Paraméteres problémák a kombinatorikus optimalizálásban és ezek távközlési alkalmazásai

Doktori értekezés

Jüttner Alpár

Matematika Doktori Iskola
vezetője: Laczkovich Miklós akadémikus

Alkalmazott Matematika Doktori Program
vezetője: Prékopa András akadémikus

Témavezető:
Frank András
tanszékvezető egyetemi tanár, a metematika tudomány doktora

Eötvös Loránd Tudományegyetem

2006.

Parametric Problems in Combinatorial Optimization and Their Applications in Telecommunications

PhD Thesis

Alpár Jüttner

Mathematics PhD School of Eötvös Loráns University
Head: Prof. Miklós Laczkovich, Member of Hungarian Academy of Sciences

Applied Mathematics PhD Program
Head: Prof. Emeritus András Prékopa, Member of Hungarian Academy of Sciences

Advisor:
András Frank, D.Sc.

Eötvös Loránd University, Budapest
Faculty of Sciences

2006.

Contents

Forewords

This thesis discusses mathematical problems having practical motivation and use. From the mathematical point of view, the discussed topics belong to the field of *combinatorial optimization*. A common feature of them is that the solutions are achieved by transforming the optimization tasks to certain *parametric problem*s having one or more parameters, thus resulting in continuous optimization tasks. Various approaches are presented to solve these special continuous optimization problems. The first applied technique is based on Megiddo's parametric search. This elegant method is discussed in detail because we believe it should also have several other applications in combinatorial optimization. We also show how basic approximating algorithms in continuous optimization, such as the Newton approximation or the subgradient method can help us finding optimal or approximating solutions for discrete optimization problems in finite or often in strongly polynomial time.

As we mentioned above, a strong emphasis is put on practical applications. An important goal of the thesis is to present complex real-life optimization tasks where nontrivial mathematical considerations make it possible to find optimal or near optimal solutions efficiently. Most of the applied research presented it the thesis was done in the Traffic Analysis and Network Performance Laboratory at Ericsson Research, Hungary, thus the practical problems this thesis deals with mainly come from the telecommunication industry, but some of them have wider applicability. In order to demonstrate the practical usability of the solutions, we also present case studies with experimental analysis of the proposed exact and approximating solution.

Structure of the thesis

The starting point of the research presented in Chapter 1 was the papers of Frederickson and Solis-Oba [FSO99, FSO98]. In their works the authors solved a kind of budgeted inverse optimization problem on minimum spanning trees, and later extended it to arbitrary matroids. These results called for a further extension to matroid intersections or even to submodular flows. The algorithm proposed by Frederickson and Solis-Oba does not seem to extend to these cases. Therefore an alternative and more general approach is presented in Chapter 1. Strangely, although the presented framework is general enough to provide a solution to several other known budgeted optimization problems of this type, the algorithm and the proof of its correctness and strong polynomiality is actually simpler than the one

of Frederickson and Solis-Oba. The results of this chapter has been published in [Jüt03] and [Jüt05a].

The budgeted matroid intersection problem above is solved by essentially reducing it to the following problem. We are given a (possibly very large) linear program and we know a strongly polynomial combinatorial and *linear* (see Section 1.1.1) algorithm that solves it. We are also given an additional linear constraint and the task is to solve this extended problem in strongly polynomial time. The way how we solved this problem extends to the case when we add any constant number of constraints (rows) or variables (columns) to the original linear program. This latter problem was posed and solved by Plotkin, Norton and Tardos in [NPT92]. Our solution described in Chapter 2 improves the running time from $O(T^{d+2})$ obtained in [NPT92] to $O(T^{d+1})$, where T is the running time of the algorithm for the basic linear program and d is the number of additional constraints and variables. This chapter is based on [Jüt05c].

Solving network routing problems one often comes across the following problem. We are given a directed graph $G = (V, E)$, a cost function $c : E \longrightarrow \mathbb{R}_+$, a delay function $d : E \longrightarrow \mathbb{R}_+$ on the edges of the graph, a resource (delay) constraint $\Delta \in \mathbb{R}_+$ a source $s \in V$, a target $t \in V$, and the task is to find a path p for which $d(p) \leq \Delta$ and which is of minimum cost among these paths. This problem is well known to be $\mathcal{NP}$-hard. A usual way to solve this problem is to apply Lagrange relaxation to the delay constraint. In this way the problem turns to a maximization problem over a single variable. To solve this dual problem a simple but practically very efficient method — referred as Handler-Zang method in this thesis — has been developed independently by several authors [HZ80, BG96, JSMR01]. However there was little known about its theoretical running time. In Chapter 3 we prove that Handler-Zang method is strongly polynomial not only for the shortest path case but also for any resource constrained optimization problem with linear cost and resource (delay) functions. For the special case of shortest paths, a tighter running time estimation is also presented in Section 3.3.2. This research was directly motivated by a vital telecommunication problem, thus we present a practical evaluation of the method in this scenario and also show some tricks to further improve its practical performance. The application of this algorithm for telecommunication networks was published in [JSMR01]. This paper has several citations by authors working in telecommunication research. The theoretical analysis of the algorithm was presented in [Jüt05b].

The techniques used in Chapter 3 also makes it possible to achieve some improvements

on the known running time bounds for the Newton-Dinkelbach method for fractional optimization problems. Namely, Chapter 4 shows that this method takes $O(n^2 \log n)$ iterations if both the cost and the weight function are linear. This improves the result obtained in [Rad98] by a factor $O(\log n)$. Moreover, the proof is significantly shorter. We also show that if the set of feasible solutions is given as the 0-1 vertices of the integer polyhedron $\mathcal{Q} := \{x \in \mathbb{R}^n : Ax = b, x \geq 0\}$, then the Newton-Dinkelbach method takes $O(n \log n)$ iterations. This is a straightforward extension of the case of shortest paths in a directed acyclic graph.

Chapter 5 is also a case study of a real life optimization problem coming from the telecommunication industry, a planning problem of the access network of third generation mobile systems. It presents a nontrivial application of Lagrange relaxation and branch-and-bound technique to solve a hard optimization problem up to optimality on moderate problem size, demonstrates how to use this method as a subroutine to solve larger problems, and also shows how a careful combination of the bipartite b-matching and a Simulated Annealing can also be used to solve this problem. The results presented in this chapter has been published in [JOF05].

Acknowledgement

First of all, the author is indebted to his advisor prof. András Frank for the vast amount of support, the fruitful discussions and for being an inexhaustible source of interesting mathematical questions and ideas.

The work on this thesis was carried out at the Department of Operations Research, Eötvös University of Sciences and at Traffic Analysis and Network Performance Laboratory of Ericsson Research, Hungary. The author wishes to thank both organizations for the support and the inspiring atmosphere.

The author would also like to acknowledge the useful ideas, valuable suggestions, comments and cooperation of Attila Bernáth, Tibor Cinkler, Zoltán Fiala, András Frank, István Gódor, János Harmatos, Zoltán Király Gábor Magyar, Ildikó Mécs, Rolf Möhring, András Orbán, Michał Pioró, Thomasz Radzik, Jácint Szabó, Áron Szentesi, Árpád Szlávik, Balázs Szviatovszki and Zsuzsa Weiner.

Finally, the author would like to express his greatest thanks to his wife, Lilla and his sons, Domokos and Ábel for their continuous support, endless patience and love.

Chapter 1

Budgeted Optimization Problems

A typical optimization problem in combinatorial optimization consists of minimizing (or maximizing) a linear objective function over some combinatorial objects. Classical problems, like shortest paths, maximum flows, minimum cost circulations can all be interpreted this way.

