is a $n \times n$ linear operator and $\lambda \in \mathbb{R}$, every $\boldsymbol{x} \neq \boldsymbol{0}$ satisfying the equation is called an *eigenvector* and the corresponding λ is called an *eigenvalue* [Shi77, Lue04].

The eigenvalues and the corresponding eigenvectors of the square matrix $\boldsymbol{A}$ can be obtained from the *characteristic equation*

$$\det\left[\boldsymbol{A}\boldsymbol{x} - \lambda\boldsymbol{I}\right] = 0. \tag{2.22}$$

Definition 2.27 (Positive (semi)definite)**.** A symmetric matrix $\boldsymbol{A}$ is called *positive definite* if and only if all eigenvalues of $\boldsymbol{A}$ are positive. If the eigenvalues are nonnegative (*i. e.*, some are positive and the rest are equal to zero) the matrix $\boldsymbol{A}$ is called *positive semidefinite*. An alternative definition can be provided if one considers the *quadratic form* $\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x}$. Then, the matrix $\boldsymbol{A}$ is positive definite if $\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x}$ is positive for all nonzero vectors $\boldsymbol{x}$. Similarly, if the quadratic form is nonnegative for all nonzero vectors $\boldsymbol{x}$, then $\boldsymbol{A}$ is said to be positive semidefinite [Lue04].

The *spectral radius* of a matrix $\boldsymbol{A}$ is defined to be the maximum absolute value of its eigenvalues [GVL96]

$$\rho(A) = \max|\lambda_i|. \tag{2.23}$$

Its value is important in deciding whether iterative algorithms on the matrix A converge to a result or not.

2.1.5 Functions

Definition 2.28 (Function)**.** Given two sets X and Y, a *function* f is a rule (mapping) that assigns a value $f(x) \in Y$ for each element $x \in X$. The sets X and Y may coincide.

The sets X and Y in Definition 2.28 are called the *domain* of the function f and the *range* of the function f, respectively [Rud76]. If $f(X) \subset Y$, then it is said that f maps A *into* B. On the other hand, if $f(X) = Y$, then it is said that f maps X *onto* Y.

We indicate the mapping between f's domain and range using set notation $f : X \to Y$. Generally, we relax the notation and write $y = f(x)$ with the implicit assumption that $x \in X$ and $y, f(x) \in Y$. The value x is called the *(independent) variable* or *argument* of function f, whereas $f(x)$ is called the *dependent variable* of f [Shi96]. Functions are also called *mappings* and *transformations*.

For a subset $A \subset X$, the set of all elements $f(a) \in Y$ such that $a \in A$ is called the *image* of A under f. For a subset $B \subset Y$, the set of all elements $x \in X$ such that $f(x) \in B$ is called the *inverse image* of B under f [Rud76, KF75].

If a function f maps X onto Y and each element $y = f(x) \in Y$ corresponds to a unique element $x \in X$, then f is called a *one-to-one* mapping (function) between X and Y. A one-to-one function is endowed with a *inverse function* denoted by f^{-1} such that $x = f^{-1}(y)$ for all $x \in X$ and $y \in Y$.

The functions within our interest fall into one of the following categories depending on the value space for their domain and space:

i) single-valued (real-valued) functions $f(x)$ of one variable such that $f : \mathbb{R} \to \mathbb{R}$.

ii) single-valued (real-valued) function $f(\boldsymbol{x})$ of a vector such that $f : \mathbb{R}^n \to \mathbb{R}$ and $n > 1$.

iii) vector-valued (multi-valued) function $\boldsymbol{f}(x)$ of one variable such that $f : \mathbb{R} \to \mathbb{R}^n$ and $n > 1$.

iv) vector-valued (multi-valued) function of a vector $\boldsymbol{f}(\boldsymbol{x})$ such that $f : \mathbb{R}^m \to \mathbb{R}^n$ and $m, n > 1$, with possibly $m = n$.

We have already seen examples of functions such as the distance metric in Section 2.1.2 and the linear mappings in Section 2.1.4.

Definition 2.29 (Sequence)**.** A *sequence* is a function f with domain in $\mathbb{N}$ (*i. e.*, $f(n)$ is a sequence if $n \in \mathbb{N}$). It is customary to write f_n instead of $f(n)$ to indicate a sequence. If $f(n) = x_n$, then $x_1, x_2, \ldots$ are called the *terms* of the sequence ($x_1, x_2, \ldots$ and $\{x_n\}_{n=1}^{\infty}$ are other different ways to denote a sequence). If the terms $x_n \in Y$ for all n, then f_n is called a *sequence in* Y [Rud76].

An element x in a metric space $\langle X, \rho \rangle$ is said to be the *limit of a sequence* $f_n = x_1, x_2, \ldots$ if, for every $\epsilon > 0$, there is a number $N \in \mathbb{N}$ such that for all $n \geq N$ we have $\rho(f_n, x) < \epsilon$. If the limit x exists, then the sequence f_n is said to *converge* towards x. Otherwise, the sequence is said to *diverge* [Rud76, Shi96]. The limiting operation described here is usually written

$$\lim_{n \to \infty} f_n = x \tag{2.24}$$

We can make a similar statement for an arbitrary function $\boldsymbol{f}(\boldsymbol{x})$ such that $\boldsymbol{x} \in \mathbb{R}^n$ and $\boldsymbol{f}(\boldsymbol{x}) \in \mathbb{R}^m$ (m and n may be equal). Given a function $\boldsymbol{f}(\boldsymbol{x})$ with domain $D \subset \mathbb{R}^n$ and a limit point $\boldsymbol{a} \in D$ (defined as in Section 2.1.3), the point $\boldsymbol{b} \in R^m$ in the expression

$$\lim_{\boldsymbol{x} \to \boldsymbol{a}} \boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{b} \tag{2.25}$$

is called the limit of $\boldsymbol{f}$ at $\boldsymbol{a}$ if and only if, given any $\epsilon > 0$, there exists a $\delta > 0$ such that $\|\boldsymbol{x} - \boldsymbol{a}\| < \delta$ imply $\|\boldsymbol{f}(x) - \boldsymbol{b}\| < \epsilon$. If $\boldsymbol{b} = \boldsymbol{f}(\boldsymbol{a})$ in Eq. 2.25 then the function $\boldsymbol{f}(\boldsymbol{x})$ is said to be *continuous* at $\boldsymbol{a}$ [Edw94].

Definition 2.30 (Continuity)**.** A function $\boldsymbol{f} : D \to \mathbb{R}^m$ with $D \subset \mathbb{R}^n$ is said to be *continuous* if it is continuous at every point $\boldsymbol{x} \in D$.

We will frequently be interested in the derivative of a continuous function $\boldsymbol{f}(\boldsymbol{x})$. We recall the definition of derivative of a function $f : \mathbb{R} \to \mathbb{R}$:

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \tag{2.26}$$

If $f'(x)$ exists, then the following equation [Rud76] holds:

$$f(x+h) - f(x) = f'(x)h + r(h), \tag{2.27}$$

where $r(h)$ can be interpreted as a small remainder or "error" due to linearization that vanishes as $h \to 0$:

$$\lim_{h \to 0} \frac{r(h)}{h} = 0. \tag{2.28}$$

The product $f'(x)h$ is nothing else but a linear approximation of the the change in f between the points x and $x+h$. If we ignore $r(h)$ (since it vanishes), we can regard $f'(x)$ as the linear operator $\mathcal{A}$ that takes h to $f'(x)h$ as shown below [Rud76, Edw94]:

$$\mathcal{A}h = f'(x)h. \tag{2.29}$$

These ideas can be extended to functions $\boldsymbol{f} : \mathbb{R}^n \to \mathbb{R}^m$ so that we obtain the following equations containing vectors equivalent to Eq. 2.26–2.29:

$$\boldsymbol{f'}(\boldsymbol{x}) = \lim_{\boldsymbol{h}\to\boldsymbol{0}} \frac{\boldsymbol{f}(\boldsymbol{x}+\boldsymbol{h}) - \boldsymbol{f}(\boldsymbol{x})}{\boldsymbol{h}}, \tag{2.30}$$

$$\boldsymbol{f}(\boldsymbol{x}+\boldsymbol{h}) - \boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{f'}(\boldsymbol{x})\,\boldsymbol{h} + \boldsymbol{r}(\boldsymbol{h}), \tag{2.31}$$

$$\lim_{\boldsymbol{h}\to\boldsymbol{0}} \frac{\boldsymbol{r}(\boldsymbol{h})}{\boldsymbol{h}} = \boldsymbol{0}, \tag{2.32}$$

$$\mathcal{A}\boldsymbol{h} = \boldsymbol{f'}(\boldsymbol{x})\,\boldsymbol{h}. \tag{2.33}$$

The derivatives in Eq. 2.26 and Eq. 2.30 are called *total derivatives*. The reason for introducing the linear operators is to enable us to switch over to matrix notation. Before doing that we need to define the meaning of *directional derivatives* and *partial derivatives*.

In the case of directional derivatives we are interested in the rate of change of a function from a point $\boldsymbol{x}$ in the *direction* $\boldsymbol{d}$ of a straight line $\boldsymbol{x} + h\boldsymbol{d}$.

Definition 2.31 (Directional derivative)**.** The directional derivative of a function $f : \mathbb{R}^n \to \mathbb{R}^m$ in the point $\boldsymbol{x}$ with respect to $\boldsymbol{d}$ is [Edw94]

$$D_{\boldsymbol{d}}\boldsymbol{f}(\boldsymbol{x}) = \lim_{h\to 0} \frac{\boldsymbol{f}(\boldsymbol{x}+h\boldsymbol{d}) - \boldsymbol{f}(\boldsymbol{x})}{h} \tag{2.34}$$

The main interest is in the situation where the direction vector $\boldsymbol{d}$ coincides with one of the vectors $\boldsymbol{e}_1, \boldsymbol{e}_2, \ldots, \boldsymbol{e}_n$ of the canonical basis of $\mathbb{R}^n$. In that case we can express the directional derivative in terms of partial derivatives. In what follows, we consider derivatives of functions with the domain $\mathbb{R}^n$ for some positive integer n. However, the derivative definitions apply equally well to functions defined on an open set $E \subset \mathbb{R}^n$.

Definition 2.32 (Partial derivative)**.** A function $f : \mathbb{R}^n \to \mathbb{R}^m$ can be described in terms of the real-valued *component* functions $f_i : \mathbb{R}^n \to \mathbb{R}$, such that $\boldsymbol{f}(\boldsymbol{x}) = (f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots, f_n(\boldsymbol{x}))$. The partial derivative of $f_i(\boldsymbol{x})$ in the point $\boldsymbol{x}$ with respect to the standard basis vector $\boldsymbol{e}_j$ is [Rud76, Edw94]

$$D_{\boldsymbol{e}_j} f_i(\boldsymbol{x}) = \frac{\partial f_i}{\partial x_j}(\boldsymbol{x}) \tag{2.35}$$

Using Definition 2.31 and Definition 2.32 and writing the terms as linear combinations of the vectors making up the canonical basis it can be shown that [Rud76, Edw94]

$$\boldsymbol{f'}(\boldsymbol{x})\,\boldsymbol{e}_j = \frac{\partial f_1}{\partial \boldsymbol{e}_j}(\boldsymbol{x})\boldsymbol{u}_1 + \frac{\partial f_2}{\partial \boldsymbol{e}_j}(\boldsymbol{x})\boldsymbol{u}_2 + \cdots + \frac{\partial f_m}{\partial \boldsymbol{e}_j}(\boldsymbol{x})\boldsymbol{u}_m \tag{2.36}$$

where the vector $\boldsymbol{e}_j$ is the j-th vector of the canonical base of $\mathbb{R}^n$ and the vectors u_i are the components of the canonical basis of $\mathbb{R}^m$. We can observe that $\boldsymbol{f'}(\boldsymbol{x})$ acts as the linear operator $\mathcal{A}$ that maps the unit vector $\boldsymbol{e}_j$ from the space $\mathbb{R}^n$ to the space $\mathbb{R}^m$. These equations are similar to Eq. 2.10 and Eq. 2.11 in Section 2.1.4. Following the development in that section we introduce matrix notation.

Definition 2.33 (Jacobian matrix)**.** For a function $f : \mathbb{R}^n \to \mathbb{R}^m$ with components denoted by $f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots, f_n(\boldsymbol{x})$, the matrix $\nabla \boldsymbol{f}\,(\boldsymbol{x})$ with m rows and n columns

$$\boldsymbol{J}\,(\boldsymbol{x}) = \begin{bmatrix} \dfrac{\partial f_1}{\partial x_1}(\boldsymbol{x}) & \ldots & \dfrac{\partial f_1}{\partial x_n}(\boldsymbol{x}) \\ \vdots & \ddots & \vdots \\ \dfrac{\partial f_m}{\partial x_1}(\boldsymbol{x}) & \ldots & \dfrac{\partial f_m}{\partial x_n}(\boldsymbol{x}) \end{bmatrix} \tag{2.37}$$

is called the *Jacobian matrix* or *derivative matrix* of $\boldsymbol{f}$ [Edw94, Lue04].

Definition 2.34 (Gradient)**.** For a function $f : \mathbb{R}^n \to \mathbb{R}$ the derivative matrix is reduced to a row vector with n elements

$$\nabla f\,(\boldsymbol{x})^T = \begin{bmatrix} \dfrac{\partial f}{\partial x_1}(\boldsymbol{x}) & \ldots & \dfrac{\partial f}{\partial x_n}(\boldsymbol{x}) \end{bmatrix} \tag{2.38}$$

The column vector containing the same elements

$$\nabla \boldsymbol{f}\,(\boldsymbol{x}) = \begin{bmatrix} \dfrac{\partial f}{\partial x_1}(\boldsymbol{x}) \\ \vdots \\ \dfrac{\partial f}{\partial x_n}(\boldsymbol{x}) \end{bmatrix} \tag{2.39}$$

is called the *gradient* of f [Edw94, Lue04].

The Jacobian matrix can be expressed in terms of the gradient of each function f_i.

$$\boldsymbol{J}\,(\boldsymbol{x}) = \begin{bmatrix} \nabla f_1\,(\boldsymbol{x}) & \nabla f_2\,(\boldsymbol{x}) & \ldots & \nabla f_m\,(\boldsymbol{x}) \end{bmatrix}^T \tag{2.40}$$

It should be dully noted that the definition of the gradient as a column vector or as a row vector differs among authors. For example in [Lue04, AMO93] the gradient is a row vector, whereas in [BT97, BG91, BV04] the gradient is defined as a column vector. We adopt the convention that by default the gradient is a *column* vector, since this is how we defined vectors in matrix notation.

Using the gradient or derivative matrix we can express the directional derivative in Eq. 2.34 by a inner product operation:

$$D_{\boldsymbol{d}}\boldsymbol{f}\,(\boldsymbol{x}) = \nabla \boldsymbol{f}\,(\boldsymbol{x}) \cdot \boldsymbol{d} = \nabla \boldsymbol{f}\,(\boldsymbol{x})^T\,\boldsymbol{d} \tag{2.41}$$

The interpretation of the gradient is that it "points" into the direction of the maximum change of $f\,(\boldsymbol{x})$. Gradient-based optimization algorithms exploit this information to find the point $\boldsymbol{x}$ where the function attains its maximum or minimum.

Definition 2.35 (Hessian)**.** For a function $f : \mathbb{R}^n \to \mathbb{R}$ with continuous partial derivatives $\partial f / \partial x_j$ for $j = 1, \ldots, n$, the square matrix $\nabla^2 \boldsymbol{f}\,(\boldsymbol{x})$

$$\nabla^2 \boldsymbol{f}\,(\boldsymbol{x}) = \begin{bmatrix} \dfrac{\partial f}{\partial x_1 \partial x_1}(\boldsymbol{x}) & \dfrac{\partial f}{\partial x_1 \partial x_2}(\boldsymbol{x}) & \ldots & \dfrac{\partial f}{\partial x_1 \partial x_n}(\boldsymbol{x}) \\ \dfrac{\partial f}{\partial x_2 \partial x_1}(\boldsymbol{x}) & \dfrac{\partial f}{\partial x_2 \partial x_2}(\boldsymbol{x}) & \ldots & \dfrac{\partial f}{\partial x_2 \partial x_n}(\boldsymbol{x}) \\ \vdots & \ddots & \vdots & \\ \dfrac{\partial f}{\partial x_n \partial x_1}(\boldsymbol{x}) & \dfrac{\partial f}{\partial x_n \partial x_2}(\boldsymbol{x}) & \ldots & \dfrac{\partial f}{\partial x_n \partial x_n}(\boldsymbol{x}) \end{bmatrix} \tag{2.42}$$

is called the *Hessian* of $\boldsymbol{f}$ [Lue04].

Definition 2.36 (Critical point)**.** If there is a point $\boldsymbol{x}$ where $\nabla \boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{0}$, then the point $\boldsymbol{x}$ is called a *critical point*.

In this report we focus on objective functions based on real-valued vector functions $f : \mathbb{R}^n \rightarrow \mathbb{R}$. Several of the optimization methods considered in our study assume that the objective function can be approximated near a point $\boldsymbol{a}$ with a *first order Taylor's series*

$$\mathcal{T}_1(\boldsymbol{x}) = f(\boldsymbol{a}) + \nabla f(\boldsymbol{a})^T (\boldsymbol{x} - \boldsymbol{a}) \tag{2.43}$$

or with a *second order Taylor's series* of the form

$$\mathcal{T}_2(\boldsymbol{x}) = f(\boldsymbol{a}) + \nabla f(\boldsymbol{a})^T (\boldsymbol{x} - \boldsymbol{a}) + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{a})^T \nabla^2 f(\boldsymbol{a})(\boldsymbol{x} - \boldsymbol{a}) \tag{2.44}$$

Alternatively, if we let $\boldsymbol{d} = \boldsymbol{x} - \boldsymbol{a}$ to denote a direction, Eq. 2.43 and Eq. 2.44 can be written as

$$\mathcal{T}_1(\boldsymbol{a} + \boldsymbol{d}) = f(\boldsymbol{a}) + \nabla f(\boldsymbol{a})^T \boldsymbol{d} \tag{2.45}$$

and

$$\mathcal{T}_2(\boldsymbol{a} + \boldsymbol{d}) = f(\boldsymbol{a}) + \nabla f(\boldsymbol{a})^T \boldsymbol{d} + \frac{1}{2}\boldsymbol{d}^T \nabla^2 f(\boldsymbol{a})\boldsymbol{d}. \tag{2.46}$$

2.2 Optimization Theory

A natural point of departure in the study of optimization theory is the question about the conditions under which a function $f(x)$ has a minimum or maximum value. According to the *Weierstrass theorem* [Shi96, Lue04], every continuous real-valued function $f(x)$ defined on a compact set Ω has a maximum value $f^*(x_{max})$ and a minimum value $f^*(x_{min})$ for the optimal points $x_{max}, x_{min} \in \Omega$.

Unfortunately, finding the optimal points in the entire function domain Ω can be extremely difficult unless certain convexity properties apply to the problem in question. This has prompted the development of methods to find the optimal solution x^* in a small neighborhood of x^*. Therefore, one has to differentiate between *global optima* and *local optima* [Lue04, PS98, Eng06]. In what follows we assume minimization problems with objective function $f : \Omega \rightarrow \mathbb{R}$, where $\Omega \subset \mathbb{E}^n$.

Definition 2.37 (Global optimum)**.** A point $\boldsymbol{x}^* \in \Omega$ is called a *global optimum (global minimum) point* of the objective function $f(\boldsymbol{x})$ if

$$f(\boldsymbol{x}^*) \leq f(\boldsymbol{x}), \forall \boldsymbol{x} \in \Omega. \tag{2.47}$$

If the inequality is strict

$$f(\boldsymbol{x}^*) < f(\boldsymbol{x}), \forall \boldsymbol{x} \in \Omega \tag{2.48}$$

then the point $\boldsymbol{x}^*$ is called a *strict global minimum*.

Definition 2.38 (Local optimum)**.** A point $\boldsymbol{x}^* \in \Omega$ is called a *local optimum (local minimum) point* of the objective function $f(\boldsymbol{x})$ in δ-neighborhood $\mathcal{N}_\delta = \{\boldsymbol{x} : \boldsymbol{x} \in \Omega, \|\boldsymbol{x} - \boldsymbol{x}^*\| < \delta\}$ if

$$f(\boldsymbol{x}^*) \leq f(\boldsymbol{x}), \forall \boldsymbol{x} \in \mathcal{N}_\delta. \tag{2.49}$$

If the inequality is strict

$$f(\boldsymbol{x}^*) < f(\boldsymbol{x}), \forall \boldsymbol{x} \in \mathcal{N}_\delta. \tag{2.50}$$

then the point $\boldsymbol{x}^*$ is called a *strict local minimum*.

Optimum conditions for maximization problems are defined the same way using the opposite inequality signs (*i. e.*, $f(\boldsymbol{x}^*) \geq f(\boldsymbol{x})$ and $f(\boldsymbol{x}^*) > f(\boldsymbol{x})$).

Definition 2.39 (Convex function)**.** A function $f(\boldsymbol{x})$ is said to be convex if [BV04, Lue04]

i) the domain Ω of $f(\boldsymbol{x})$ is a convex set (see Section 2.1.3)

ii) for all $\boldsymbol{x}_1, \boldsymbol{x}_2 \in \Omega$ and $0 < \lambda < 1$ the function satisfies *Jensen's inequality*

$$f(\lambda \boldsymbol{x}_1 + (1-\lambda)\boldsymbol{x}_2) \leq \lambda f(\boldsymbol{x}_1) + (1-\lambda) f(\boldsymbol{x}_2) \tag{2.51}$$

The function $f(\boldsymbol{x})$ is called *strictly convex* if the condition ii) is changed to

$$f(\lambda \boldsymbol{x}_1 + (1-\lambda)\boldsymbol{x}_2) < \lambda f(\boldsymbol{x}_1) + (1-\lambda) f(\boldsymbol{x}_2) \tag{2.52}$$

for all $0 < \lambda < 1$ and $\boldsymbol{x}_1, \boldsymbol{x}_2 \in \Omega$ such that $\boldsymbol{x}_1 \neq \boldsymbol{x}_2$.

For a convex function local minima are also global minima. This is a very powerful property, which was paramount for the development of efficient methods to find the global minima of convex functions. Unfortunately, there is no similar property that allows finding the global maxima of convex functions.

Definition 2.40 (Concave function)**.** A function $g = -f$ defined on a convex set Ω is called *concave* if the function f is convex.

For a concave function g defined as above, finding the global maxima is equivalent to finding the global minima of the function f.

Definition 2.41 (Convex optimization problem)**.** The problem of finding the global minima of a convex function defined on a convex set is called a *convex optimization problem*. A convex optimization problem has a convex objective function and convex constraints.

Definition 2.42 (Linear optimization problem)**.** An optimization problem with linear objective function and linear constraints is called a *linear optimization problem*. A linear optimization problem is always a convex problem, but the converse is not true. In addition a linear problems is also a concave problem [PS98].

Definition 2.43 (Nonlinear optimization problem)**.** If an optimization problem is not convex or linear it is then called a *nonlinear optimization problem*.

The conditions that govern if a point $\boldsymbol{x}$ is a local minimum are very important for the development and analysis of efficient optimization algorithms. They are divided into *first order* and *second order* conditions. First order conditions apply when the objective function has first-order derivatives that are all continuous, which we denote by $f \in C^1$. Similarly, the second order conditions apply when the objective function has both first and second-order continuous partial derivatives, which is denoted by $f \in C^2$ [Lue04, BV04].

Definition 2.44 (First-order conditions for unconstrained functions)**.** Consider an *unconstrained* optimization problem with an objective function $f \in C^1$ defined on a set Ω. The *interior* point $\boldsymbol{x}^* \in \Omega$ is a local minimum if the gradient vector is equal to zero:

$$\nabla f(\boldsymbol{x}^*) = \boldsymbol{0} \tag{2.53}$$

Definition 2.45 (Second-order conditions for unconstrained functions)**.** Consider an *unconstrained* optimization problem with an objective function $f \in C^2$ defined on a set Ω. The *interior* point $\boldsymbol{x}^* \in \Omega$ is a local minimum if the gradient vector is equal to zero and the Hessian matrix is positive definite (see Section 2.1.4):

$$\nabla f(\boldsymbol{x}^*) = \mathbf{0} \tag{2.54}$$

$$\nabla^2 f(\boldsymbol{x}^*) \text{ is positive definite} \tag{2.55}$$

For constrained optimization problems the local minima can occur on the boundary of the set Ω. For these cases it is appropriate to apply the *Karush-Kuhn-Tucker (KKT)* optimality conditions [Lue04, BV04, PM04]. The KKT-conditions rely on the definition of *active constraints* and *regular points* [Lue04].

Definition 2.46 (Active constraints)**.** For a constrained optimization problem

$$\begin{array}{ll} \text{minimize} & f(\boldsymbol{x}) \\ \text{subject to} & \boldsymbol{g}(\boldsymbol{x}) \leq \mathbf{0} \\ & \boldsymbol{h}(\boldsymbol{x}) = \mathbf{0} \\ & \boldsymbol{x} \in \Omega \end{array}$$

the inequality constraint $g_i(\boldsymbol{x})$ is said to be *active* at the point $\boldsymbol{x}$ if $g_i(\boldsymbol{x}) = 0$. Otherwise the constraint $g_i(\boldsymbol{x})$ is said to be *inactive*.

Definition 2.47 (Regular point)**.** For a constrained optimization problem

$$\begin{array}{ll} \text{minimize} & f(\boldsymbol{x}) \\ \text{subject to} & \boldsymbol{g}(\boldsymbol{x}) \leq \mathbf{0} \\ & \boldsymbol{h}(\boldsymbol{x}) = \mathbf{0} \\ & \boldsymbol{x} \in \Omega \end{array}$$

let I be the set of indices i such that $g_i(\boldsymbol{x}) = 0$. The point $\boldsymbol{x}$ is said to be a *regular point* if the gradient vectors $\nabla h_1(\boldsymbol{x}), \ldots, \nabla h_m(\boldsymbol{x})$ and $\nabla g_i(\boldsymbol{x}), \forall i \in I$ are linearly independent.

Definition 2.48 (KKT conditions for constrained problems)**.** Given a constrained optimization problem of the type

$$\begin{array}{ll} \text{minimize} & f(\boldsymbol{x}) \\ \text{subject to} & \boldsymbol{g}(\boldsymbol{x}) \leq \mathbf{0} \\ & \boldsymbol{h}(\boldsymbol{x}) = \mathbf{0} \\ & \boldsymbol{x} \in \Omega \end{array}$$

with $f, \boldsymbol{g}, \boldsymbol{h} \in C^1$, assume that the *regular* point $\boldsymbol{x}^*$ is a local minimum optimum. This implies that there must exist vectors $\boldsymbol{\lambda} \in \mathbb{E}^m$ and $\boldsymbol{\mu} \in \mathbb{E}^k$ with $\boldsymbol{\mu} \geq \mathbf{0}$ such that

$$\boldsymbol{\mu}^T \boldsymbol{g}(\boldsymbol{x}^*) = 0 \tag{2.56}$$

$$\nabla f(\boldsymbol{x}^*) + \boldsymbol{\lambda}^T \nabla \boldsymbol{h}(\boldsymbol{x}^*) + \boldsymbol{\mu}^T \boldsymbol{g}(\boldsymbol{x}^*) = \mathbf{0}. \tag{2.57}$$

The components $\lambda_1, \ldots, \lambda_k$ of the vector $\boldsymbol{\lambda}$ are called *Lagrange* multipliers [Lue04, PM04].

Definition 2.49 (The dual problem)**.** For a constrained optimization problem (called the *primal problem*)

$$\begin{array}{ll} \text{minimize} & f(\boldsymbol{x}) \\ \text{subject to} & \boldsymbol{g}(\boldsymbol{x}) \leq \mathbf{0} \\ & \boldsymbol{h}(\boldsymbol{x}) = \mathbf{0} \\ & \boldsymbol{x} \in \Omega \end{array}$$

where f is a convex function defined on a convex set, define the *Lagrangian* $\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu})$

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\boldsymbol{x}) + \boldsymbol{\lambda}^T \boldsymbol{h}(\boldsymbol{x}) + \boldsymbol{\mu}^T \boldsymbol{g}(\boldsymbol{x}) \tag{2.58}$$

with $\boldsymbol{\lambda} \geq \mathbf{0}$. Using the Lagrangian, we define the *dual problem*

$$\begin{array}{ll} \text{maximize} & \mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) \text{ over } \lambda, \mu \\ \text{subject to} & \boldsymbol{\lambda} \geq \mathbf{0} \end{array}$$

The solution $\boldsymbol{x}$ to the min-max optimization problem

$$\min_{\boldsymbol{x}} \max_{\boldsymbol{\lambda}, \boldsymbol{\mu}} \mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) \tag{2.59}$$

is a solution to the the primal problem as well [Eng06].

The dual problem presented above can be used to convert a constrained problem to a unconstrained problem.

2.3 Graph Theory

The routing problems discussed are typically cast in the language of graph theory. The purpose of this section is to introduce the basic notions of graph theory and set them in connection to their corresponding parts in the theory of routing in computer networks.

Definition 2.50 (Graph)**.** An *undirected graph* $G = (V, E)$ consists of a nonempty set V of *vertices* (also called *nodes*) and a collection E of pairs of distinct vertices from V. The elements of V are called *edges*, *links* or *arcs*. In an undirected graph the edge (i, j) between node i and node j is indistinguishable for the edge (j, i). In the case of a *directed graph* (also called *digraph*) the edges (i, j) and (j, i) are distinct [BT97, BG91, PS98]. For the directed edge (i, j) we say that the edge is *outgoing* from the perspective of node i and *incoming* for node j.

The graph of a computer network uses vertices to represent *hosts (nodes)* and edges to denote *communication links* (one hop) connecting two hosts. Since properties of network traffic often depend on the direction in which the traffic goes we focus exclusively on digraphs.

If (i, j) is an edge in $G(V, E)$ then we say that edge (i, j) is *incident* to node i and j. Additionally, we say that i and j are *adjacent nodes* (or *neighbors*).