There are, however, results dealing with other type of optimization problems over the same combinatorial objects. For example, it is well-known that if we are given a directed graph $G = (V, E)$, a capacity function $w : E \longrightarrow \mathbb{R}$ and two nodes $s, t \in V$, then the minimum s—t cut can be found algorithmically in strongly polynomial time. Fulkerson [Ful59] introduced and solved the so-called *budgeted version* of this minimum cut problem. In this case we are also given a cost function $c : E \longrightarrow \mathbb{R}$, and a budget constraint $B > 0$ and the task is to increase the capacity of the minimum s—t cut as much as possible by increasing the components of the capacity function w individually with the following side constraint. If we increase the capacity of an edge e by δ, it costs us $\delta c(e)$ and the total cost of capacity increments is bounded by B. Later R. K. Ahuja and J. B. Orlin [AO95] gave a more efficient polynomial algorithm to this problem.

Other papers were also devoted to similar problems. Fulkerson and Harding [Har77, FH75] solved the same budgeted version of the shortest s—t path problem. Later G.N. Frederickson and R. Solis-Oba solved the budgeted version of the Minimum Spanning Tree problem [FSO99]. In contrast with the budgeted versions of the minimum cut and the shortest s—t path problem — which were both transformed essentially to the minimum cost flow problem, the solution of the budgeted version of the minimum spanning tree problem needed deeper considerations. Later they extended their method to arbitrary

matroids in [FSO97, FSO98].

A natural extension of this result would be the problem of decreasing the maximum weight common base of two matroids. The algorithm presented in [FSO98] does not seem to extend to this case. As the main contribution of this chapter, we propose a different approach that can solve this latter problem in strongly polynomial time. In contrast with [FSO98], this solution does not depend on any substantial property of matroids or matroid intersections, thus the same scheme can be also used to solve other budgeted optimization problems including all the problems mentioned above.

Therefore it is worth considering the problem above in the following general form. (In order to be consistent with [FSO97, FSO98], we use an equivalent form where the problem is to decrease the weight of the maximum weight element.) We are given an underlying set E and a combinatorial optimization problem called *basic problem* (e.g. the common bases of two given matroids in the latter example). We assume that the convex hull $\mathcal{P}$ of the feasible solution is a polyhedron. (This holds in the case of the problems mentioned above.) Then, for a given weight function $w : E \longrightarrow \mathbb{R}$, cost function $c : E \longrightarrow \mathbb{R}$, and budget constraint $B > 0$ the corresponding *budgeted optimization problem* is to compute the value

$$\alpha := \min\{\max\{(w - y)x : x \in \mathcal{P}\} : y \in \mathbb{R}^E, y \geq 0, cy \leq B\} \tag{1.1}$$

along with a minimizing vector y^*.

It will be pointed out that this problem essentially leads to the *bounded version* of the basic problem, that is, the problem of finding a maximum weight element in the polyhedron

$$\mathcal{P}^u := \mathcal{P} \cap \{x \in \mathbb{R}^E : x \leq u\}, \tag{1.2}$$

where $u \in \mathbb{R}^E$ is an arbitrary given constraint vector. More exactly, the following will be proved.

Theorem 1.1 *If there exists an algorithm which is able to find a maximum weight element in $\mathcal{P}^u$ along with a dual optimal solution (with respect to a linear programming description of $\mathcal{P}^u$) in time T for an arbitrary vector u and this algorithm satisfies the so called* linearity condition *in u (see Section 1.1.1 for the precise definition), then there exists an algorithm for solving Problem (1.1) in time $O(T^2)$.*

Note that the linearity condition of an algorithm does not refer to the running time, but it is a restriction on what kind of operations can be made with the input. So, the meaning of "*linear algorithm*" and "*linear time algorithm*" are very different.

In Section 1.2, problem (1.1) will be reduced to a parametric problem using Lagrangian relaxation, that is, we will show that problem (1.1) can be transformed to the maximization of the function

$$L(\lambda) := \max\{wx : x \in \mathcal{P}^{\lambda c}\} - \lambda B \tag{1.3}$$

with only one variable.

To do this maximization, one may use e.g. the binary search technique. This gives the value of the optimal solution, but some more effort is still necessary to find the optimal solution itself. In addition, the focus is on *strongly polynomial* time algorithms in this chapter and binary search does not provide such an algorithm.

So, in Section 1.3 we show another way to construct an algorithm $\mathcal{A}'$ to maximize $L(\lambda)$ and to find an optimal solution to (1.1) in strongly polynomial time. The algorithm is based on N. Megiddo's Parametric Search Method. This technique uses a separation subroutine, and it considers this subroutine not only as a black-box but it also uses its *inner structure* to construct the algorithm $\mathcal{A}'$. An advantage of this technique is the straightforward bound on the running time and that it can be used in very general context (see e.g. [NPT92, CM93]), but the separation subroutine must satisfy the linearity condition. This assumption is not a strong restriction in the sense that most combinatorial optimization problems that can be solved in strongly polynomial time can also be solved with linear algorithms. Section 1.1 gives a sketch of Megiddo's technique in general, the definition of linear algorithms and shows some examples what can be and what cannot be computed by linear algorithms.

In Section 1.4 a slightly more general problem will be discussed, when the modification of some components of the weight function can be prohibited.

In Section 1.5 the presented method will be applied to some already solved problems and we will show two new applications. The first one is the budgeted optimization problem of the minimum cost circulation problem and the second one extends the Frederickson's and Solis-Oba's result [FSO99, FSO97, FSO98] to matroid intersections from single matroids. Moreover, this method can be used for budgeted submodular optimization problems as well.

1.1 Megiddo's principle

Definition 1.2 *A real number λ^* is said to be given by a* separation algorithm $\mathcal{A}(\lambda)$ *if $\mathcal{A}(\lambda)$ is able to decide whether $\lambda < \lambda^*$, $\lambda = \lambda^*$ or $\lambda > \lambda^*$ by answering -1, 0 or $+1$.*

Solving combinatorial problems, one often comes across the problem of computing the explicit value of a number $\lambda^* \in \mathbb{R}$ given by a strongly polynomial time separation algorithm. Without any restrictions, only approximation-like algorithms can be given for this problem. For example, it is easy to see that for $\lambda^* := \sqrt{2}$ there exists a separation algorithm using only comparisons, additions and multiplications of rational numbers and the input number, but λ^* cannot be obtained through these operations.

Megiddo [Meg79] showed that, roughly, if one can avoid multiplications in the separation algorithm, then λ^* can also be computed in strongly polynomial time. Namely, he (implicitly) proved the following theorem.

Theorem 1.3 *Suppose that we are given a $\lambda^* \in \mathbb{R}$ through a separation algorithm $\mathcal{A}(\lambda)$. If $\mathcal{A}$ is* linear in λ *and works in time T, then there exists an algorithm that runs in time $O(T^2)$ and computes the explicit value of λ^*.*

The idea of this method is to simulate the *steps* of the execution of $\mathcal{A}$ on the input λ^* by using only the necessary partial information about the input in each step of the algorithm. At the end of this procedure from these partial information we will be able to determine the right value of λ^*.

For this, however, it is necessary to require the linearity of the algorithm. The precise definition of this assumption comes in the next section, then the theorem will be proven in Section 1.1.2.

1.1.1 Linear Algorithms

To define the notion of a linear algorithm we use a RAM machine which has an additional storage called ***L**imited **A**ccess **M**emory*. It may store real numbers, but an algorithm running on this machine has only limited access to this storage. Namely, it can reach the contents of the LAM only through the following operations.

- It can write an element of the RAM or LAM into an element of the LAM,
- it can multiply an element of the LAM with an element of the RAM and store the result in the LAM,
- it can add an element of the LAM to another element of the LAM and store the result in the LAM,

- it can compare two elements of the LAM.