The number of outgoing edges (i, j) is called the *outdegree* of node i. Similarly, the *indegree* of node j is defined as the number of incoming edges at j from various nodes i.

Definition 2.51 (Path)**.** A directed *path* $P(n_1, n_k)$ in a directed graph $G(V, E)$ is a sequence of nodes $n_1, n_2, \ldots, n_k$ with $k \geq 2$. This definition is equivalent to saying that the path P is a sequence of $(k-1)$ links $l_1, \ldots, l_{k-1}$. The number of links in a path P defines the *length* of the path, which is $(k-1)$ in this case.

In a path $P(n_1, n_k)$, the node n_1 is called the *source* or *origin* node and n_k is called the *destination* node. For a node n_i in P all nodes $\{n_j : j < i\}$ (if any) are called *upstream* nodes and all nodes $\{n_j : j > i\}$ are called *downstream* nodes.

A graph $G(V, E)$ is said to be *connected* if, for each pair of nodes $(i, j) : i, j \in V, i \neq j$, there is a path $P(i, j)$. If in addition to that, there is also a path $P(j \rightarrow i)$ (*i. e.*,, there is a path from i to j and another from j to i) the graph is said to be *strongly connected*.

Definition 2.52 (Weighted graph)**.** In a *weighted graph* $G(V, E)$ all edges have an associated number $w \in R$ called the *weight* that represents a metric of interest (*e. g.*, cost, bandwidth, delay). Clearly, if we consider n metrics simultaneously then the weight is a vector $\boldsymbol{w} = [w_1, \ldots w_n]$. A weighted graph can be represented by a symmetric matrix $\boldsymbol{A} = \left[\boldsymbol{w}_{ij}\right]$ where $\boldsymbol{w}_{ij}$ is set to a suitable value (*e. g.*, 0 or ∞) if there is no edge (i, j) in E.

In routing optimization the weights of the graph generally fall into one of the following categories:

additive: delay, jitter, cost

multiplicative: packet loss

min-max: bandwidth

Multiplicative weights can be turned into additive weights by taking the logarithm of their product [VMK04]. Therefore, we focus in the remainder of the report on additive and min-max weights only.

2.4 Routing Optimization Problems

The typical computer network scenario that leads to a optimization problem is the following. It is assumed that information about network topology is available in the form of a weighted digraph $G(V, E)$. The link weight in the graph represents a set of metrics of interest, such as bandwidth, delay, jitter, packet loss and cost. Furthermore, information about the *flow demands* is available as well. A flow demand is a set of *path constraints* between a pair of nodes (A and B) in the graph. In its simplest form, the flow demand contains only the bandwidth required to transfer data from A to B. We assume that flow demands depend on the direction of the path. The path from B to A is free to use a different flow demand, if any. The most complex flow demands can contain up to the number of elements of the link weight used in the digraph.

There can be more than one path connecting A and B and each path consists of one or more links. The goal is to spread the network traffic between each pair of nodes A and B in such a way as to accommodate the flow demand, if possible. This is called the feasibility problem for *bifurcated flows*.

Sometimes it is required that the entire traffic flow between two nodes follows a single path, instead of being spread across several. We call this a feasibility problem for *non-bifurcated flows*.

If, in addition to the flow demands, it is desired that the traffic delay across the entire network be minimized as well, then the feasibility problem becomes an optimization problem.

It is not in our intention to survey the different types of optimization problems that can be applied to computer networks. The interested reader can find this type of information

in [AMO93, PM04, DF07]. Our approach is to consider two optimization problems and two feasible problems that can fit into the ORP framework discussed in Chapter 1.3 and Chapter 1.4. We consider the optimal routing problem [BG91] and the multi-constrained optimal path (MCOP) selection problem [VMK04] as optimal problems. By removing the objective function each of the problems is cast into a feasibility framework.

In the optimal routing problem we have a set of flow demands consisting of bandwidth constraints and a convex cost function to minimize. Following Chapter 5 in [BG91] we denote by F_{ij} the flow on the *link* between i and j. We define the cost function by

$$\sum_{(i,j)} D_{ij}\left(F_{ij}\right) \tag{2.60}$$

where D_{ij} is a convex cost function in F_{ij}. For example, denoting by C_{ij} the available bandwidth of the link (i,j) and by d_{ij} the sum of processing and propagation delay, then

$$D_{ij}\left(F_{ij}\right) = \frac{F_{ij}}{C_{ij} + Fij} + d_{ij}F_{ij} \tag{2.61}$$

The set W contains pair of distinct nodes $w = (i,j)$ for which there is a flow demand r_w. Further, we denote by P_w the set of all paths connecting a pair of nodes w and by x_p the flow allocated to path p. With this notation we can write

$$F_{ij} = \sum_{\forall p:\,(i,j)\in p} x_p \tag{2.62}$$

where the sum is taken over all paths p containing link (i,j). The complete optimal routing problem is shown in Table 2.1.

$$\begin{aligned} \text{minimize} \quad & \sum_{(i,j)} D_{ij}\left(\sum_{\forall p:\,(i,j)\in p} x_p\right) \\ \text{subject to} \quad & \sum_{p\in P_w} x_p = r_w, \text{ for all } w \in W \\ & x_p \geq 0, \text{ for all } p \in P_w \text{ and } w \in W \end{aligned}$$

Table 2.1: Optimal routing problem

If we are only interested in ensuring that all flow demands are satisfied, then we have a *feasible routing* problem as shown in Table 2.2. The last inequality constraint ensures that all paths passing along the link (i,j) do not consume more bandwidth than available.

In the case of a multi-constrained path (MCP) problem we attempt to find a constrained path at a time. This is a feasibility problem. Each link in $G(V,E)$ has a number of *additive* QoS metrics (*e. g.* , delay and delay jitter) as well as *min-max* QoS metrics (*e. g.* , bandwidth). The constraints on *min-max* metrics can be dealt with by pruning the links of the graph that do not satisfy the *min-max* constraints [VMK04, KVMKK02]. When this is done we are left with additive metrics only. For $i = 1, \ldots, m$ we denote by $w_i(u,v)$ the i-th additive metric for the link (u,v) between nodes u and v such that $(u,v) \in E$. The MCP optimization problem for m additive constraint values L_i on the requested path is shown in Table 2.3.

find	x_p, for all $w \in P_w$ and all $p \in P_w$
subject to	$\sum_{p \in P_w} x_p = r_w$, for all $w \in W$
	$x_p \geq 0$, for all $p \in P_w$ and $w \in W$
	$\sum_{\forall p:\,(i,j) \in p} x_p \leq C_{ij}$ for all $p \in P_w$ and $w \in W$

Table 2.2: Feasible routing problem

find	path P
subject to	$w_i(P) = \sum_{(u,v) \in P} w_i(u,v) \leq L_i$ for $i = 1 \ldots m$ and all $(u,v) \in E$

Table 2.3: Multi-constrained path selection problem (MCP)

The MCP problem problem can be comfortably extended to an multi-constrained optimal path (MCOP) selection problem by minimizing or maximizing over one of the metrics w_i. The alternative is to define a path-length function over all metrics [VMK04, KVMKK02] and to minimize the path-length function itself. More details about this approach are provided in Chapter 8.

The four problems presented here imply that for each pair of nodes (u, v) all possible paths connecting u and v have been precomputed leaving the optimization problem itself to deal with selecting one of these paths according to the constraints. All paths connecting two nodes (u, v) in a graph $G(V, E)$ can be computed efficiently, for example by the depth-first search method [CLR01]. The exception is described in Chapter 8, where we see that path discovery is part of SAMCRA's algorithm.

2.5 Algorithms and Complexity

The optimality conditions presented in Chapter 2.2 provide good hints on how to solve optimization problems by using *differential analysis*. For example, in the case of unconstrained optimization, the first-order conditions say that if one can solve the system of differential equations

$$\nabla f(\boldsymbol{x}^*) = \mathbf{0} \tag{2.63}$$

the solution $\boldsymbol{x}^*$ is a local minimum (and also a global minimum in the case of a convex function).

This approach works well for problems with a small number of variables. For larger problems the effort to solve them becomes prohibitive. In that case it is more reasonable to approximate the solution using an *algorithmic* approach.

Definition 2.53 (Algorithm)**.** An algorithm is a step-by-step iterative procedure to solve a problem [Lue04, GJ79]. The general form of an algorithm for a feasible space X is

$$\boldsymbol{x}_{k+1} = \Xi(x_k) \tag{2.64}$$

where $\boldsymbol{x}_k \in X$ denotes step k of the algorithms and Ξ is a mapping from X to X. The term Ξ can be a matrix, a function or some other form of mathematical expression. When applied to a vector $\boldsymbol{x} \in X$, Ξ produces another vector $\boldsymbol{y} \in X$.

The algorithmic approaches can be divided into three categories:

i) numerical methods

ii) heuristics

iii) meta-heuristics.

Numerical methods solve problems by function approximation and finite differential calculus [Ham86, DF07]. For example, a trick often used is to approximate a function f at a point x with the polynomial generated by the second-degree Taylor series. The resulting gradient vector and the Hessian matrix can provide valuable insight about how the function changes in different directions in a neighborhood of x. Numerical methods require that certain conditions apply to the problem in question or else the approximations may not converge to an optimal value. Further, they are susceptible to roundoff and truncation errors and to stability problems related to the feedback loop in Eq. 2.64 [Ham86]. Chapter 3–6 present various examples of numerical methods for optimization.

If numerical methods do not work well on the problem at hand, then it may be possible to apply a heuristic [DF07]. Heuristics are algorithms that search the feasible space in an intelligent way. In general, heuristics involve a tradeoff between computation time and accuracy: fast heuristics sometimes cannot find the optimal solution and accurate heuristics find the optimal solution always, but for some problem instances they can take an unreasonable amount of time to finish. In Chapter 8 we describe a heuristic called SAMCRA.

Metaheurstics are algorithms that combine various heuristics in an effort to obtain an approximate solution even in the case of difficult problem instances [BR03, DF07]. Metaheuristic often employ a probabilistic element in order to avoid being trapped in a local minima. The PSO method described in Chapter 7 is an example of metaheuristic.

All algorithms use some type of *convergence criteria*. Intuitively, we say that an algorithm has converged when $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k$ or when $f(\boldsymbol{x}_{k+1}) = f(\boldsymbol{x}_k)$, meaning that $\boldsymbol{x}^* = \boldsymbol{x}_k$. However, this may not happen at all when the algorithm is implemented by a computer program. Since computers are finite-state machines they have finite precision in representing real numbers (*i. e.*, they use floating-point numbers). This implies that numbers are rounded off, which leads to round-off errors. In practice, it means that the optimal value of the algorithm approaches the true value $\boldsymbol{x}^*$ in an ϵ-neighborhood dictated by the floating-point precision used. Therefore, the convergence criteria should be of one of the forms shown below [Eng06]:

i) $f(\boldsymbol{x}_{k+1}) - f(\boldsymbol{x}_k) < \epsilon(1 + |f(\boldsymbol{x}_{k+1})|)$,

ii) $\|\boldsymbol{x}_k - \boldsymbol{x}_{k+1}\| < \sqrt{\epsilon}(1 + \|\boldsymbol{x}_{k+1}\|)$,

iii) $\|\nabla f(\boldsymbol{x}_{k+1})\| < \epsilon^{1/3}(1 + |f(\boldsymbol{x}_{k+1})|)$.

Additionally, one needs a *stopping condition*. Obviously if the algorithm converges according to one of the criteria above, then it can stop. Otherwise, one needs an upper bound ζ on the number of steps k performed by the algorithm. If the $k = \zeta$ the algorithm stops. Clearly, the choice of ζ is very important since setting the value too low causes the algorithm to stop

even if the problem has an optimal solution, whereas if the value is too high the algorithm will run for a long time, even if no solution exists.

An important issue related to algorithms is that of *computational complexity*. It is convenient to express complexity in terms of the size of input data to the algorithm. Then, the complexity is related to space required to store the input data and to the time it takes to run the algorithm until a solution is found[1]. In general it is assumed that time complexity has more impact than space complexity. This means that in general we are confident that there is a way to store the input data, but we are worried about the running time of the algorithm. The procedure to estimate the time complexity can be briefly explained as follows.

It is assumed that the input to an algorithm consists of n elements. For example, the n elements can be the set of edges in a graph $G(V, E)$. The algorithm must process the input data in a number of steps that is roughly proportional to n. This proportion is expressed as a polynomial $an^x + bn^y + \ldots cn^z$, where a, b, c, x, y, z are constant integers[2]. Then the time complexity is the polynomial term that dominates the expression when $n \to \infty$. Unfortunately, estimating the numbers of steps can be nontrivial. Therefore one tends to settle for estimating an upper bound, which is referred to as the *worst-case scenario*. The worst-case scenario is an estimate on the maximum number of steps that the algorithm would require for any instance of a problem. If, for example, it is calculated that the worst-case scenario of a problem with n elements scales as n^2 (the dominating factor in the polynomial), then we write $\mathrm{O}\left(n^2\right)$. If the problem scales as a constant, which means it is independent of the size of the input, we write $\mathrm{O}\left(1\right)$.

Sometimes it is possible to also compute (in a similar way) a lower bound, which we call the *minimum complexity*. The minimum complexity tells us what is the the minimum computational effort for any problem instance. If, for example, the dominating factor in the minimum complexity polynomial is n, we write $\mathrm{o}\left(n\right)$.

So far we have assumed that the time complexity can be approximated by a polynomial. Actually, there is also a large volume of problems with exponential time complexity (*e. g.*,, $\mathrm{O}\left(n!\right)$ and $\mathrm{O}\left(k^n\right)$ for $k \in \mathbb{Z}$) [PS98].

Problems that can be solved in polynomial time are said to belong to the class of problems $\mathcal{P}$ and those with exponential time complexity are confined to the class $\mathcal{NP}$[3]. Some problems are said to be NP-complete. The implications of this are that if a polynomial-time algorithm is found to solve the problem, then it would mean that all problems in the class $\mathcal{NP}$ can be solved in polynomial time. Should this occur, it would also answer one of the greatest questions in computer science, namely if the class $\mathcal{P}$ is equivalent to $\mathcal{NP}$. The interested reader can find more information in [Coo83, GJ79].

In terms of the problems considered in Chapter 2.4, the MCP and MCOP problem are NP-complete [WC96]. However, Kuipers and Van Mieghem have shown [KVM05] that NP-complete behavior is unlikely to occur in realistic communication networks. The optimal routing problem is NP-complete only if the solution is constrained to non-bifurcated paths [PM04].

[1] We assume that $\zeta = \infty$.

[2] They don't have to be integers, but we assume this for simplicity purposes.

[3] This is actually a gross simplification, but it is good enough for the scope of the report.

Chapter 3

Descent Methods

In this chapter we focus on basic *descent methods*, which are iterative methods used for solving unconstrained optimization problems. We selected the steepest descent method and Newton's method because of their importance to the field of unconstrained optimization. In fact, the conjugate gradient method presented in Chapter 4 is an intermediate between these two methods. Additionally, the gradient projection method presented in Chapter 6 is based on the steepest descent method [Lue04].