However, for example, it cannot multiply two elements of the LAM and it cannot read them (that is it cannot copy an element of LAM into the RAM).

Definition 1.4 *Let $\mathcal{A}(x, y)$ be an algorithm where x and y are its input vectors. We say that $\mathcal{A}$ is* ***linear in*** **x** *if it gets x in the LAM, and also puts the output in the LAM. The algorithm has full access to the other part of the input, in other words it gets it in the RAM.*

It can be seen that most of the basic operations of data structures can be implemented on a LAM machine, for example we can choose the minimal element of a set of numbers stored in the LAM and we can also sort its elements. However if we put the elements of the matrix into the LAM, we cannot compute its determinant, because we cannot avoid the multiplications of two elements of the matrix (the determinant itself is a nonlinear polynomial of the elements of the matrix).

The following claims are given to demonstrate the capability of linear algorithms.

Claim 1.5 *There exists an implementation of Dijkstra's algorithm to find a shortest $s-t$ path in a weighted directed graph $G = (V, E, w)$ which is linear in the weight function.*

Proof. During its execution, Dijkstra's algorithm builds a shortest-path tree T from the node s. First, T consists of only the node s. Then in each round it finds a node $v \in V \setminus V(T)$ whose path constructed from a path in T and one additional edge has the smallest weight, and it puts v and the last edge of the corresponding path to T. It can be done, because we can calculate the weight of these paths in the LAM, and we can also choose the path having the smallest weight using only comparisons. □

Claim 1.6 *The strongly polynomial time Edmonds-Karp algorithm [EK72] for the MAX-FLOW problem can be implemented in such a way that it is linear in the capacity function.*

Proof. The algorithm starts with the zero flow and it repeats the following steps until the maximum flow is found.

First, it constructs an auxiliary digraph from the original graph by comparing the current edge-values of the flow with the edge-capacities and with zero. Then it looks for a shortest source-sink path in this graph and increases the amount of the flow by modifying its value on the edges of this paths, which can be done using a minimum calculation and some additions. □

Claim 1.7 *Goldberg and Tarjan's strongly polynomial time cycle canceling algorithm [GT89] for the minimum cost circulation problem can be implemented in such a way that it is linear in the cost function. It also can be implemented in such a way that it is linear in the bounding vectors. In addition, these algorithms compute the corresponding linear programming dual optimal solution.*

Proof. This algorithm starts with an arbitrary feasible circulation, then in each iteration an auxiliary digraph is constructed and the circulation is improved on the edges of a minimum mean cycle of the auxiliary graph. All these can be done without multiplying the elements of the cost function to each other and also without multiplying the elements of the bounding vectors to each other. □

It is worth mentioning that there cannot exist a strongly polynomial time algorithm for this problem which is linear *both* in the cost function and in the bounding vectors because we cannot avoid multiplying the elements of the bounding vectors with their corresponding costs.

A common generalization of maximum cost circulations, maximum cost bases of a matroid, and maximum cost common base of two polymatroids are *submodular flows*[EG77]. One of its several equivalent definitions is the following. (See e.g. [Fuj91])

Definition 1.8 *A family $\mathcal{F} \subseteq 2^V$ is called a* ***crossing family*** *over the underlying set V if for each crossing $X, Y \in \mathcal{F}$ (i.e., $X, Y \in \mathcal{F}$, $X \cap Y \neq \emptyset$, $X \backslash Y \neq \emptyset$, $Y \backslash X \neq \emptyset$, and $X \cup Y \neq V$) we have $X \cup Y, X \cap Y \in \mathcal{F}$. A function $b : \mathcal{F} \longrightarrow \mathbb{R}$ on the crossing family $\mathcal{F}$ is called* ***crossing-submodular*** *if for each crossing $X, Y \in \mathcal{F}$ we have the submodularity inequality*

$$b(X) + b(Y) \geq b(X \cup Y) + b(X \cap Y). \tag{1.4}$$

Definition 1.9 *Let $G = (V, E)$ be a directed graph with a vertex set V and an edge set E. Let $f, g : E \longrightarrow \mathbb{R} \cup \{\pm\infty\}$ be a lower and an upper capacity function and let $w : E \longrightarrow \mathbb{R}$ be a cost function. Let $\mathcal{F} \subseteq 2^V$ be a crossing family with $\emptyset, V \in \mathcal{F}$ and $b : \mathcal{F} \longrightarrow \mathbb{R}$ be a crossing-submodular function with $b(\emptyset) = b(V) = 0$. The* submodular flow problem *is described as follows.*

$$\max \quad wx \tag{1.5a}$$

$$f \leq x \leq g \tag{1.5b}$$

$$\varrho_x(X) - \delta_x(X) \leq b(X) \quad \text{for all } X \subseteq V, \tag{1.5c}$$

where $\varrho_x(X)$ and $\delta_x(X)$ are the sum of the components of x corresponding to the edges entering X and leaving X, respectively.

Although the algorithm for finding a minimum cost submodular flow is quite complex, the following claim is straightforward to check.

Claim 1.10 *The strongly polynomial time algorithm for the minimum cost submodular flow problem described in [Fuj91] can be implemented in such a way that it will be linear in the bounding functions f and g. This algorithm also gives the corresponding linear programming dual optimal solution.* □

1.1.2 Parametric Search

In this section Theorem 1.3 is proven by giving an algorithm $\mathcal{A}'$ (running on a plain RAM machine) that computes the value of λ^*. The new idea of the Parametric Search developed by Megiddo [Meg79] is that instead of making an independent "optimizer" algorithm that repeatedly *calls* the separation algorithm $\mathcal{A}$ during its execution, $\mathcal{A}'$ will "simulate" how $\mathcal{A}$ would run on the value λ^*. Therefore we now give a scheme how to construct the algorithm $\mathcal{A}'$ from the linear separation algorithm $\mathcal{A}$.

First, we replace each element of the LAM with a pair of real numbers called *parametric numbers*. Addition and subtraction of these numbers are defined in the same way as the usual vector addition and vector subtraction while the multiplications and divisions of these numbers are avoided by the linearity assumption on the algorithm $\mathcal{A}$. When a real number x is put into an element of the LAM, it is converted to the parametric number $(x, 0)$. The multiplication of a real number c and a parametric one (x, y) is defined to be (cx, cy).

The only undefined part of $\mathcal{A}'$ is when it compares two values in the LAM, namely it inquires about whether $(x_1, y_1) \leq (x_2, y_2)$ or not. In this case

- if $y_1 = y_2$, then we answer *true* if and only if $x_1 \leq x_2$.
- if $y_1 > y_2$, then we call $\mathcal{A}$ with the value $\lambda := \frac{x_2 - x_1}{y_1 - y_2}$. We answer *true* if and only if $\mathcal{A}$ returns that $\lambda^* < \lambda$.
- if $y_1 < y_2$, then we call $\mathcal{A}$ with the value $\lambda := \frac{x_2 - x_1}{y_1 - y_2}$. We answer *true* if and only if $\mathcal{A}$ returns that $\lambda^* > \lambda$.

If $\mathcal{A}$ happens to return that $\lambda = \lambda^*$, we stop, since we found λ^*.

The idea behind this construction is that a parametric number (x, y) corresponds to the linear expression $x + y\lambda^*$. The algebraic operations are defined to be consistent with the result of the same operations on the corresponding linear expressions.

A comparison $(x_1, y_1) \leq (x_2, y_2)$ means that $x_1 + y_1\lambda^* \leq x_2 + y_2\lambda^*$, which is equivalent to $\lambda^* \leq \frac{x_2 - x_1}{y_1 - y_2}$, $\lambda^* \geq \frac{x_2 - x_1}{y_1 - y_2}$ or $x_1 \leq x_2$ depending on the sing of $y_1 - y_2$. So, we can make decision by a simple comparison or using the original $\mathcal{A}$ with $\lambda := \frac{x_2 - x_1}{y_1 - y_2}$ as input.

Now, let us run $\mathcal{A}'$ with the parametric number $(0, 1)$ as the input.

Claim 1.11 *$\mathcal{A}'((0,1))$ calls $\mathcal{A}$ with λ^* during its execution.*

Proof. Let us suppose that it is not so. In this case $\mathcal{A}'$ returns with a value when it terminates. $\mathcal{A}$ stored the return value originally in the LAM, so $\mathcal{A}'$ returns a parametric number, i.e. a linear function of λ.

During the execution each comparison involved in calling $\mathcal{A}$ gives us a half-line as a set of possible place of λ^*. Let Λ be the intersection of these closed half-lines. Λ is a (closed) interval and $\lambda^* \in \Lambda$. On the other hand it is easy to see that for any $\lambda \in \Lambda$ the steps of $\mathcal{A}'((0,1))$ correspond to the steps of $\mathcal{A}(\lambda)$. So, the value returned by $\mathcal{A}'$ is right for all $\lambda \in \Lambda$. Moreover, it is a linear expression of λ and for λ^* it is equal to 0. These, together with the fact that the result is ± 1 or 0 for all λ, yield that Λ should be equal to $\{\lambda^*\}$. But, it is only possible if $\mathcal{A}$ was called with λ^* during the execution of $\mathcal{A}'$, contradicting to the assumption. □

Finally, let T denote the running time of $\mathcal{A}$. The running time of the main algorithm is the sum of the time used by $\mathcal{A}'$ itself and the time required by the comparisons. Each comparison can be computed by a simple execution of $\mathcal{A}$, the number of the comparisons is at most T and the number of the steps are taken by $\mathcal{A}'$ is $O(T)$. Thus the total time required by the algorithm is $O(T^2)$.

In special cases the running time can often be significantly improved in several ways. See [Meg83] or [Rad98] for more detail.

1.2 The Lagrangian relaxation of the problem

In this section we show how problem (1.1) can be transformed to a parametric problem. Let

$$L(\lambda) = \max_{z \in \mathcal{P}^{\lambda c}} wz - \lambda B \tag{1.6}$$

and

$$L^* := \max_{\lambda \geq 0} L(\lambda). \tag{1.7}$$

Theorem 1.12 *$L^* = \alpha$, where α is the optimal solution of the budgeted optimization problem (1.1).*

Proof. From now on let $Ax \leq b$ be a linear description of $\mathcal{P}$, that is

$$\mathcal{P} = \{x \in \mathbb{R}^n : Ax \leq b\}. \tag{1.8}$$

It is worth mentioning that the inequalities defining $\mathcal{P}$ need not be given explicitly and there is no constraint on the number of them.

By the Duality Theorem

$$\max_{x \in \mathcal{P}} (w - y)x = \min\{\pi b : \pi \geq 0, \pi A = w - y\}, \tag{1.9}$$

So, Problem (1.1) is equivalent to the following linear program.

$$\begin{aligned} \min \quad & \pi b \\ \pi, y &\geq 0 \\ \pi A + y &= w \\ yc &\leq B \end{aligned} \tag{1.10}$$

From (1.6) and (1.7) it follows that

$$\begin{aligned} L^* = \max & \, wz - \lambda B \\ \lambda &\geq 0 \\ z &\leq \lambda c \\ Az &\leq b, \end{aligned} \tag{1.11}$$

which is the dual problem of (1.10). □

1.3 Maximization of $L(\lambda)$

In this section we give an algorithm which maximizes the function $L(\lambda)$ and gives an optimal solution to (1.10).

First, using the Duality Theorem, we get that

$$L(\lambda) = \min \pi b + \lambda(cy - B) \tag{1.12}$$

$$\pi, y \geq 0$$
$$\pi A + y = w.$$

Let L^{opt} denote the set of λ's maximizing $L(\lambda)$. Obviously L^{opt} is a (closed) interval. For the sake of simplicity the following notation is used.

Definition 1.13 *$\lambda \leq L^{opt}$ if and only if $\lambda \leq x$ for all $x \in L^{opt}$. $\lambda \geq L^{opt}$, $\lambda < L^{opt}$ and $\lambda > L^{opt}$ are defined similarly.*

Claim 1.14 *Let π^0, y^0 be an optimal solution to (1.12) for some $\lambda \geq 0$. Then*

- *if $cy^0 > B$ then $\lambda \leq L^{opt}$,*
- *if $cy^0 < B$ then $\lambda \geq L^{opt}$.*

Proof. Let us suppose that $cy^0 > B$ and $\lambda' < \lambda$. Therefore

$$L(\lambda) = \pi^0 b + \lambda(cy^0 - B) > \pi^0 b + \lambda'(cy^0 - B) \geq L(\lambda'), \tag{1.13}$$

proving that $\lambda' \notin L^{opt}$. The second implication can be proven similarly. □

Claim 1.15 *Let π^0, y^0 be an optimal solution to (1.12) for some $\lambda \geq 0$ and let $cy^0 = B$. Then it i s also an optimal solution to (1.10), that is y^0 is an optimal decrement of the weight vector w in Problem (1.1).*

Proof. The feasibility of π^0, y^0 is clear. Let π^*, y^* be an optimal solution to (1.10). Then

$$\alpha = \pi^* b \leq \pi^0 b = \pi^0 b + \lambda(cy^0 - B) \leq \pi^* b + \lambda(cy^* - B) \leq \pi^* b = \alpha \tag{1.14}$$

implies the optimality. □

If $L(\lambda) = -\infty$ that is the polyhedron $\mathcal{P}^{\lambda c}$ is empty, a *certification of its emptiness* is a vector $(\pi^\urcorner, y^\urcorner)$ for which

$$\begin{aligned} \pi^\urcorner, y^\urcorner &\geq 0 \\ \pi^\urcorner A + y^\urcorner &= 0 \\ \pi^\urcorner b + y^\urcorner \lambda c &< 0. \end{aligned} \tag{1.15}$$

Claim 1.16 *Suppose that $\mathcal{P}^{\lambda c}$ is empty for some $\lambda \geq 0$ and let $\pi^\urcorner, y^\urcorner$ be a certification of its emptiness (i.e. a solution to (1.15)). Then*

- *if $cy^\urcorner \geq 0$ then $\lambda < L^{opt}$,*
- *if $cy^\urcorner \leq 0$ then $\lambda > L^{opt}$.*

Proof. $\pi^\urcorner, y^\urcorner$ is also certification of emptiness of $\mathcal{P}^{\lambda' c}$ for all $\lambda' \leq \lambda$ or for all $\lambda' \geq \lambda$ depending on whether $cy^\urcorner \geq 0$ or not. □

Now, we are ready to prove

Theorem 1.1 (Restated) *Let $\mathcal{A}(\lambda)$ be an algorithm linear in λ and with running time T, and assume that it computes either $L(\lambda)$ along with a dual optimal solution (1.12) or a certification of the emptiness of $\mathcal{P}^{\lambda c}$ if $L(\lambda) = -\infty$. Then there exists an algorithm for solving Problem (1.1) in time $O(T^2)$.*

Proof. Claim 1.15 shows that to solve Problem (1.1) it is enough to find a multiplier λ^* maximizing the function $L(\lambda)$ and an optimal solution π^*, y^* to (1.12) with λ^* in place of λ with the property that $cy^* = B$.