The general idea of descent methods is that given a point $\boldsymbol{x}$ in the feasible space they attempt to discover a *descent direction* $\boldsymbol{d}$. A descent direction $\boldsymbol{d}$ is a vector such that $f(\boldsymbol{x}+\boldsymbol{d}) < f(\boldsymbol{x})$. However, in order to speed up the convergence it is desired to scale $\boldsymbol{d}$ by a factor σ chosen in such a way that $f(\boldsymbol{x}+\sigma\boldsymbol{d})$ achieves a minimum value along the line $\boldsymbol{x}+\sigma\boldsymbol{d}$. This is called *line search* and is the second important ingredient of all descent-based optimization methods. Line search is essentially the problem of finding the minimum in one dimension. This can be done either analytically, or through heuristics such as golden section search or Brent's method [PTVF02, Lue04].

It follows from the discussion above that the main property of descent methods is to decrease the value of the objective function at each step k such that

$$f(\boldsymbol{x}_{k+1}) < f(\boldsymbol{x}_k) \tag{3.1}$$

except when $\boldsymbol{x}_k$ is optimal [BV04]. The general descent method (GDM) is shown in Algorithm 1.

```
1: Initialize x_0
2: repeat
3:     Find a descent direction Δ_k
4:     Line search: find a step length σ_k
5:     x_{k+1} ← x_k + σ_k Δ_k
6: until stopping condition is satisfied
```

Algorithm 1: General descent method (GDM)

The choice of an initial point $\boldsymbol{x}_0$ does not affect the convergence of the algorithm to a local minimum, provided that one exists. However, choosing an initial point close to the optimal solution requires less iterations. The procedure to find a descent direction depends on the specific descent algorithm in question. The symbol $\leftarrow$ on line 5 of Algorithm 1 indicates that

the expression or variable on the left side of the symbol is assigned the value (or matrix) resulting from the expression on the right side of the symbol. Within an algorithm we use the symbol "=" for comparison and not for assignment.

The methods described in this chapter can applied only to unconstrained problems. Constrained problems can be solved only if, for example, one can construct the Lagrangian of the objective function and then solve the resulting min-max problem as described in Chapter 2.2.

3.1 Steepest Descent Method

We recall from Chapter 2.1.5 that the gradient at $\boldsymbol{x}$ points in the direction of the greatest increase of $f(\boldsymbol{x})$. The steepest descent method (SDM) searches for the next feasible point in the direction opposite to the gradient(*i. e.*, $\Delta_k = -\nabla f(\boldsymbol{x}_k)$), as shown in Algorithm 2. The step length σ_k is restricted to nonnegative values.

1: Initialize $\boldsymbol{x}_0$
2: **repeat**
3: Line search: find σ_k that minimizes $f(\boldsymbol{x}_k - \sigma_k \nabla f(\boldsymbol{x}_k))$
4: $\boldsymbol{x}_{k+1} \leftarrow \boldsymbol{x}_k - \sigma_k \nabla f(\boldsymbol{x}_k)$
5: **until** stopping condition is satisfied

Algorithm 2: Steepest descent method (SDM)

The steepest method is motivated by considering first-order Taylor approximation in the direction $\boldsymbol{d}_k$ from the point $\boldsymbol{x}_k$

$$f(\boldsymbol{x}_k + \boldsymbol{d}_k) \approx \mathcal{T}_1(\boldsymbol{x}_k + \boldsymbol{d}_k) = f(\boldsymbol{x}_k) + \nabla f(\boldsymbol{x}_k)^T \boldsymbol{d}_k \tag{3.2}$$

Accordingly, the term $\nabla f(\boldsymbol{x}_k)^T \boldsymbol{d}$ provides the descent direction $-\Delta_k$ when $\|\boldsymbol{d}\| = 1$ [BV04].

3.2 Newton's Method

Newton's method is based on using a second order Taylor approximation of the objective function $f(\boldsymbol{x})$ in a direction $\boldsymbol{d}_k$ from the point $\boldsymbol{x}_k$

$$f(\boldsymbol{x}_k + \boldsymbol{d}_k) \approx \mathcal{T}_2(\boldsymbol{x}_k + \boldsymbol{d}_k) = f(\boldsymbol{x}_k) + \nabla f(\boldsymbol{x}_k)^T \boldsymbol{d}_k + \frac{1}{2}\boldsymbol{d}_k^T \nabla^2 f(\boldsymbol{x}_k)\, \boldsymbol{d}_k. \tag{3.3}$$

The idea is to find the optimum solution of the Taylor approximation and use that as the step Δ_k. We do this by calculating the gradient of $\mathcal{T}_2(\boldsymbol{x}_k + \boldsymbol{d}_k$ and then applying Definition 2.44 on page 25 [BV04, LRV01, Lue04]. The gradient of $\mathcal{T}_2(\boldsymbol{x}_k + \boldsymbol{d}_k)$ is

$$\nabla \mathcal{T}_2(\boldsymbol{x}_k + \boldsymbol{d}_k) = \nabla f(\boldsymbol{x}_k) + \nabla^2 f(\boldsymbol{x}_k)\, \boldsymbol{d}_k. \tag{3.4}$$

If we let

$$\nabla \mathcal{T}_2(\boldsymbol{x}_k + \boldsymbol{d}_k) = \boldsymbol{0} \tag{3.5}$$

and solve for $\boldsymbol{d}_k$, we obtain

$$\Delta_k = \boldsymbol{d}_k = -\left[\nabla^2 f(\boldsymbol{x}_k)\right]^{-1} \nabla f(\boldsymbol{x}_k). \tag{3.6}$$

Newton's method uses a step length $\sigma_k = 1$. It possible to decide the step length using a linear search in which case, the method is called the *damped* or *guarded* Newton's method. In Algorithm 3 we assume that there is a boolean variable controlling which version of the algorithm is being used.

1: Initialize $\boldsymbol{x}_0$
2: **repeat**
3: Compute $\nabla f(\boldsymbol{x}_k)$ and $-\left[\nabla^2 f(\boldsymbol{x}_k)\right]^{-1}$
4: $\boldsymbol{d}_k \leftarrow -\left[\nabla^2 f(\boldsymbol{x}_k)\right]^{-1} \nabla f(\boldsymbol{x}_k)$
5: **if** damped Newton's method **then**
6: Line search: find σ_k that minimizes $f(\boldsymbol{x}_k + \sigma_k \boldsymbol{d}_k)$
7: **else**
8: $\sigma_k \leftarrow 1$
9: **end if**
10: $\boldsymbol{x}_{k+1} \leftarrow \boldsymbol{x}_k + \sigma_k \boldsymbol{d}_k$
11: **until** stopping condition is satisfied

Algorithm 3: Newton's method (NM)

Chapter 4

Conjugate Gradient Method

The conjugate gradient method (CGM) was initially intended for solving a system $\boldsymbol{Ax} = \boldsymbol{b}$ of n equations in n variables [HS52]. This is equivalent to minimizing the quadratic function

$$f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T\boldsymbol{Ax} - \boldsymbol{b}^T\boldsymbol{x}, \tag{4.1}$$

where $\boldsymbol{A}$ is a $n \times n$ positive definite matrix [GVL96, Lue04]. The main idea of the CGM is to gain better performance than that of the steepest descent method or of the Newton's method by finding the solution in n steps. This implies that the algorithm takes a step of the "right" length along each of the n basis vectors. It also means that each step is taken in a direction orthogonal to the previous direction. In the CGM this is done using conjugate vectors.

Definition 4.1 (Conjugate vectors)**.** Two vectors $\boldsymbol{d}_1$ and $\boldsymbol{d}_2$ are said to be *conjugate* with respect to the positive definite matrix $\boldsymbol{A}$, if $\boldsymbol{d}_1^T\boldsymbol{Ad}_2 = 0$ [Lue04]. A set of vectors $\boldsymbol{d}_0, \ldots, \boldsymbol{d}_k$ that are pairwise conjugate to $\boldsymbol{A}$ (*i. e.*, $\boldsymbol{d}_i^T\boldsymbol{Ad}_j = 0, \forall i \neq j$) are also linearly independent.

Given a set of n conjugate vectors $\boldsymbol{d}_0, \ldots, \boldsymbol{d}_{n-1}$, the solution $\boldsymbol{x}^*$ to the equation $\boldsymbol{Ax} = \boldsymbol{b}$ can be written as the linear combination of conjugate vectors

$$\boldsymbol{x}^* = c_0\boldsymbol{d}_0 + \cdots + c_{n-1}\boldsymbol{d}_{n-1}. \tag{4.2}$$

Each constant c_i can be calculated by multiplying both sides of Eq. 4.2 by $\boldsymbol{d}_i^T\boldsymbol{A}$ such that

$$c_i = \frac{\boldsymbol{d}_i^T\boldsymbol{Ax^*}}{\boldsymbol{d}_i^T\boldsymbol{Ad}_i} = \frac{\boldsymbol{d}_i^T\boldsymbol{b}}{\boldsymbol{d}_i^T\boldsymbol{Ad}_i}. \tag{4.3}$$

The remaining question is how to chose the conjugate vectors. We do not dwell upon the theory surrounding the algorithm, but rather present each step as shown in Algorithm 4. The interested reader can find more information in [HS52, GVL96, Lue04].

We assume that the CGM starts at some point $\boldsymbol{x}_0$ in the feasible space. We let $\boldsymbol{r}_0 = \boldsymbol{Ax}_0 - \boldsymbol{b}$, which can be thought of as the residual value (error) of the equation $\boldsymbol{Ax} = \boldsymbol{b}$ when $\boldsymbol{x}_0 \neq \boldsymbol{x}^*$. The vectors $\boldsymbol{r}_k$ have actually a gradient interpretation in underlaying theory, which explains the name of the method [Lue04]. Additionally, we set $\boldsymbol{d}_0$ to the negative of the gradient (residual) value $\boldsymbol{r}_0$.

The step length σ_k is calculated by the expression on line 5 in Algorithm 4. Each step length σ_k is equivalent to a variable c_i in Eq. 4.2. The new feasible solution $\boldsymbol{x}_{k+1}$ is computed

1: Initialize $\boldsymbol{x}_0$
2: $\boldsymbol{r}_0 \leftarrow \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_0$
3: $\boldsymbol{d}_0 \leftarrow -\boldsymbol{r}_0$
4: **repeat**
5: $\quad \sigma_k \leftarrow -\dfrac{\boldsymbol{r}_k^T \boldsymbol{d}_k}{\boldsymbol{d}_k^T \boldsymbol{A} \boldsymbol{d}_k}$
6: $\quad \boldsymbol{x}_{k+1} \leftarrow \boldsymbol{x}_k + \sigma_k \boldsymbol{d}_k$
7: $\quad \boldsymbol{r}_{k+1} \leftarrow \boldsymbol{A}\boldsymbol{x}_{k+1} - \boldsymbol{b}$
8: $\quad \beta_k \leftarrow \dfrac{\boldsymbol{r}_{k+1}^T \boldsymbol{A} \boldsymbol{d}_k}{\boldsymbol{d}_k^T \boldsymbol{A} \boldsymbol{d}_k}$
9: $\quad \boldsymbol{d}_{k+1} \leftarrow -\boldsymbol{r}_{k+1} + \beta_k \boldsymbol{d}_k$
10: **until** stopping condition is satisfied

Algorithm 4: Conjugate gradient method (CGM)

on Line 6. Line 7–9 are used to compute a new direction $\boldsymbol{d}_{k+1}$, which is orthogonal to the previous direction.

To use the method with a nonquadratic function we perform a second degree Taylor approximation in each point $\boldsymbol{x}_k$. In the neighborhood of those points the Taylor polynomial behaves as a quadratic function. This is the same idea as in the case of Newton's method. In the case of the CGM we replace in Algorithm 4 the residuals $\boldsymbol{r}_k$ by the gradient $\nabla f(\boldsymbol{x}_k)$ and the matrix $\boldsymbol{A}$ by the Hessian $\nabla^2 f(\boldsymbol{x}_k)$ [Lue04]. These changes are shown in Algorithm 5.

The CGM with quadratic approximation (Q-CGM) does not usually converge in n steps. It is customary to restart the algorithm after n steps in order to improve convergence, which should explain the **for**-loop.

1: Initialize $\boldsymbol{x}_0$
2: **repeat**
3: $\quad \boldsymbol{r}_0 \leftarrow \nabla f(\boldsymbol{x}_0)$
4: $\quad \boldsymbol{d}_0 \leftarrow -\boldsymbol{r}_0$
5: $\quad$ **for** $k = 0, 1, \ldots, n-1$ **do**
6: $\quad\quad \sigma_k \leftarrow -\dfrac{\boldsymbol{r}_k^T \boldsymbol{d}_k}{\boldsymbol{d}_k^T \nabla^2 f(\boldsymbol{x}_k)\, \boldsymbol{d}_k}$
7: $\quad\quad \boldsymbol{x}_{k+1} \leftarrow \boldsymbol{x}_k + \sigma_k \boldsymbol{d}_k$
8: $\quad\quad \boldsymbol{r}_{k+1} \leftarrow \nabla f(\boldsymbol{x}_{k+1})$
9: $\quad\quad$ **if** $k \neq N-1$ **then**
10: $\quad\quad\quad \beta_k \leftarrow \dfrac{\boldsymbol{r}_{k+1}^T \nabla^2 f(\boldsymbol{x}_k)\, \boldsymbol{d}_k}{\boldsymbol{d}_k^T \nabla^2 f(\boldsymbol{x}_k)\, \boldsymbol{d}_k}$
11: $\quad\quad\quad \boldsymbol{d}_{k+1} \leftarrow -\boldsymbol{r}_{k+1} + \beta_k \boldsymbol{d}_k$
12: $\quad\quad$ **end if**
13: $\quad$ **end for**
14: $\quad \boldsymbol{x}_0 \leftarrow \boldsymbol{x}_n$
15: **until** stopping condition is satisfied

Algorithm 5: Conjugate gradient method with quadratic approximation (Q-CGM)

The requirement to evaluate the Hessian in each iteration of the Q-CGM leads to a performance bottleneck [Lue04]. The Fletcher-Reeves CGM (FR-CGM) tries to alleviate this

problem by proposing a line search for σ_k. Additionally, the computation of $\boldsymbol{\beta}_k$ does not need to evaluate the Hessian. These changes can be observed on line 6 and 10 in Algorithm 6.

1: Initialize $\boldsymbol{x}_0$
2: **repeat**
3: $\quad \boldsymbol{r}_0 \leftarrow \nabla f(\boldsymbol{x}_0)$
4: $\quad \boldsymbol{d}_0 \leftarrow -\boldsymbol{r}_0$
5: $\quad$ **for** $k = 0, 1, \ldots, N-1$ **do**
6: $\quad\quad$ Line search: find σ_k that minimizes $f(\boldsymbol{x}_k + \sigma_k \boldsymbol{d}_k)$
7: $\quad\quad \boldsymbol{x}_{k+1} \leftarrow \boldsymbol{x}_k + \sigma_k \boldsymbol{d}_k$
8: $\quad\quad \boldsymbol{r}_{k+1} \leftarrow \nabla f(\boldsymbol{x}_{k+1})$
9: $\quad\quad$ **if** $k \neq N-1$ **then**
10: $\quad\quad\quad \beta_k \leftarrow \dfrac{\boldsymbol{r}_{k+1}^T \boldsymbol{r}_{k+1}}{\boldsymbol{r}_k^T \boldsymbol{r}_k}$
11: $\quad\quad\quad \boldsymbol{d}_{k+1} \leftarrow -\boldsymbol{r}_{k+1} + \beta_k \boldsymbol{d}_k$
12: $\quad\quad$ **end if**
13: $\quad$ **end for**
14: $\quad \boldsymbol{x}_0 \leftarrow \boldsymbol{x}_N$
15: **until** stopping condition is satisfied

Algorithm 6: Fletcher-Reeves conjugate gradient method (FR-CGM)

Additional changes in the computation of β_k were suggested by Polak and Ribiere in an effort to increase the algorithm efficiency. Algorithm 7 presents the Polak-Ribiere CGM (PR-CGM).

1: Initialize $\boldsymbol{x}_0$
2: **repeat**
3: $\quad \boldsymbol{d}_0 \leftarrow -\boldsymbol{r}_0 \leftarrow \nabla f(\boldsymbol{x}_0)$
4: $\quad$ **for** $k = 0, 1, \ldots, N-1$ **do**
5: $\quad\quad$ Line search: find σ_k that minimizes $f(\boldsymbol{x}_k + \sigma_k \boldsymbol{d}_k)$
6: $\quad\quad \boldsymbol{x}_{k+1} \leftarrow \boldsymbol{x}_k + \sigma_k \boldsymbol{d}_k$
7: $\quad\quad \boldsymbol{r}_{k+1} \leftarrow \nabla f(\boldsymbol{x}_{k+1})$
8: $\quad\quad$ **if** $k \neq N-1$ **then**
9: $\quad\quad\quad \beta_k \leftarrow \dfrac{(\boldsymbol{r}_{k+1} - \boldsymbol{r}_k)^T \boldsymbol{r}_{k+1}}{\boldsymbol{r}_k^T \boldsymbol{r}_k}$
10: $\quad\quad\quad \boldsymbol{d}_{k+1} \leftarrow -\boldsymbol{r}_{k+1} + \beta_k \boldsymbol{d}_k$
11: $\quad\quad$ **end if**
12: $\quad$ **end for**
13: $\quad \boldsymbol{x}_0 \leftarrow \boldsymbol{x}_N$
14: **until** stopping condition is satisfied

Algorithm 7: Polak-Ribiere conjugate gradient method (PR-CGM)

Conjugate gradient methods are suitable for use in unconstrained optimization problems. If constrained optimization problems can be converted to an unconstrained problem, then the conjugate gradient methods can be used to solve them. Additionally, conjugate gradient methods can be used to solve feasibility problems of the type $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, where $\boldsymbol{A}$ is a $n \times n$ matrix.

Chapter 5

Simplex Method

The simplex method attempts to solve the linear problems with linear constraints of the form shown in Table 5.1.

minimize	$f(\boldsymbol{x}) = \boldsymbol{c}^T\boldsymbol{x}$	
subject to	$\boldsymbol{E}\boldsymbol{x} = \boldsymbol{p}$	(1)
	$\boldsymbol{G}\boldsymbol{x} \geq \boldsymbol{s}$	(2)
	$\boldsymbol{H}\boldsymbol{x} \leq \boldsymbol{t}$	(3)
	$x_i \in \mathbb{R}$ for $i = 1, \ldots, n$	

Table 5.1: Linear optimization problem in general form

This is called the *general form* of the linear optimization problem. The unknown variables are denoted by the column vector $\boldsymbol{x}$ with n elements. The vector $\boldsymbol{c}^T$ holds the coefficients for the unknown variables. The matrix $\boldsymbol{E}$ contains the coefficients for the equality constraints. Similarly, the matrices $\boldsymbol{G}$ and $\boldsymbol{H}$ represent the coefficients for the inequality constraints. The vectors $\boldsymbol{p}$,$\boldsymbol{s}$ and $\boldsymbol{t}$ contain the constants on the right hand side of the equalities and inequalities respectively.

In the description of the linear problem any combination of the constraints (1)–(3) may appear. If there are no constraints of the type (1)–(3), then the linear problem is unconstrained.

In order to apply the simplex method it is necessary to convert the problem description from the general form to the *standard form* shown in Table 5.2.

minimize	$f(\boldsymbol{x}) = \boldsymbol{c}^T\boldsymbol{x}$
subject to	$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$
	$x_i \geq 0$ and $b_i > 0$ for $i = 1, \ldots, N$

Table 5.2: Linear optimization problem in standard form

In this case the matrix $\boldsymbol{A}$ contains the constraints (1)–(3) where the inequality constraints (2) and (3) have been changed to equality constraints [Lue04].

For example, given the the constraint row i in matrix $\boldsymbol{G}$

$$g_{i1}x_1 + g_{i2}x_2 + \cdots + g_{in}x_n \geq s_i \tag{5.1}$$

this row can be converted into standard form by subtracting from it a *surplus* variable y_1 such that

$$g_{i1}x_1 + g_{i2}x_2 + \cdots + g_{in}x_n - y_i = s_1. \tag{5.2}$$

There are just as many surplus variables as rows in the matrix $\boldsymbol{G}$. Similarly, for the constraints row i in matrix $\boldsymbol{H}$

$$h_{i1}x_1 + h_{i2}x_2 + \cdots + h_{in}x_n \geq t_i \tag{5.3}$$

we introduce add a *slack variable* w_i on the right side of the equality sign such that

$$h_{i1}x_1 + h_{i2}x_2 + \cdots + h_{in}x_n + w_i = t_i. \tag{5.4}$$

When surplus variables and slack variables have been introduced we can view the matrix $\boldsymbol{A}$ in Table 5.2 as the original matrix $\boldsymbol{E}$ from Table 5.1 augmented by matrix $\boldsymbol{G}$ and $\boldsymbol{H}$. Similarly, the vector $\boldsymbol{x}$ from Table 5.1 has been augmented by the vector $\boldsymbol{y}$ and $\boldsymbol{w}$ at the same time as vector $\boldsymbol{p}$ has been augmented by $\boldsymbol{s}$ and $\boldsymbol{t}$. In the course of applying the simplex method surplus and slack variables are treated as the other $\boldsymbol{x}$ values. In the end they will be set equal to zero.

Also, note that in Table 5.2 all elements of $\boldsymbol{x}$ (*i. e.*, all unknown variables x_i) are not allowed to be negative. If the problem in general form requires that some variables x_i be allowed to take on negative values in addition to zero and positive values, we need to introduce free variables $q_i \geq 0$ and $r_i \geq 0$. We then let $x_i = q_i - r_i$ and replace all x_i accordingly. Additionally, all components of the vector $\boldsymbol{b}$ must be nonnegative. This can be easily achieved by multiplying with -1 both sides of the equal sign [Lue04].

The simplex method assumes that there are m independent columns in $\boldsymbol{A}$ (*i. e.*, the matrix $\boldsymbol{A}$ is of rank m), which also means that the columns are base vectors in the space $\mathbb{R}^m$. The m columns correspond to m components in $\boldsymbol{x}$. We denote by $\boldsymbol{B}$ the matrix containing the m independent columns of $\boldsymbol{A}$. Then we can uniquely solve the equation $\boldsymbol{By} = \boldsymbol{b}$ for a vector $\boldsymbol{y}$ with m components. If we let the m components in $\boldsymbol{x}$ assume the corresponding values in $\boldsymbol{y}$ and set the remaining $n - m$ components to zero, then $\boldsymbol{x}$ is a solution to the equation $\boldsymbol{Ax} = \boldsymbol{b}$. This is called a *basic solution* and the components of the vector $\boldsymbol{x}$ are called *basic variables*. If the basic solution lies within the feasible region of the problem in Table 5.2 then it is called a *basic feasible solution* [PS98, Lue04].

Geometrically, the feasible region of the problem described in Table 5.2 is always a *convex polytope*, also called *simplex*. For example, in two dimensions the simplex corresponds to a triangle, whereas in three dimensions it corresponds to a polyhedron (*i. e.*, a pyramid with triangular base). The vertices of the simplex constitute basic feasible solutions. For a problem with n variables and m constraints there are

$$\binom{n}{m} = \frac{n!}{m!(n-m)!} \tag{5.5}$$

feasible solutions. The simplex method attempts to find the optimal solution by searching the vertices of the polytope in an efficient manner [PS98].

The algorithm starts in one of the simplex vertices, which according to the discussion above corresponds to a basic feasible solution. There it evaluates how the objective function changes if it were to "move" into one of the $(n-m)$ neighboring vertices. The move is always along the edges of the simplex. When the algorithm has moved to a neighboring vertex one of the the m base vectors is replaced by one of the other $(n - m)$ vectors.

If we denote by $\boldsymbol{x}_k$ the feasible solution at step k in the simplex method and by $\boldsymbol{d}_k$ the direction along the edge to one of the neighboring vertices, then the rate of the change in the objective function due to moving along $\boldsymbol{d}_k$ is given by the directional derivative $D_{\boldsymbol{d}} f(\boldsymbol{x}_k) = \nabla f(\boldsymbol{x}_k)^T \boldsymbol{d} = \boldsymbol{c}^T \boldsymbol{d}$. The algorithm computes the $(n-m)$ directional derivatives and then selects a direction that provides a maximum decrease (or increase if we try to solve a maximization problem) in the objective function. The next step in the algorithm is to compute the feasible solution along the selected direction by the formula [LRV01]

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \sigma_k \boldsymbol{d}_k \tag{5.6}$$

where the step σ_k can be calculated from $\boldsymbol{d}_k$ and the current base vectors.

Before showing the complete algorithm we need to discuss the following issues:

i) how to find the $(n-m)$ directions,

ii) how to choose a step length σ_k,

iii) how to choose the initial vertex (*i. e.*, basic feasible solution) to start from.

We assume that the linear problem is already in standard form and that the matrix $\boldsymbol{A}$ contains m rows and n columns. For simplicity we assume that $x_1, \ldots, x_m$ are the basic variables. We rewrite the terms of the equation $\boldsymbol{Ax} = \boldsymbol{b}$ so that the basic variables are expressed in terms of nonbasic ones and of the $\boldsymbol{b}$ values.

$$\begin{aligned}
x_1 &= b_1 - a_{1,m+1}x_{m+1} - a_{1,m+2}x_{m+2} - \cdots - a_{1,n}x_n \\
x_2 &= b_2 - a_{2,m+1}x_{m+1} - a_{2,m+2}x_{m+2} - \cdots - a_{2,n}x_n \\
&\vdots \\
x_m &= b_m - a_{m,m+1}x_{m+1} - a_{m,m+2}x_{m+2} - \cdots - a_{m,n}x_n
\end{aligned} \tag{5.7}$$

Expressing the equations in this form is possible only when $m \leq n$ [PTVF02]. We can now read the $(n-m)$ direction vectors as the columns containing the a_{ij} coefficients. For example, the directions along the x_{m+1} axis and x_{m+2} axis respectively are the n-elements vectors

$$\boldsymbol{d}_{m+1} = \begin{bmatrix} -a_{1,m+1} \\ -a_{2,m+1} \\ \vdots \\ -a_{m,m+1} \\ 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \qquad \boldsymbol{d}_{m+2} = \begin{bmatrix} -a_{1,m+2} \\ -a_{2,m+2} \\ \vdots \\ -a_{m,m+2} \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{5.8}$$

Note that the vector component corresponding to the new direction (*i. e.*, the $(m+1)$ component of the vector $\boldsymbol{d}_{m+1}$ and the $(m+2)$ component of the vector $\boldsymbol{d}_{m+2}$) were set to one and the remaining $(n-(m+1))$ components are set to zero. The direction vectors are properly scaled by the step length σ_k, which is described next.

Once we have chosen a direction we have in essence decided which variable x_j will join the basis. Since we can only have m variables in a basic feasible solution we must decide which one of the old basis components will leave the basis. The obvious choice is to get rid of the component that limits the decrease the most (or increase in case of a maximum problem) in the objective function. Given a direction vector $\boldsymbol{d}$ along the axis x_j, we focus on the negative vector elements $\{d_i : d_i < 0\}$. For each such element we compute the ratio $x_i/-d_i$, where x_i is the corresponding element in the vector $\boldsymbol{x}$. We select the minimum ratio and the value i decides which element x_i leaves the basis [LRV01]. Incidentally, the ratio is also the step length σ_k that we were looking for:

$$\sigma_k = \min \frac{x_i}{-d_i} \text{ for all } i \text{ such that } d_i < 0 \tag{5.9}$$

In the case that no $d_i < 0$, then it is said that the solution is *unbounded*. This means that the simplex extends towards infinity along the axis x_j and consequently the objective function approaches $-\infty$ or $+\infty$.

Algorithm 9 summarizes the discussion so far. The only detail left for discussion is how to obtain the basic feasible solution required at the beginning of the algorithm. If all m equations contain slack variables then the initial basic feasible solution will consist entirely by slack variables. Otherwise one must consider the *auxiliary* optimization problem

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^{m} \alpha_i \\ \text{subject} & \boldsymbol{A}\boldsymbol{x} + \boldsymbol{\alpha} = \boldsymbol{b} \\ & \boldsymbol{x} \geq 0 \\ & \boldsymbol{\alpha} \geq 0 \end{array}$$

where $\boldsymbol{A}$, $\boldsymbol{x}$ and $\boldsymbol{b}$ are the original variables as in Table 5.2 (including slack variable, surplus variables and free variables) and $\boldsymbol{\alpha}$ is a vector of *artificial variables*. Artificial variables are used only for the equations that do not contain any slack variables. This problem has a initial basic solution $\boldsymbol{\alpha} = \boldsymbol{b}$. Consequently, it can be solved using Algorithm 9. If the algorithm finds a solution, then all elements of $\boldsymbol{a}$ equal zero due to the objective function $\sum_{i=1}^{m} \alpha_i$ and the constraint $\boldsymbol{\alpha} \geq 0$. The elements of $\boldsymbol{x}$ constitute a basic feasible solution that can be used to solve the original problem by using the simplex method. If no feasible solution is found then we cannot solve the original problem either. The *two-phase* simplex method is shown in Algorithm 8.