We apply Megiddo's Parametric Search for the maximization of the function $L(\lambda)$, but a small technical difficulty arises because we are not able to check directly whether or not $\lambda \in L^{opt}$ for an arbitrary λ.

Apart from the comparisons we construct $\mathcal{A}'$ from $\mathcal{A}$ in the same way as in Section 1.1.

The only difference is when the algorithm makes a comparison which is involved in calling $\mathcal{A}$ with the input $\lambda := \frac{x_2 - x_1}{y_1 - y_2}$. In this case we do the following.

- If we get $L(\lambda) = -\infty$, then we use Claim 1.16 to decide whether $\lambda < \lambda^*$ or $\lambda > \lambda^*$.
- If $\mathcal{A}$ happens to consider λ to be optimal, that is we get a solution π^λ, y^λ of (1.12) such that $cy^\lambda = B$, then by Claim 1.15 (π^λ, y^λ) is also an optimal solution to (1.10) so the execution can be finished.

- If $cy^\lambda > B$, then by Claim 1.14 it means that $\lambda < L^{opt}$ *unless* $\lambda \in L^{opt}$ so we accept if the question was $\lambda \leq \lambda^*$ and reject if the question was $\lambda \geq \lambda^*$.

- In the case when $cy^\lambda < B$ we give opposite answers.

For some technical reasons, the algorithm also has to take care to avoid the redundant executions of $\mathcal{A}$ i.e. if the outcome of a comparison can be derived from the outcomes of the previous ones, then we make decision based on the previous comparisons rather then to call $\mathcal{A}$ once more.

Now let us run $\mathcal{A}'$. During the execution each comparison which is involved in calling $\mathcal{A}$ gives us a half-line as a set of possible values of λ^*. Let Λ be the intersection of these closed half-lines. Λ is a (closed) interval and $L^{opt} \subseteq \Lambda$. Since we never made redundant comparisons, $\text{int}\Lambda \neq \emptyset$. ($\text{int}\Lambda$ is the set of the interior points of Λ.)

On the other hand it is easy to see that for any $\lambda \in \text{int}\Lambda$ the steps of $\mathcal{A}'$ correspond to the steps of $\mathcal{A}(\lambda)$. So, at the end of its execution $\mathcal{A}'$ gives us the optimum value, which parametric number. Thus, it corresponds to a linear function of λ and it is equal to $L(\lambda)$ for all $\lambda \in \text{int}\Lambda$. Moreover, because of the continuity of $L(\lambda)$, this holds for all $\lambda \in \Lambda$. $\mathcal{A}'$ also gives us an optimal solution $(\pi(\lambda), y(\lambda))$ to (1.12) as two vectors of linear functions of λ. These are right optimal solutions to (1.12) for all $\lambda \in \text{int}\Lambda$. Again, because of continuity, it follows that they are right for all $\lambda \in \Lambda$.

From the above considerations it follows that one of the extrema of Λ maximizes the function $L(\lambda)$. Denote it by λ^o and let (π^o, y^o) be the solution $\mathcal{A}$ returned when it was called with λ^o. Because Λ is a subset of the half-line defined by y^o and λ^o maximizes $L(\lambda)$ over the set Λ, it follows that $cy^o - B$ and $cy(\lambda^o) - B$ have opposite signs. So, there exists a suitable coefficient $0 \leq \mu \leq 1$ such that $cy^* = B$, where $\pi^* := \mu\pi^o + (1-\mu)\pi(\lambda^o)$ and $y^* = \mu y^o + (1-\mu)y(\lambda^o)$. Since both (π^o, y^o) and $(\pi(\lambda^o), y(\lambda^o))$ are optimal solutions to (1.12) with λ^o in place of λ, (π^*, y^*) is also an optimal solution. Finally, Claim 1.15 ensures that (π^*, y^*) is an optimal solution to (1.10) as well. To sum up, y^* is an optimal solution to Problem (1.1).

The bound on the running time can be obtained in the same way as in Section 1.1.2. □

1.4 When some components of the cost are fixed

In this section we extend the problem (1.1), so that the modification of some components of the weight function can be prohibited. The main benefit of this extension is that it enables us to use auxiliary variables to define the polyhedron $\mathcal{P}$ for the basic problem.

Let $\mathcal{P} := \{(x_1, x_2) \in \mathbb{R}^{n+m} : Ax_1 + Cx_2 \leq b\}$. So, we are looking for the optimal solution to the problem

$$\alpha := \min \{\max \{(w_1 - y)x_1 + w_2x_2 : (x_1, x_2) \in \mathcal{P}\} : y \geq 0, cy \leq B\} \tag{1.16}$$

As in Section 1.2 this can be transformed to the following linear program.

$$\begin{aligned} \min \quad & \pi b \\ \pi, y &\geq 0 \\ \pi A + y &= w_1 \\ \pi B &= w_2 \\ yc &\leq B. \end{aligned} \tag{1.17}$$

The corresponding Lagrangian relaxation of this problem is

$$\begin{aligned} L(\lambda) = \min \quad & \pi b + \lambda(cy - B) \\ \pi, y &\geq 0 \\ \pi A + y &= w_1 \\ \pi B &= w_2, \end{aligned} \tag{1.18}$$

and by the Duality Theorem

$$L(\lambda) = \max\{w_1z_1 + w_2z_2 : (z_1, z_2) \in \mathcal{P}^\lambda\} - \lambda B, \tag{1.19}$$

where $\mathcal{P}^\lambda := \{(x_1, x_2) \in \mathcal{P} : x_1 \leq \lambda c\}$. The following theorem can be proven similarly to Theorem 1.12.

Theorem 1.17 *$L^* = \alpha$, where*

$$L^* := \max_{\lambda \geq 0} L(\lambda). \tag{1.20}$$

□

Finally, the maximization of $L(\lambda)$ can be done in the same way as in Section 1.3.

1.5 Applications

The algorithm presented in the previous sections can be used for a wide range of problems; it is enough to check whether we are able to optimize on the corresponding bounded polyhedron using an algorithm which is linear in the bounding vector (or more generally, using an algorithm that uses λc as the bounding vector and which is linear in λ).

First, as an example, we show how Theorem 1.1 can be applied to Fulkerson's and Harding's problem. Then, Section 1.5.2 and 1.5.3 present two new applications, the budgeted minimum cost circulation and the budgeted polymatroid intersection problem. The latter one extends the problem examined in [FSO98].

Let us mention that the direct use of Theorem 1.1 gives worse running time than those obtained in [AO95, FH75] and in [FSO98] for the budgeted maximum flow, minimum source-sink path and matroid optimization problems. However, these running times can be improved several ways in special cases. See [Meg83] and [Rad98] for these techniques.

1.5.1 Maximizing the minimum source-sink path

Fulkerson and Harding [FH75, Har77] solved the case of budgeted optimization problems when we have a non-negative length and a cost function on the edges of a directed graph and we want to increase the length of the minimum length path between two predefined nodes as much as possible by increasing the lengths of the edges keeping the budget constraint.

The minimum length $s-t$ path problem can be formulated as an uncapacitated minimum cost flow problem. So, the computation of $L(\lambda)$ is a capacitated minimum cost flow problem, for which there exists strongly polynomial time algorithm which is linear in its constraint vector. It also gives back the dual solution which is in the suitable form we need in (1.12).

1.5.2 Increasing the Cost of the Minimum Cost Circulation

We are given a directed graph $G = (V, E)$, a cost function $c : E \longrightarrow \mathbb{R}$, a lower and an upper bound $l, u : E \longrightarrow \mathbb{R}$ on its edges. Moreover we are given a cost function $c^m : E \longrightarrow \mathbb{R}$ of the modification and a budget constraint $B > 0$. The task is to find an increment of the

cost function c within the budget constraint which increases the cost of the c-minimal cost circulation as much as possible.