Require: linear optimization problem in standard form
1: *PHASE 1*
2: Add artificial variables $\boldsymbol{\alpha}$ to the original problem to construct the auxiliary problem
3: Solve the auxiliary problem using the simplex method described in Algorithm 9
4: **if** auxiliary problem cannot be solved **then**
5: **exit**
6: **end if**
7: *PHASE 2*
8: Let $\boldsymbol{x}_0$ equal the optimal solution to the auxiliary problem
9: Solve the original problem using the simplex method described in Algorithm 9

Algorithm 8: Two-phase simplex method (2-SM)

Require: basic feasible solution $\boldsymbol{x}_0$

1: $k \leftarrow 0$
2: **loop**
3: Compute the direction vector $\boldsymbol{d}_k^{(i)}$ for each direction i as shown in Eq. 5.7
4: **for all** direction vectors $\boldsymbol{d}_k^{(i)}$ **do**
5: $r_i \leftarrow \boldsymbol{c}^T \boldsymbol{d}_k^{(i)}$
6: **end for**
7: **if** minimization problem **then**
8: **if** all $r_i \geq 0$ **then**
9: $\boldsymbol{x}_k$ is the optimal solution
10: **return** $\boldsymbol{x}_k$
11: **else**
12: $m_j \leftarrow \min r_i$ for all $r_i < 0$
13: **end if**
14: **else if** maximization problem **then**
15: **if** all $r_i \leq 0$ **then**
16: $\boldsymbol{x}_k$ is the optimal solution
17: **exit**
18: **else**
19: $m_j \leftarrow \max r_i$ for all $r_i > 0$
20: **end if**
21: **end if**
22: The element x_j is joining basis
23: **if** $\boldsymbol{d}_k^{(i)}$ has no negative elements **then**
24: The solution is unbounded
25: **exit**
26: **end if**
27: $\sigma_k \leftarrow \min \frac{x_l}{-d_l}$ for all l such that $d_l < 0$ where d_l is an element of $\boldsymbol{d}_k^{(i)}$
28: The element x_l is leaving basis
29: $\boldsymbol{x}_{k+1} \leftarrow \boldsymbol{x}_k + \sigma_k \boldsymbol{d}_k^{(i)}$
30: $k = k + 1$
31: **end loop**

Algorithm 9: Simplex method (SM)

If there is no objective function in the standard form of a linear optimization problem, then we have a feasibility problem, expressed as a system of linear equations:

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \tag{5.10}$$

Such systems can be solved by Gauss elimination or LU-decomposition [GVL96, PTVF02], although neither one is discussed here. Additionally, if the matrix $\boldsymbol{A}$ is symmetric positive definite, then the conjugate gradient method discussed in Chapter 4 can be used to solve the system of linear equations as well.

Chapter 6

Gradient Projection Method

We consider here the gradient projection method (GPM) with the ability to solve the nonlinear problem with linear constraints

$$\begin{array}{ll} \text{minimize} & f(\boldsymbol{x}) \\ \text{subject to} & \boldsymbol{G}\boldsymbol{x} \leq \boldsymbol{b} \\ & \boldsymbol{H}\boldsymbol{x} = \boldsymbol{c} \\ & \boldsymbol{x} \in \Omega. \end{array}$$

We assume that the vector $\boldsymbol{x}$ has n elements and that $\boldsymbol{G}$ and $\boldsymbol{H}$ are matrices with $p \times n$ and $m \times n$ elements, respectively. The GPM can be extended to handle nonlinear constraints [Lue04], but that is not within the scope of this report.

The idea of a *working set* is fundamental to the GPM. The working set W consists of all or a subset of active constraints at a point $\boldsymbol{x}$. A constraint $G_i\boldsymbol{x}$ is said to be active at $\boldsymbol{x}$ if $G_i\boldsymbol{x} = 0$, as formally defined in Chapter 2.2. The use of a working set implies that at each feasible point $\boldsymbol{x}$ we are in fact considering the problem with equality constraints only

$$\begin{array}{ll} \text{minimize} & f(\boldsymbol{x}) \\ \text{subject to} & \boldsymbol{A}_q\boldsymbol{x} = \boldsymbol{b} \\ & \boldsymbol{x} \in \Omega \end{array}$$

where $\boldsymbol{A}_q\boldsymbol{x}$ denote the set of q active constraints. Additionally, this means that the KKT-conditions apply at $\boldsymbol{x}$.

The working set constraints define a surface across which we have to search the next feasible point. This is called the *working surface*. In the GPM, the direction $\boldsymbol{d}_k$ in which we search for the next feasible point $\boldsymbol{x}_{k+1}$ is given by the projection of the negative gradient on the working space. This projection is provided by the *projection matrix*

$$\boldsymbol{P}_k = \left[\boldsymbol{I} - \boldsymbol{A}_q^T(\boldsymbol{A}_q\boldsymbol{A}_{\boldsymbol{q}}^{\boldsymbol{T}})^{-1}\boldsymbol{A}_q\right]. \tag{6.1}$$

The new direction is computed by multiplying the projection matrix with the gradient $\boldsymbol{d_k} = \boldsymbol{P}_k\nabla f(\boldsymbol{x}_k)$ [Lue04].

When a direction $\boldsymbol{d}_k$ is found, the next issue is finding the step length σ_k. We can do this by using a line search algorithm. However, in this case we are not interested in searching along the whole line. Instead, we want to find the longest line segment $\boldsymbol{x}_k + \sigma_k\boldsymbol{d}_k$ that can be contained within the constraints. We can do that in two phases. During phase one we find

the maximum number α such that $\boldsymbol{x}_k + \alpha \boldsymbol{d}_k$ is feasible. Then we can proceed with phase two, which gives the step length σ_k that minimizes $f(\boldsymbol{x}_k + \sigma_k \boldsymbol{d}_k)$, where $0 \leq \sigma_k \leq \alpha$.

If $\boldsymbol{d} = \boldsymbol{0}$ we have either found the optimal solution or the algorithm is prevented by the constraints $\boldsymbol{A}_q$ from going any further.

To test if we have found an optimal solution we compute the Lagrange multipliers $\boldsymbol{\lambda} = -(\boldsymbol{A}_q \boldsymbol{A}_q^T)^{-1} \boldsymbol{A}_q \nabla f(\boldsymbol{x}_k)$. If all $\lambda_j \geq 0$ for all j corresponding to active inequalities, then we have indeed found the optimum [Lue04].

If on the other hand one or more λ_j are negative, then we need to remove one constraint from the working set so that the algorithm can advance to the next feasible point. To do that we remove from $\boldsymbol{A}_q$ row j corresponding to the smallest (most negative) λ_j.

The complete method is shown in Algorithm 10. A feasible solution $\boldsymbol{x}_0$ is required to start the algorithm. This can be either identified in the problem specification or through a procedure similar to *PHASE 1* of the simplex method described in Chapter 5.

Require: $\boldsymbol{x}_0$ such that it is a feasible point (*e. g.*, use *PHASE 1* of the simplex method)
1: Let the rows of $\boldsymbol{A}_q$ represent the q active constraints
2: **repeat**
3: $\quad \boldsymbol{P} \leftarrow \left[\boldsymbol{I} - \boldsymbol{A}_q^T (\boldsymbol{A}_q \boldsymbol{A}_q^T)^{-1}\right]$
4: $\quad \boldsymbol{d} \leftarrow -\boldsymbol{P} \nabla f(\boldsymbol{x})^T$
5: $\quad$ **if** $\boldsymbol{d} \neq 0$ **then**
6: $\quad\quad \alpha \leftarrow \max\{\alpha : (\boldsymbol{x} + \alpha \boldsymbol{d})$ is feasible$\}$
7: $\quad\quad$ Line search: find σ_k that minimizes $f(\boldsymbol{x}_k + \sigma_k \boldsymbol{d}_k)$, such that $0 \leq \sigma_k \leq \alpha$
8: $\quad\quad \boldsymbol{x} \leftarrow \boldsymbol{x} + \sigma_k \boldsymbol{d}$
9: $\quad$ **else**
10: $\quad\quad \boldsymbol{\lambda} \leftarrow -(\boldsymbol{A}_q \boldsymbol{A}_q^T)^{-1} \boldsymbol{A}_q \nabla f(\boldsymbol{x})^T$
11: $\quad\quad$ **if** $\lambda_j \geq 0$ for all j corresponding to active inequalities **then**
12: $\quad\quad\quad$ **return** $\boldsymbol{x}$ since it satisfies the KKT-conditions.
13: $\quad\quad$ **else**
14: $\quad\quad\quad$ delete row j from $\boldsymbol{A}_q$ corresponding to the smallest λ_j
15: $\quad\quad$ **end if**
16: $\quad$ **end if**
17: **until** stopping condition is satisfied

Algorithm 10: Gradient projection method (GPM)

If the optimization problem has no constraints other than that all solutions $\boldsymbol{x}$ are restricted to the positive orthant, then there is a more efficient version of the GPM, as described in [BG91].

Chapter 7

Particle Swarm Optimization

Particle swarm optimization (PSO) is a metaheuristic optimization method. The method is based on the swarm intelligence displayed by bird flocking behavior. In this case the swarm consists of particles. The term particle is used to denote that individuals in the swarm have velocity and acceleration, although they lack mass and volume. Each individual particle "flows" through the multidimensional search space having its position and velocity influenced by neighboring particles as well as by a random component. The general PSO method is used to solve unconstrained optimization problems of the form

$$\begin{array}{ll} \text{minimize} & f(\boldsymbol{x}) \\ \text{subject to} & \boldsymbol{x} \in \mathbb{R}^n \end{array}$$

We assume that each vector $\boldsymbol{x}$ consists of n elements and we denote by m the number of particles in a swarm (*i. e.*, the swarm size). The typical size of the swarm is 10–30 particles [Eng06].

In order to cope with particle identification we overload the variable notation used so far. We write $\boldsymbol{x}_i$ to specify the position in $\mathbb{R}^n$ for particle i and we denote by x_{ij} the position of particle i along the jth coordinate axis. Note that $\boldsymbol{x}_j$ is a vector and x_{ij} is a scalar. Additionally, if we want to specify their value at a specific step k in the algorithm, we write $\boldsymbol{x}_i^{(k)}$ and $x_{ij}^{(k)}$.

For each particle i there is a *personal best position* variable

$$\boldsymbol{y}_i^{(k+1)} = \begin{cases} \boldsymbol{y}_i^{(k)} & \text{if } f\left(\boldsymbol{x}_i^{(k+1)}\right) \geq f\left(\boldsymbol{y}_i^{(k)}\right) \\ \boldsymbol{x}_i^{(k+1)} & \text{otherwise} \end{cases} \tag{7.1}$$

being maintained. For a maximization problem the first condition is reversed:

$$f\left(\boldsymbol{x}_i^{(k+1)}\right) \leq f\left(\boldsymbol{y}_i^{(k)}\right). \tag{7.2}$$

In addition, the method must maintain a *global best position*, $\hat{\boldsymbol{y}}^{(k)}$, or a *local best position*, $\hat{\boldsymbol{y}}_i^{(k)}$.

The global best PSO (GB-PSO) method computes $\hat{\boldsymbol{y}}^{(k)}$ be considering the personal best position of all particles in the swarm, such that

$$\hat{\boldsymbol{y}}^{(k)} = \boldsymbol{y}_i^{(k)} \text{ such that } f\left(\boldsymbol{y}_i^{(k)}\right) = \min\left(f\left(\boldsymbol{y}_0^{(k)}\right), \ldots, f\left(\boldsymbol{y}_m^{(k)}\right)\right). \tag{7.3}$$

In the case of the local best PSO (LB-PSO), for each particle i, a neighborhood $\mathcal{N}_i$ of size η must be defined. The neighborhood is defined as [Eng06].

$$\mathcal{N}_i = \{\boldsymbol{y}_{i-\eta}^{(k)}, \boldsymbol{y}_{i-\eta+1}^{(k)}, \ldots, \boldsymbol{y}_{i-1}^{(k)}, \boldsymbol{y}_i^{(k)}, \boldsymbol{y}_{i+1}^{(k)}, \ldots, \boldsymbol{y}_{i+\eta}^{(k)}\} \tag{7.4}$$

The local best position $\hat{\boldsymbol{y}}_i^{(k)}$ is the best position within the neighborhood

$$\hat{\boldsymbol{y}}_i^{(k)} = \boldsymbol{y}_i^{(k)} \in \mathcal{N}_i \text{ such that } f\left(\hat{\boldsymbol{y}}_i^{(k)}\right) = \min f\left(\boldsymbol{x}\right),\ \forall \boldsymbol{x} \in \mathcal{N}_i \tag{7.5}$$

The position of a particle $x_{ij}^{(k)}$ is influenced by a *cognitive* component $C_j^{(k)}$ and by a *social* component $S_j^{(k)}$. Both components are stochastic variables drawn from the standard unit distribution $U(0,1)$. This fact is indicated in the complete algorithm by writing $C_j^{(k)} \overset{d}{=} U(0,1)$ and $S_j^{(k)} \overset{d}{=} U(0,1)$, respectively. C_j and S_j are scaled by the positive scalars c and s. The scalars are called *acceleration constants* and their main use is to establish a sort of ratio between the cognitive and social component. The value of the acceleration constants is static and typically decided by empirical studies [Eng06].

The cognitive and social components of a particle are combined into a *velocity* component. This is done depending upon the radius of social network used for particles. In the case of GB-PSO the social network covers the entire swarm and the velocity component is

$$v_{ij}^{(k)} = v_{ij}^{(k)} + cC_j^{(k)}\left(y_{ij}^{(k)} - x_{ij}^{(k)}\right) + sS_j^{(k)}\left(\hat{y}_j^{(k)} - x_{ij}^{(k)}\right) \tag{7.6}$$

On the other hand, for the LB-PSO the social network is confined to the neighborhood $\mathcal{N}_i$ of particle i and the velocity component is calculated as

$$v_{ij}^{(k)} = v_{ij}^{(k)} + cC_j^{(k)}\left(y_{ij}^{(k)} - x_{ij}^{(k)}\right) + sS_j^{(k)}\left(\hat{y}_{ij}^{(k)} - x_{ij}^{(k)}\right). \tag{7.7}$$

Using the velocity component the position of particle can be updated as

$$\boldsymbol{x}_i^{(k+1)} = \boldsymbol{x}_i^{(k)} + \boldsymbol{v}_i^{(k+1)}. \tag{7.8}$$

The initial position $\boldsymbol{x}_i^{(0)}$ can be sampled from a uniform distribution $U(\boldsymbol{x}_{min}, \boldsymbol{x}_{max})$, where $\boldsymbol{x}_{min}$ and $\boldsymbol{x}_{max}$ contain the minimum and maximum position along each coordinate in $\mathbb{R}^n$.

Algorithm 11 and Algorithm 12 summarize these two PSO algorithms. In terms of efficiency, the difference between GB-PSO and LB-PSO is that GB-PSO converges faster than LB-PSO but is more likely to be trapped in local minima. The stopping condition is one of those described in Chapter 2.5. Additional termination criteria are discussed in [Eng06].

Constrained problems that can be converted to unconstrained problems can also be solved with PSO methods. Another possibility to solve constrained problems is to reject infeasible solutions by not allowing them to be recorded as personal best, local best and global best positions. Additionally, one can generate new random positions within the feasible space. In [Eng06] there are several other methods presented that can extend the PSO methods to handle constrained optimization problems.