This problem can also be handled with the method because the corresponding parametric problem is a minimum cost circulation problem on the graph G with the cost function c, lower bound l and upper bound $u^\lambda(e) := \min(u(e), \lambda c^m(e))$ which can be computed by a strongly polynomial time algorithm which is linear in λ. The dual optimal solution returned by this algorithm can be easily transformed to a solution of (1.12) in the same way as it is discussed in the following section.

1.5.3 Polymatroid intersection and submodular flows

A natural extension of the problem examined in [FSO98] is the *budgeted matroid intersection problem*. In this case we are given two matroids $M_1 = (E, \mathcal{I}_1)$ and $M_2 = (E, \mathcal{I}_2)$ with a common ground set, a weight and a cost function $w, c : E \longrightarrow \mathbb{R}$ and a budget constraint $B > 0$ and we want to decrease the weight of the maximum weight common base of M_1 and M_2 as much as possible by decreasing independently the weight of the elements of the ground set with the side constraint that the total cost of the decreasing must be at most B.

The common generalization of this problem and the problem presented in Section 1.5.2 is the case of budgeted optimization problem when we are given a submodular flow problem and we are looking for a modification of its cost function which increases the cost of the minimum cost feasible flow as much as possible.

Namely, using the notation of Definition 1.9, we are given an additional cost function $c : E \longrightarrow \mathbb{R}$, a budget constraint B and the problem is to find

$$\min\{\max\{(w - y)x : x \in \mathcal{P}\} : y \geq 0, cy \leq B\} \tag{1.21}$$

where

$$\mathcal{P} := \{x \in \mathbb{R}^E : f \leq x \leq g \ , \ \varrho_x(X) - \delta_x(X) \leq b(X) \quad \text{for all } X \subseteq V\}. \tag{1.22}$$

The corresponding bounded problem is

$$\max \quad wx \tag{1.23a}$$
$$\varrho_x(X) - \delta_x(X) \leq b(X) \quad \text{for all } X \subseteq V \tag{1.23b}$$
$$x \leq -f \tag{1.23c}$$
$$x \leq g \tag{1.23d}$$
$$x \leq u, \tag{1.23e}$$

where f and g are the lower and the upper capacities and u is the bounding vector.

This is also a submodular flow problem with $g^u(e) := \min(g(e), u(e))$ in place of g. Using Claim 1.10 we get a strongly polynomial time algorithm that is linear in u and computes an optimal flow x and an optimal dual solution (π, z_1, z_2), that is a solution to

$$\min \sum_{X \subseteq V} \pi_X b(X) - z_1 f + z_2 g^u \tag{1.24a}$$
$$\pi \in \mathbb{R}^{2^V}, \quad z_1, z_2 \in \mathbb{R}^E \tag{1.24b}$$
$$\pi, z_1, z_2 \geq 0 \tag{1.24c}$$
$$\sum_{X : e \in \delta(X)} \pi_X - \sum_{X : e \in \varrho(X)} \pi_X - z_1(e) + z_2(e) = w(e) \quad \text{for all } e \subseteq E, \tag{1.24d}$$

where π_X denotes the component of π corresponding to the subset $X \subseteq V$.

Although π is an exponential-size vector, the optimal solution computed by the algorithm consists of at most $|E|$ non-zero elements and it is given by the set of non-zero coordinates and the corresponding values.

Finally, (π, z_1, z_2) can be transformed to a dual optimal solution (π, ν_1, ν_2, y) to (1.23), where

$$\nu_1 := z_1 \tag{1.25a}$$
$$\nu_2(e) := \begin{cases} z_2(e) & \text{if } x(e) = g(e) \\ 0 & \text{otherwise} \end{cases} \tag{1.25b}$$
$$y(e) := \begin{cases} z_2(e) & \text{if } x(e) < g(e) \\ 0 & \text{otherwise} \end{cases} \tag{1.25c}$$

To sum up, we get a strongly polynomial time algorithm for the bounded version of the submodular flow problem, so Theorem 1.1 provides a strongly polynomial time algorithm for the budgeted submodular flow problem (Problem (1.21)).

1.6 An Open Problem

Finally, we pose a natural problem to which no strongly polynomial time algorithm is known.

The *budgeted maximum matching problem* is the following. We are given a graph $G = (V, E)$ a cost and a weight function $c, w : E \longrightarrow \mathbb{R}$ on its edges and a budget constraint B and we are looking for a vector $z \geq 0$, $cz \leq B$ which minimizes the weight of the $(w - z)$-maximal weight matching.

If G is a bipartite graph, then the problem can be reduced either to the budgeted minimum cost circulation problem or to the budgeted matroid intersection problem, so it can be solved in strongly polynomial time, but the general case is still open.

According to Theorem 1.1 it would be enough to give a linear strongly polynomial time algorithm to the following *bounded maximum weight matching problem*. We are given a graph $G = (V, E)$, a weight and a capacity function $w, u : E \longrightarrow \mathbb{R}$ on its edges and the problem is to find

$$\max\{wx : x \in \mathcal{P}, x \leq u\}, \tag{1.26}$$

where $\mathcal{P}$ is matching polyhedron, that is, the convex hull of the incidence vectors of the matchings of G.

Chapter 2

LP Optimization with Additional Variables and Constraints

Let us remind the theorem due to Megiddo[Meg79] presented in Section 1.1 with a little bit more sophisticated running time estimation.

Theorem 1.3 (Restated) *Suppose that we are given a $\lambda^* \in \mathbb{R}$ through a separation algorithm $\mathcal{A}(\lambda)$ which is able to decide whether $\lambda < \lambda^*$, $\lambda = \lambda^*$ or $\lambda > \lambda^*$ by answering -1, 0 or $+1$. If $\mathcal{A}$ is* linear in λ *(see Definition 1.4), works in time T and takes at most q comparisons, then we can determine the value of λ^* in time $O(Tq)$.*

This theorem inspired a variety of applications, improvements, and extensions (see e.g. [BP95, BDK93, Meg83, NPT92, Rad98]).

In higher dimension, a *separation algorithm* for a convex set $\mathcal{P}$ is a subroutine which decides whether or not a given vector is in the set $\mathcal{P}$, and if not, gives a hyperplane that separates the given vector from $\mathcal{P}$. As an extension of Khachian's ellipsoid method [Kha80], Grötschel, Lovász and Schrijver [GLS81], Karp and Papadimitriou [KP82] and Padberg and Rao [PR81] independently showed that a linear object function can be maximized in polynomial time over any polyhedron (having only rational vertices with known binary length) given by a separation algorithm. Unfortunately this method is not strongly polynomial.

On the other hand Theorem 1.3 shows that in case when $\mathcal{P} \subset \mathbb{R}$, if we have a *linear* strongly polynomial separation algorithm, then we can optimize in strongly polynomial time. Using a similar idea, C. H. Norton, S. A. Plotkin and É. Tardos [NPT92] extended

this result to any higher (but fixed) dimension. Namely, they proved the following theorem.

Theorem 2.1 ([NPT92]) *Let a closed convex set $\mathcal{P} \in \mathbb{R}^d$ be given through a separation algorithm which is linear in its input, runs in time T and makes at most q comparisons. Then there is an algorithm which in $O(Tq^d)$ time either finds a point $x \in \mathcal{P}$ maximizing cx, or concludes that $\{cx : x \in \mathcal{P}\}$ is unbounded from above.*

In [NPT92] the authors also prove the following important consequence of Theorem 2.1.