1: $\boldsymbol{x}_i^{(0)} \overset{d}{=} U\left(\boldsymbol{x}_{min}, \boldsymbol{x}_{max}\right)$ for all $i = 1, \ldots, m$
2: $\boldsymbol{y}_i^{(0)} \leftarrow \boldsymbol{x}_i^{(0)}$ for all $i = 1, \ldots, m$
3: $\hat{\boldsymbol{y}}_i^{(0)} \leftarrow \boldsymbol{x}_i^{(0)}$ for all $i = 1, \ldots, m$
4: **repeat**
5: $C_j^{(k)} \overset{d}{=} U(0,1)$ for all $j = 1, \ldots, n$
6: $S_j^{(k)} \overset{d}{=} U(0,1)$ for all $j = 1, \ldots, n$
7: **for** each particle $i = 1, \ldots, m$ **do**
8: **if** $f\left(\boldsymbol{x}_i\right) < f\left(\boldsymbol{y}_i\right)$ **then**
9: $\boldsymbol{y}_i \leftarrow \boldsymbol{x}_i$
10: **end if**
11: **if** $f\left(\boldsymbol{y}_i\right) < f\left(\hat{\boldsymbol{y}}\right)$ **then**
12: $\hat{\boldsymbol{y}} \leftarrow \boldsymbol{y}$
13: **end if**
14: **end for**
15: **for** each particle $i = 1, \ldots, m$ **do**
16: **for** each dimension $j = 1, \ldots, n$ **do**
17: $v_{ij}^{(k)} \leftarrow v_{ij}^{(k)} + cC_j^{(k)}\left(y_{ij}^{(k)} - x_{ij}^{(k)}\right) + sS_j^{(k)}\left(\hat{y}_j^{(k)} - x_{ij}^{(k)}\right)$
18: $\boldsymbol{x}_i^{(k+1)} \leftarrow \boldsymbol{x}_i^{(k)} + \boldsymbol{v}_i^{(k+1)}$
19: $\boldsymbol{y}_i^{(k+1)} \leftarrow \begin{cases} \boldsymbol{y}_i^{(k)} & \text{if } f\left(\boldsymbol{x}_i^{(k+1)}\right) \geq f\left(\boldsymbol{y}_i^{(k)}\right) \\ \boldsymbol{x}_i^{(k+1)} & \text{otherwise} \end{cases}$
20: $\hat{\boldsymbol{y}}^{(k)} \leftarrow \boldsymbol{y}_i^{(k)}$ such that $f\left(\boldsymbol{y}_i^{(k)}\right) = \min\left(f\left(\boldsymbol{y}_0^{(k)}\right), \ldots, f\left(\boldsymbol{y}_m^{(k)}\right)\right)$
21: **end for**
22: **end for**
23: **until** stopping condition is satisfied

Algorithm 11: Global best particle swarm optimization (GB-PSO)

$\boldsymbol{x}_i^{(0)} \overset{d}{=} U(\boldsymbol{x}_{min}, \boldsymbol{x}_{max})$ for all $i = 1, \ldots, m$
$\boldsymbol{y}_i^{(0)} \leftarrow \boldsymbol{x}_i^{(0)}$ for all $i = 1, \ldots, m$
$\hat{\boldsymbol{y}}_i^{(0)} \leftarrow \boldsymbol{x}_i^{(0)}$ for all $i = 1, \ldots, m$
repeat
  $C_j^{(k)} \overset{d}{=} U(0,1)$ for all $j = 1, \ldots, n$
  $S_j^{(k)} \overset{d}{=} U(0,1)$ for all $j = 1, \ldots, n$
  for each particle $i = 1, \ldots, m$ **do**
    if $f(\boldsymbol{x}_i) < f(\boldsymbol{y}_i)$ **then**
      $\boldsymbol{y}_i \leftarrow \boldsymbol{x}_i$
    end if
    if $f(\boldsymbol{y}_i) < f(\hat{\boldsymbol{y}})$ **then**
      $\hat{\boldsymbol{y}} \leftarrow \boldsymbol{y}$
    end if
  end for
  for each particle $i = 1, \ldots, m$ **do**
    for each dimension $j = 1, \ldots, n$ **do**
      $v_{ij}^{(k)} \leftarrow v_{ij}^{(k)} + cC_j^{(k)}\left(y_{ij}^{(k)} - x_{ij}^{(k)}\right) + sS_j^{(k)}\left(\hat{y}_{ij}^{(k)} - x_{ij}^{(k)}\right)$
      $\boldsymbol{x}_i^{(k+1)} \leftarrow \boldsymbol{x}_i^{(k)} + \boldsymbol{v}_i^{(k+1)}$
      $\boldsymbol{y}_i^{(k+1)} \leftarrow \begin{cases} \boldsymbol{y}_i^{(k)} & \text{if } f\left(\boldsymbol{x}_i^{(k+1)}\right) \geq f\left(\boldsymbol{y}_i^{(k)}\right) \\ \boldsymbol{x}_i^{(k+1)} & \text{otherwise} \end{cases}$
      $\hat{\boldsymbol{y}}_i^{(k)} \leftarrow \boldsymbol{y}_i^{(k)} \in \mathcal{N}_i$ such that $f\left(\hat{\boldsymbol{y}}_i^{(k)}\right) = \min f(\boldsymbol{x}),\ \forall \boldsymbol{x} \in \mathcal{N}_i$
    end for
  end for
until stopping condition is satisfied

Algorithm 12: Local best particle swarm optimization (LB-PSO)

Chapter 8

SAMCRA

Self-Adaptive Multiple Constraints Routing Algorithm (SAMCRA) is a heuristic that is able to solve efficiently MCP and MCOP problems with additive metrics. The algorithm relies on the following four key concepts [VMK04, KKKVM04]:

i) non-linear path length definition,

ii) k-shortest path computation,

iii) principle of path dominance,

iv) look-ahead concept.

For a path P with a set of m additive metrics (weights) $\boldsymbol{w}(P)$ and m constraints $\boldsymbol{L}$, SAMCRA collapses all metrics into the *path length* metric

$$\Lambda(P) = \left\| \frac{\boldsymbol{w}}{\boldsymbol{L}} \right\|_{\infty} = \max_{1 \leq i \leq m} \frac{w_i(P)}{L_i}. \tag{8.1}$$

The purpose of the non-linear path length is to obtain a better match of the boundaries of the constraints when searching the feasible space. Since each path is represented by a single metric, one is tempted to apply Dijkstra's algorithm to solve the problem. However, Dijkstra's algorithm relies on the property that subsections of the shortest path are also shortest paths. This property does not hold in this case due to the non-linear path length definition [VMK04].

SAMCRA attempts to alleviate this problem by its second key concept, which is the k-shortest path computation. At each intermediate node on the path the algorithm stores the shortest path found, the second shortest, up to the kth shortest. The value of k is decided by the remaining two key concepts.

The principle of path dominance says that a path P_1 from the source node to some intermediate node is *dominated* if there is another path P_2 for which at least one metric i is $w_i(P_2) > w_i(P_1)$ and all other j metrics are $w_j(P_2) \geq w_j(P_1)$, $\forall j \neq i$. If there is no such path P_2 then the path P_1 is called *nondominated*. SAMCRA discards dominated paths.

SAMCRA's look-ahead concept revolves around computing the shortest path from the *destination* node to each node n in the graph $G(V, E)$, for each of the m metrics. This is done during the initialization phase of SAMCRA and implies that Dijkstra's algorithm is run m times. We denote by A the source node, by B the destination node and by n any other node in the graph. The look-ahead computes a bound $b_i^n = w_i\left(P^*(n, B)\right)$ for metric i on

the shortest path $P^*(n, B)$ (on metric w_i) between n and B. The usefulness of b_i becomes apparent when one considers the situation that SAMCRA has just computed the path from the source node A to the intermediate node n. At that point, the algorithm knows the values of $w_i\left(P(A,n)\right)$ and b_i for all metrics i. Therefore, it can check if the inequality

$$w_i\left(P(A,n)\right) + b_i(n) \leq L_i \tag{8.2}$$

holds for $i = 1, \ldots, m$. If the inequality does not hold for at least one i, then the path from A to B over the intermediate node n can be ignored.

If the inequality in Eq. 8.2 holds then one can compute the expected path length over an intermediate node n using Eq. 8.1. This value is an upper bound on the expected path length for all paths connecting A and B. In SAMCRA's algorithm shown below, this value is stored in the variable MAX and any path with a length greater than MAX is discarded [VMK04].

We have tried to keep SAMCRA's pseudo-code structured the same way as in [VMK04, Kui04]: main function shown in Algorithm 13, initialization routine displayed in Algorithm 14, feasibility function in Algorithm 15 and update queue function shown in Algorithm 16. The functions INSERT, EXTRACT_MIN and DECREASE_KEY are explained in depth in [CLR01]. Here we mention only the effect of calling them. Table 8.1 contains the variable definitions. SAMCRA requires as input the graph $G(V, E)$, the number of link metrics (weights) m, the source and destination nodes A and B, and the constraint vector $\boldsymbol{L}$.

```
Require: G, m, A, B, L
 1: b ← call INITIALIZE(G, m, A, B)
 2: while Q ≠ ∅ do
 3:    [u, c_u, u_i, Λ(u_i)] ← call EXTRACT_MIN(Q)
 4:    if u_i ≠ NIL then
 5:       u_i ← GRAY
 6:    end if
 7:    if u = B then
 8:       DONE: return u_i
 9:    else
10:       for all nodes v adjacent to u, except π(u_i) and A do
11:          D ← call FEASIBILITY(G, u, i, v, c, d, w, MAX)
12:          PLEN ← Λ(d(u_i) + w(u → v) + b_v
13:          if PLEN ≤ MAX and D = FALSE then
14:             call UPDATEQUEUE(G, u, i, v, j, c, d, w, π, PLEN)
15:             if v = B and PLEN < MAX then
16:                MAX ← PLEN
17:             end if
18:          end if
19:       end for
20:    end if
21: end while
```

Algorithm 13: Self-Adaptive Multiple Constraints Routing Algorithm (SAMCRA)

Line 1 of Algorithm 13 call the initialization function. For each node in the graph, the function clears the number of stored paths and then it stores the theoretical maximum path length in the variable MAX. The INITIALIZE function calls the DIJKSTRA function, which

Variable	Definition
V	set of vertices (nodes)
E	set of edges (links)
G	graph $G(V, E)$
m	number link weights considered in this problem
A	source node on the path to be computed
B	destination node on the path to be computed
$\boldsymbol{L} = [L_1, \ldots, L_m]$	constraint values vector
$\Lambda(P)$	the nonlinear length of (sub)path P
u, v	intermediate nodes
$\boldsymbol{c} = [c_1, \ldots, c_N]$	counter for the number of paths stored at each node, where N is the number of nodes in V
MAX	maximum length that a (sub)path may have
$b_i^n = w_i\left(P_i^*(n, B)\right)$	lower bound for weight i on the shortest path for weight i between some node n and node B
$\boldsymbol{w}(P) = [w_1(P), \ldots, w_m(P)]$	weight vector for the path P, such that $w_i(P) \leq L_i$
$\boldsymbol{w}(u \rightarrow v)$	weight vector for the link connecting node u and v
$\boldsymbol{Q}$	queue storing the nodes on the path from A to B
$u_i = P_i(A, u)$	ith path from A to u stored in $\boldsymbol{Q}$ in the entry for node u
$v_i = P_i(A, v)$	ith path from A to v stored in $\boldsymbol{Q}$ in the entry for node v
$\boldsymbol{\pi}$	predecessor list
$\boldsymbol{\pi}\left(u_i\right) = \boldsymbol{\pi}\left(P_i(A, u)\right)$	predecessor node on the ith path from A to u
$\mathcal{D}$	boolean variable denoting if the (sub)path is dominated or not
$\boldsymbol{b}_n = \left[b_1^n \ldots b_m^n\right]$	lower bounds vector
$\boldsymbol{d}(u_i) = \boldsymbol{w}\left(P(A, u)\right)$	subpath weight vector for u_i
$PLEN$	predicted length variable

Table 8.1: Variable definitions for SAMCRA

calculates for each i the bounds b_i for all nodes $n \in V$ and also the shortest paths from A to B. If any path has an expected length smaller than one, then that path length is stored in MAX. Line 11–12 clear the queue variable Q and update the number of paths stored at node A. The queue Q stores entries of the type `[node, number of stored subpaths, list of stored subpaths` u_i`, path length for` b_i `bounds]`. The entries are sorted by minimum path length u_i. On line 13 we inserted node A into the queue. Since we have no subpaths computed for A, the corresponding variable in the entry is marked NIL. Line 14 returns the vector $\boldsymbol{b}$ with bounds to the main function, where an integrative process is started on line 2.

Require: G, m, A, B
1: **for all** $v \in E$ **do**
2: $\quad c_v = 0$
3: **end for**
4: $MAX = 1.0$
5: **for** $i = 1, \ldots, m$ **do**
6: $\quad [b_i(n), P_i^*(A,B)] \leftarrow$ **call** DIJKSTRA(G, A, B, i)
7: $\quad$ **if** $\Lambda\left(P_i^*(A,B)\right) < MAX$ **then**
8: $\quad\quad MAX = \Lambda\left(P_i^*(A,B)\right)$
9: $\quad$ **end if**
10: **end for**
11: Clear queue: $\boldsymbol{Q} \leftarrow \emptyset$
12: $c_A \leftarrow 1$
13: **call** INSERT$(\boldsymbol{Q}, A, c_A, NIL, \Lambda\left(\boldsymbol{b}_A\right))$
14: **return** $\boldsymbol{b}$

Algorithm 14: INITIALIZE

Line 3 in the main function extracts from Q the entry with the shortest subpath u_i. SAMCRA uses a color scheme for subpaths: new (undiscovered paths) are considered to be white, discovered paths are colored grey and discarded paths are colored black. This explains line 4–6. If the destination node u on the extracted subpath u_i is node B then the algorithm has successfully found the optimal path. Otherwise, it attempts to extend the subpath to one of node u's neighbors, with the exception of the previous node on the path and node A (to avoid loops). For each of these nodes SAMCRA calls the FEASIBILITY function.

Require: $G, u, i, v, \boldsymbol{c}, \boldsymbol{d}, \boldsymbol{w}, MAX$
1: **for** $j = 1, \ldots, c_v$ **do**
2: $\quad$ **if** $\left(\boldsymbol{d}(u_i) + \boldsymbol{w}(u \rightarrow v)\right) - \boldsymbol{d}(v_j) \leq \boldsymbol{0}$ or $\Lambda\left(\boldsymbol{d}(v_j)\right) > MAX$ **then**
3: $\quad\quad v_j \leftarrow$ `BLACK`
4: $\quad\quad$ **return** `FALSE`
5: $\quad$ **else if** $\boldsymbol{d}(v_j) - \left(\boldsymbol{d}(u_i) + \boldsymbol{w}(u \rightarrow v)\right) \leq \boldsymbol{0}$ **then**
6: $\quad\quad$ **return** `TRUE`
7: $\quad$ **end if**
8: **end for**

Algorithm 15: FEASIBILITY

Given node u's neighbor v, the FEASIBILITY function checks the length of the extended path $A \rightarrow u \rightarrow v$ against each subpath $A \rightarrow v$ stored at node v. If the subpath is longer than

the extended path or longer than MAX the subpath is colored black and is not considered anymore. If this is not the case, line 5 checks if the extended path is dominated by the subpath v_j. The FEASIBILITY function returns TRUE if the extended subpath is dominated and false otherwise.

Line 12 in the main function computes the predicted length $PLEN$ for the extended subpath. If the predicted length does not exceed MAX and the extended subpath is not dominated, then the function UPDATEQUEUE is called.

Require: $G, u, i, v, j, \boldsymbol{c}, \boldsymbol{d}, \boldsymbol{w}, \boldsymbol{\pi}, PLEN$
1: **for** $j = 1, \ldots, c_v$ **do**
2: **if** $v_j = BLACK$ and $\Lambda(\boldsymbol{d}(v_j + b_v)) > PLEN$ **then**
3: **call** DECREASE_KEY($\boldsymbol{Q}, v, j, PLEN$)
4: $\boldsymbol{d}(v_j) \leftarrow \boldsymbol{d}(u_i) + \boldsymbol{w}(u \rightarrow v)$
5: $\boldsymbol{\pi}(v_j) \leftarrow u_i$
6: **return**
7: **end if**
8: **end for**
9: $\boldsymbol{c}_v \leftarrow \boldsymbol{c} + 1$
10: **call** INSERT($Q, v, c_v, u_v, PLEN$)
11: $k \leftarrow c_v$
12: $\boldsymbol{d}(v_k) \leftarrow \boldsymbol{d}(u_i) + \boldsymbol{w}(u \rightarrow v)$
13: $\boldsymbol{\pi}(v_k) \leftarrow u_i$

Algorithm 16: UPDATEQUEUE

Line 2 of the UPDATEQUEUE method checks if any black path exists with predicted length greater than the predicted length $PLEN$ of the extended path. If this is the case, then DECREASE_KEY updates the predicted length of path v_j in node v's entry with the value PLEN. Line 4–6 update the subpath weight vector, the predecessor list of the path v_j and then cause the function to return. If the condition on line 2 in UPDATEQUEUE is not met by any path v_j, then line 9-13 updates the subpath count, inserts the subpath into Q's entry for v and updates the weight vector and predecessor list. Line 15-16 in the Algorithm 13 updates the MAX variable if necessary.

Chapter 9

Final Remarks

The last few years the Internet has witnessed a tremendous growth in terms of multimedia-based service. These services, with emphasis on those being served live, demand better QoS than that provided by IP's best-effort service.

Although two major QoS architectures have been designed, none of them has been widely deployed. The last few years new research has uncovered a new class of QoS architecture based on overlay networks. The ROVER project at BTH in Karlskrona, Sweden is concerned with the research and development of components for overlay-based QoS architectures. QoS routing is one such component. Briefly stated, the mission of QoS routing is to find and to manage optimal network paths subject to various QoS constraints. In this report we have surveyed a number of optimization algorithms that can be used for this purpose.

We started by presenting the theoretical foundation of optimization theory. Applied optimization theory lies at the intersection of several disciplines: calculus, linear algebra, graph theory, computation and complexity theory. For each discipline we tried to provide the minimum amount of theory required by concepts in optimization theory.

In terms of optimization algorithms we presented a mixture of numerical methods, a heuristic and a metaheuristic method. We also discussed briefly the applicability of each method to routing optimization problems.

The future work is to implement these methods on top of a library of numerical methods such as GNU Scientific Library (GSL) [GDT+06] and test their performance in solving problems on different network topologies. A plan for the longer term is to integrate some of these optimization algorithms with routing protocols in overlay networks.

Appendix A

Acronyms

API	application programming interface
AS	autonoumous system
BGP	Border Gateway Protocol
BTH	Blekinge Institute of Technology
CGM	conjugate gradient method
DiffServ	Differentiated Services
FEC	forward error correction
FR-CGM	Fletcher-Reeves CGM
GDM	general descent method
GPM	gradient projection method
GB-PSO	global best PSO
GSL	GNU Scientific Library
IP	Internet Protocol
IntServ	Integrated Services
KKT	Karush-Kuhn-Tucker
LB-PSO	local best PSO
MCP	multi-constrained path
MCOP	multi-constrained optimal path
QoS	quality of service
ORP	Overlay Routing Protocol
OSPF	Open Shortest Path First
PR-CGM	Polak-Ribiere CGM
PSO	particle swarm optimization
Q-CGM	CGM with quadratic approximation
RDP	Route Discovery Protocol
RIP	Routing Information Protocol
RMP	Route Management Protocol
ROVER	Routing in Overlay Networks
RSVP	Resource Reservation Protocol
RTT	round-trip time
SAMCRA	Self-Adaptive Multiple Constraints Routing Algorithm
SDM	steepest descent method
UDP	User Datagram Protocol
UUID	universally unique identifier
VoIP	voice over IP

Bibliography

[AMO93] Ravindra K Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows: Theories, Algorithms, and Applications*. Prentice Hall, Upper Saddle River, NJ, USA, 1993. ISBN: 0-13-617549-X.

[And01] David G. Andersen. Resilient overlay networks. Master's thesis, Dept. of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, May 2001.

[Arm03] Grenville J. Armitage. Revisiting IP QOS. *ACM SIGCOMM Computer Communications Review*, 33(5):81–88, October 2003.

[BBC+98] Steven Blake, David L. Black, Mark A. Carlson, Elwyn Davies, Zheng Wang, and Walter Weiss. *RFC 2475: An Architecture for Differentiated Services*. IETF, December 1998. Category: Informational.

[BCS94] Robert Braden, David D. Clark, and Scott Shenker. *RFC 1633: Integrated Services in the Internet Architecture: an Overview*. IETF, June 1994. Category: Informational.

[BDH+03] L. Burgsthaler, K. Dolzer, C. Hauser, J. Jähnert, S. Junghans, C. Macián, and W. Payer. Beyond technology: The missing pieces for QoS success. In *Proceedings ot the ACM SIGCOMM Workshops*, pages 121–130, Karlsruhe, Germany, August 2003.

[Bel03] Gregory Bell. Failure to thrive: QoS and the culture of operational networking. In *Proceedings ot the ACM SIGCOMM Workshops*, pages 115–120, Karlsruhe, Germany, August 2003.

[BG91] Dimitri P. Bertsekas and Robert G. Gallager. *Data Networks*. Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 1991. ISBN: 0-13-200916-1.

[BR03] Christian Blum and Andrea Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308, September 2003.

[BT97] Dimitri P. Bertsekas and John N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods*. Athena Scientific, Belmont, MA, USA, 1997. ISBN: 1-886529-01-9.

[BV04] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, Cambridge, UK, 2004. ISBN: 0-521-83378-7.

[CFSK04] Byung-Gon Chun, Rodrigo Fonseca, Ion Stoica, and John Kubiatowicz. Characterizing selfishly constructed overlay routing networks. In *Proceedings of INFOCOM*, volume 2, pages 1329–1339, Hong Kong, China, March 2004.

[CHM+03] Jon Crowcroft, Steven Hand, Richard Mortier, Timothy Roscoe, and Andrew Warfield. QoS's downfall: At the bottom, or not at all! In *Proceedings of the ACM SIGCOMM Workshops*, pages 109–114, Karlsruhe, Germany, August 2003.

[CLR01] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. The MIT Press, Cambridge, MA, USA, 2nd edition, 2001. ISBN: 0-262-53196-8.

[Coo83] Stephen A. Cook. An overview of computational complexity. *Communications of the ACM*, 26(6):400–408, June 1983.

[DF07] Yezid Donoso and Ramon Fabregat. *Multi-Objective Optimization in Computer Networks Using Metaheuristics*. Auerbach Publications, Boca Raton, FL, USA, 2007. ISBN: 0-8493-8084-7.

[DV07] Karel De Vogeleer. QoS routing in overlay networks. Master's thesis, Blekinge Institute of Technology (BTH), Karlskrona, Sweden, June 2007. MEE07:24.

[Edw94] Charles H. Edwards, Jr. *Advanced Calculus of Several Variables*. Dover Publications, Mineola, NY, USA, 1994. ISBN: 0-486-68336-2.

[Eng06] Andries P. Engelbrecht. *Fundamentals of Computational Swarm Intelligence*. John Wiley & Sons, Chichester, West Sussex, England, 2006. ISBN: 0-470-09191-6.

[FML+03] Chuck Fraleigh, Sue Moon, Bryan Lyles, Chase Cotton, Mujahid Khan, Deb Moll, Rob Rockell, Ted Seely, and Christophe Diot. Packet-level traffic measurements from the Sprint IP backbone. *IEEE Network*, 17(6):6–16, November 2003.

[GDT+06] Mark Galassi, Jim Davies, James Theiler, Brian Gough, Gerard Jungman, Michael Booth, and Fabrice Rossi. *GNU Scientific Library Reference Manual*. Network Theory Limited, Bristol, UK, 2nd edition, 2006. ISBN: 0-9541617-3-4.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completness*. W. H. Freeman and Company, New York, NY, USA, 1979. ISBN: 0-7167-1045-5.

[GO97] Roch Guérin and Ariel Orda. QoS-based routing in networks with inaccurate information: Theory and algorithms. In *Proceedings of INFOCOM*, volume 1, pages 75–83, Kobe, Japan, April 1997.

[GVL96] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, MD, USA, 3rd edition, 1996. ISBN: 0-8018-5414-8.

[Ham86] Richard W. Hamming. *Numerical Methods for Scientists and Engineers*. Dover Publications, Mineola, NY, USA, 1986. ISBN: 0-486-65241-6.

[HS52] Magnus R. Hestenes and Eduard Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, 49(6):409–436, December 1952.

[Hui00] Christian Huitema. *Routing in the Internet*. Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 2000. ISBN: 0-13-022647-5.

[Ili06] Dragos Ilie. *Gnutella Network Traffic: Measurements and Characteristics*. Licentiate dissertation, Blekinge Institute of Technology (BTH), Karlskrona, Sweden, April 2006. ISBN: 91-7295-084-6.

[IP07] Dragos Ilie and Adrian Popescu. A framework for overlay QoS routing. In *Proceedings of 4th Euro-FGI Workshop*, Ghent, Belgium, May 2007.

[KF75] Andrey Nikolaevich Kolmogorov and Sergei Vasilovich Fomin. *Introductory Real Analysis*. Dover Publications, Mineola, NY, USA, 1975. ISBN: 0-486-61226-0.

[KKKVM04] Fernando Antonio Kuipers, Turgay Korkmaz, Marwan Krunz, and Piet Van Mieghem. Performance evaluation of constrained-based path selection algorithms. *IEEE Network*, 18(5):16–23, September 2004.

[KR01] James F. Kurose and Keith W. Ross. *Computer Networking: A Top-Down Approach Featuring the Internet*. Addison Wesley Longman, Boston, MA, USA, 2001. ISBN: 0-201-47711-4.

[Kui04] Fernando Antonio Kuipers. *Quality of Service Routing in the Internet: Theory, Complexity and Algorithms*. PhD thesis, Delft University, Delft, The Netherlands, 2004. ISBN: 90-407-2523-3.

[KVM05] Fernando A. Kuipers and Piet F. A. Van Mieghem. Conditions that impact the complexity of QoS routing. *IEEE/ACM Transactions on Networking*, 13(4):717–730, August 2005.

[KVMKK02] Fernando Kuipers, Piet Van Mieghem, Turgay Korkmaz, and Marwan Krunz. An overview of constraint-based path selection algorithms for QoS routing. *IEEE Communications Magazine*, 40(2):50–55, December 2002.

[KW98] Richard Kaye and Robert Wilson. *Linear Algebra*. Oxford University Press, Oxford, UK, 1998. ISBN: 0-19-850237-0.

[Lee95] Whay C. Lee. Topology aggregation for hierarchical routing in ATM networks. *ACM SIGCOMM Computer Communications Review*, 25(2):82–92, April 1995.

[LM04] Zhi Li and Prashant Mohapatra. QRON: QoS-aware routing in overlay networks. *IEEE Journal on Selected Areas in Communications*, 22(1):29–40, January 2004.

[LNC04] King-Shan Lui, Klara Nahrstedt, and Shigang Chen. Routing with topology aggregation in delay-bandwith sensitive networks. *IEEE/ACM Transactions on Networking*, 12(1):17–29, February 2004.

[LRV01] Jan Lundgren, Mikael Rönnqvist, and Peter Värbrand. *Linjär och Ickelinjär Optimering*. Studentlitteratur, Lund, Sweden, 2001. ISBN: 91-44-01798-7.

[Lue04] David G. Luenberger. *Linear and Nonlinear Programming*. Kluwer Academic Publishers, Norwell, MA, USA, 2004. ISBN: 1-4020-7593-6.

[Men90] Bert Mendelson. *Introduction to Topology*. Dover Publications, Mineola, NY, USA, 3rd edition, 1990. ISBN: 0-486-66352-3.

[PD00] Larry L. Peterson and Bruce S. Davie. *Computer Networks: A Systems Approach*. Morgan Kaufman Publishers, San Francisco, CA, USA, 2nd edition, 2000. ISBN: 1-55860-514-2.

[PM04] Michal Pióro and Deepankar Medhi. *Routing, Flow, and Capacity Design in Communication and Computer Networks*. Morgan Kaufman Publishers, San Francisco, CA, USA, 2004. ISBN: 0-12-557189-5.

[PS98] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Dover Publications, Mineola, NY, USA, 1998. ISBN: 0-486-40258-4.

[PTVF02] William H. Press, Saul A. Teukolsky, William T. Vettering, and Brain P. Flannery. *Numerical Recipes in C++: The Art of Scientific Computing*. Cambridge University Press, Cambridge, UK, 2nd edition, 2002. ISBN: 0-521-75033-4.

[RT02] Tim Roughgarden and Éva Tardos. How bad is selfish routing? *Journal of the ACM*, 49(2):236–259, March 2002.

[Rud76] Walter Rudin. *Principles of Mathematical Analysis*. McGraw-Hill, New York, NY, USA, 3rd edition, 1976. ISBN: 0-07-085613-3.

[Shi77] Georgi E. Shilov. *Linear Algebra*. Dover Publications, Mineola, NY, USA, 1977. ISBN:0-486-63518-X.

[Shi96] Georgi E. Shilov. *Elementary Real and Complex Analysis*. Dover Publications, Mineola, NY, USA, 1996. ISBN: 0-486-68922-0.

[SSBK04] Lakshminarayanan Subramanian, Ion Stoica, Hari Balakrishnan, and Randy Katz. OverQoS: An overlay based architecture for enhancing Internet QoS. In *Proceedings of NSDI*, San Francisco, CA, USA, March 2004.

[TLUN07] Wing-Yan Tam, King-shan Lui, Suleyman Uludag, and Klara Nahrstedt. Quality-of-service routing with path information aggregation. *Journal of Computer Networks*, 51:3574–3594, March 2007.

[TMW97] Kevin Thompson, Gregory J. Miller, and Rick Wilder. Wide-area Internet traffic patterns and characteristics. *IEEE Network*, 11(6):10–23, November 1997.

[VMK04] Piet Van Mieghem and A. Kuipers, Fernando. Concepts of exact QoS routing algorithms. *IEEE/ACM Transactions on Networking*, 12(5):851–864, October 2004.

[Wan00] Zheng Wang. *Internet QoS: Architectures and Mechanisms*. Morgan Kaufman Publishers, San Francisco, CA, USA, 2000. ISBN: 1-55860-608-4.

[WC96] Zheng Wang and Jon Crowfort. Quality-of-service routing for supporting multimedia applications. *IEEE Journal on Selected Areas in Communications*, 14(7):1228–1234, September 1996.

[Wro97] John Wroclawski. *RFC 2210: The Use of RSVP with IETF Integrated Services*. IETF, September 1997. Category: Standards Track.

[ZDE+93] Lixia Zhang, Stephen Deering, Deborah Estrin, Scott Shenker, and Daniel Zappala. RSVP: A new resource reservation protocol. *IEEE Network*, 7(5):8–18, September 1993.