Theorem 2.2 ([NPT92]) *Suppose that there exists an algorithm to solve the linear program $\max\{ax : Ax \leq b\}$ for arbitrary b, which runs in time T, makes at most q comparisons and which is linear in b. Then for any fixed d, there is an algorithm which runs in $O(Tq^{d+1})$ time and solves the linear program $\max\{ax + cz : Ax + Cz \leq b\}$ for any matrix C with d columns.*

The idea of the proof is to use Theorem 2.1 by transforming the problem to a maximization problem over a convex set in $\mathbb{R}^{d+1}$ for which there exists a separation algorithm constructed from the basic algorithm.

Turning to the dual formulation of a certain problem we get that constant number of additional constraints can also be handled with this method, namely we get:

Theorem 2.3 ([NPT92]) *Suppose that there exists an algorithm to solve the linear program $\max\{ax : Ax \leq b\}$ for arbitrary a, which runs in time T, makes at most q comparisons and which is linear in a. Then for any fixed d, there is an algorithm which runs in $O(Tq^{d+1})$ time and solves the linear program $\max\{ax : Ax \leq b, Cx \leq c\}$ for any vector c and matrix C with d rows.*

Note that there are no restrictions on the size of the linear program corresponding to the basic problem. Also, it doesn't have to be given explicitly, the only thing that we need is the existence of a linear algorithm which is able to solve the basic problem. Since most combinatorial optimization problems (such as matchings, b-factors or trees of a graph, flows, circulations, submodular flows, matroids, intersections of a pair of matroids etc) have — maybe exponentially large — linear programming descriptions, these theorems are quite useful tools for designing strongly polynomial algorithms, in addition to the theoretical importance that it widens the class of strongly polynomially solvable linear programs.

As an example, let us see the following problem introduced by Shahrokhi and Matula [SM88].

Problem 2.4 (Concurrent Multi-Commodity Flows) *We are given a network $G = (V, E)$ with capacities c on its arcs, and a network of required demands $R = (V, F)$ with demands r on the arcs of R. A feasible solution to this problem is a collection of non-negative flows, f_{ij} for $(i, j) \in F$, all satisfying the same percentage of the corresponding demands. The objective is to maximize this percentage.*

This problem can be easily formulated as a multi-commodity flow problem and a single additional column (see [NPT92]). The multi-commodity flow problem can be solved in strongly polynomial time using the general result of É. Tardos[Tar86]. Denote the running time of this algorithm by T. Since $q \leq T$, Theorem 2.2 provides an $O(T^3)$ time algorithm to the concurrent multi-commodity flow problem.

In several other cases, the problem has also only a small number of additional variables or constraints, typically one or two. In these cases any improvements in the exponent of the running time can be useful.

In this chapter — by a more careful application of Megiddo's technique, an algorithm that reduces the running time from $O(Tq^{d+1})$ to $O(Tq^d)$ is presented in Section 2.1. The idea of the proof is to apply Megiddo's technique directly instead of using Theorem 2.1, which makes it possible to eliminate a redundant factor in the running time.

2.1 Eliminating the additional variables

Consider the following slightly more general problem.

$$\max cx + dy \tag{2.1a}$$

subject to

$$Ax + By \leq b, \quad Ly \leq l, \quad Ey = e. \tag{2.1b}$$

Let $\mathcal{R} := \{y \in \mathbb{R}^k : Ly \leq l, Ey = e\}$. We prove, that

Theorem 2.5 *Suppose, that there exists an algorithm which solves the problem* $\max\{cx : Ax \leq b\}$*, is linear in b, runs in time T and makes at most q comparisons. Then Problem (2.1a)-(2.1b) can be solved in time $O\left((T + |L| + |E|)\, q^{\dim \mathcal{R}}\right)$, where $|L|$ and $|E|$ denotes the size of the matrix L and E, respectively.*

We prove the theorem by induction on $k := \dim \mathcal{R}$. Let us suppose that the theorem holds whenever $\dim \mathcal{R} < k$.

First let $\mathcal{A}(A, B, b, c, d, y)$ be the algorithm which gets the matrices A and B and the vectors b, c, d and y, is linear in y and gives the optimal solution x^* to the problem

$$L(y) := \max cx + dy \tag{2.2a}$$

$$Ax \leq b - By \tag{2.2b}$$

for any *fixed* y, together with the dual optimal solution, which is an optimal solution z^* to the problem

$$L(y) = \min z(b - By) + dy \tag{2.3a}$$

$$z \geq 0 \tag{2.3b}$$

$$zA = c. \tag{2.3c}$$

Clearly, if we had an oracle which could give the second part y^* of an optimal solution to Problem (2.1a)-(2.1b), we could compute a (primal) optimal solution to the problem using $\mathcal{A}$. While an optimal place of y^* seems hard to compute directly, the linearity of $\mathcal{A}$ enables us to run it in such a way that at its each step it uses only an efficiently computable partial information on the optimal place of y^*. These partial information are computed using the following subroutine.

Claim 2.6 (Comparing subroutine) *Let us be given a vector $v \in \mathbb{R}^k$ and a real number α. Then, using the induction hypothesis, we can choose a true one from the following three statements together with a certificate of its veracity.*

"$\leq$": $vy^* \leq \alpha$ holds for any optimal solution (x^*, y^*) to (2.1a)-(2.1b),

"$\geq$": $vy^* \geq \alpha$ holds for any optimal solution (x^*, y^*) to (2.1a)-(2.1b),

"$=$": there is an optimal solution (x^*, y^*) to (2.1a)-(2.1b) for which $vy^* = \alpha$.

Proof. First, using e.g. Megiddo's linear time linear programming method[Meg84] we check whether either $vy \leq \alpha$ or $vy \geq \alpha$ holds for all $y \in \mathcal{R}$ by maximizing/minimizing the function vy over the set $\mathcal{R}$. If one of these holds we return with "$\leq$" or "$\geq$" respectively and with the dual solution of the corresponding linear program as a certificate. (Note that Megiddo's algorithm can also compute the the dual variables or the certificate of

emptiness if the linear program is infeasible. See e.g. [Sch86] for more details). Then we check whether

$$vy = \alpha \tag{2.4}$$

holds for all $y \in \mathcal{R}$. If it holds, we return with "=" and also with the dual solution.

Otherwise, let $\mathcal{R}' := \{y \in \mathcal{R} : vy = \alpha\}$. Since $\dim \mathcal{R}' < \dim \mathcal{R}$, we are able to compute the primal and dual optimal solution to the system (2.1a)-(2.1b) and (2.4), so we have an optimal solution z^b, z^l, z^e, γ to the system

$$\min z^b b + z^l l + z^e e + \gamma\alpha \tag{2.5a}$$

$$z^b A = c \tag{2.5b}$$

$$z^b B + z^l L + z^e E + \gamma v = d. \tag{2.5c}$$

Then,

- If $\gamma = 0$ then z^b, z^l, z^e is also a feasible dual solution to the system (2.1a)-(2.1b) so the primal optimal solution is an optimal solution to (2.1a)-(2.1b), too. Thus we answer "=" and the vectors z^b, z^l, z^e.
- If $\gamma < 0$ then when we increase α, the optimal object value of (2.5a)-(2.5c) will strictly decrease, implying that $vy^* \leq \alpha$ for all optimal solutions x^*, y^* to (2.1a)-(2.1b). So we answer "$\leq$" and give z^b, z^l, z^e, γ as the certificate of this fact.
- Similarly, if $\gamma > 0$ then we declare "$\geq$" and also give z^b, z^l, z^e, γ.

□

Now, let

$$\mathcal{Y} := \{y^* : \exists x^* \text{ such that } (x^*, y^*) \text{ is an optimal solution to (2.1a)-(2.1b)}\}.$$

We make a new algorithm $\mathcal{A}'$, which is a modification of $\mathcal{A}$. The algorithm sometimes calls the previous comparing algorithm. Whenever this subroutine is called with a vector v and a number α it either finds an optimal solution to (2.1a)-(2.1b) together with the dual solution (so we can stop) or states whether $vy^* \leq \alpha$ or $vy^* \leq \alpha$ holds for each optimal $y^* \in \mathcal{R}$. We store these returned half-spaces. At a certain step of the algorithm let $\mathcal{Q} := \{y \in \mathcal{R} : v_i y \leq \alpha_i \ \forall i = 1, \cdots, t\}$, where v_i and α_i define the half planes returned by the comparing subroutine. Clearly, $\mathcal{Y} \subseteq \mathcal{Q}$.

Similarly to the one dimensional parametric search shown in Section 1.1.2, we first replace each element of the LAM with a pair of a real numbers and a k dimensional vector called *parametric numbers*. What we have in mind is that the parametric number (β, v) means the linear expression $\beta + vy^*$. Addition and subtraction of these parametric numbers are made component-wise, while the multiplications and divisions of these numbers are avoided by the linearity assumption on the algorithm $\mathcal{A}$.

The only undefined part of $\mathcal{A}'$ is when it makes comparisons, namely it inquires about whether $(\beta_1, v_1) \leq (\beta_2, v_2)$ or not. It means whether $\beta_1 + v_1 y^* \leq \beta_2 + v_2 y^*$ or not, which is equivalent to $(v_1 - v_2)y^* \leq \beta_2 - \beta_1$. Our aim is to give an answer which holds all $y^* \in \mathcal{Y}$, whenever it is possible. For this, first we test with Megiddo's linear-time algorithm whether either this or its negation is a consequence of the previous answers. If it is, we make decision according to this. If it is not, then we call the comparing subroutine with $v := (v_1 - v_2)$ and $\alpha := \beta_2 - \beta_1$. If the result is

"=" , then the comparing subroutine found a primal and a dual optimal solution to the system (2.1a)-(2.1b) so we can stop and return these data.

"$\leq$" , then we consider $(\beta_1, v_1) \leq (\beta_2, v_2)$.

"$\geq$" , then we consider $(\beta_1, v_1) > (\beta_2, v_2)$.

Note that in the third case the decision is right only if there is no $y^* \in \mathcal{Y}$ for which $(v_1 - v_2)y^* = \beta_2 - \beta_1$.

Now, let us run $\mathcal{A}'(A, B, b, c, y')$, where y' is the vector of parametric numbers

$$y'_i := (0, (0, \cdots, 0, \overset{i^{th}}{1}, 0 \cdots, 0)). \tag{2.6}$$

Let $\mathcal{Q}' := \{y \in \mathcal{R} : y \text{ satisfies all previous decisions}\}$, that is, the set of $y \in \mathcal{R}$ for which $\mathcal{A}'(A, B, b, c, y')$ certainly does the same as $\mathcal{A}(A, B, b, c, y)$. Our comparing method ensures that $\mathcal{Q}' \neq \emptyset$ (that is the decisions are consistent), and $\overline{\mathcal{Q}'} = \mathcal{Q}$.

At the end of its running $\mathcal{A}'$ returns a primal and a dual optimal solution x^y, z^y to (2.2a)-(2.2b) as *a linear function of* y, i.e. it gives vectors x_0, z_0 and matrices X, Z, for which $x^y := x_0 + Xy$ and $z^y := z_0 + yZ$. These solutions must be right for all $y \in \mathcal{Q}'$, so they are also right for all $y \in \overline{\mathcal{Q}'} = \mathcal{Q}$, thus $L(y) = cx^y + dy$ for all $y \in \mathcal{Q}$.

Obviously $cx^y + dy$ is a linear function in y, namely $cx^y + dy = cx_0 + (cX + d)y$.

Claim 2.7 *For all $y' \in \mathcal{Q}$, the functions $f_{y'}(y) := (d - z^{y'}B)y + z^{y'}b$ are equal to $L(y)$ for all $y \in \mathcal{Q}$.*

Proof. First, suppose that $y' \in \text{rel int}\mathcal{Q}'$ and $y \in \mathcal{Q}'$. Then there exists $\varepsilon > 0$, such that $y_1 := y' + \varepsilon(y' - y) \in \mathcal{Q}'$. Since z^y is an optimal solution to (2.3a)-(2.3c), it follows that $f_{y'}(y) \geq f_y(y) = L(y)$ and $f_{y'}(y_1) \geq f_{y_1}(y_1) = L(y_1)$. From this we get that

$$f_{y'}(y) = f_{y'}(y') + z^{y'}B(y - y') = L(y') + z^{y'}B(y - y') \tag{2.7}$$

$$\geq L(y) = L(y') + (cX + d)(y - y'), \tag{2.8}$$

yielding that

$$z^{y'}B(y - y') \geq (cX + d)(y - y'). \tag{2.9}$$

We get the same way that $z^{y'}B(y_1 - y') \geq (cX + d)(y_1 - y')$, followed by

$$z^{y'}B\varepsilon(y - y') \leq (cX + d)\varepsilon(y - y'). \tag{2.10}$$

(2.9) and (2.10) together give that $z^{y'}B(y - y') = (cX + d)(y - y')$, so $f_{y'}(y) = L(y)$. For arbitrary $y, y' \in \mathcal{Q}$ the claim follows from the continuity of z^y and $L(y)$. □

□

On the other hand $\mathcal{Y} \subseteq \mathcal{Q}$, so a vector y^* maximizing the function $L(y)$ (i.e. a $y^* \in \mathcal{Y}$) can be obtained by maximizing the function $(cX + d)y$ over the set $\mathcal{Q} = \{y : Ly \leq l, Ey = e, Vy \leq a\}$, where the rows of V and a correspond to the calls of the comparing subroutine. It can also be done with Megiddo's linear time algorithm. Now, we call the original $\mathcal{A}$ with the parameters A, B, b, c, y^* and let x^* and z^* be the returned primal and dual solution. Then x^*, y^* is an *optimal primal solution* to the problem (2.1a)-(2.1b).

The only thing that we still have to do is to compute the *dual optimal solution* to Problem (2.1a)-(2.1b). Namely, we need vectors z_b^*, z_l^*, z_e^* for which

$$z_b^*, z_l^* \geq 0 \tag{2.11a}$$

$$z_b^* A = c \tag{2.11b}$$

$$z_b^* B + z_l^* L + z_e^* E = d \tag{2.11c}$$

$$z_b^* b + z_l^* l + z_e^* e \leq cx^* + dy^* \tag{2.11d}$$

Let $w := d - z^{y^*}B$. According to Claim 2.7 y^* is an optimal primal solution to the following problem.

$$\max wy \tag{2.12a}$$

$$Vy \leq a \tag{2.12b}$$

$$Ly \leq l \tag{2.12c}$$

$$Ey = e. \tag{2.12d}$$

Let z_b^M, z_l^M, z_e^M be the optimal dual solution to the previous problem, that is the optimal solution to the problem

$$\min z_1^M a + z_2^M l + z_3^M e \tag{2.13a}$$

$$z_1^M, z_2^M \geq 0 \tag{2.13b}$$

$$z_1^M V + z_2^M L + z_3^M E = d - z^{y^*} B. \tag{2.13c}$$

Now, let us define the following vectors

$$z_b^* := \frac{1}{\delta}\left(z^{y^*} - \sum_i \frac{z_1^M(i)}{\gamma_i} z_i^b\right), \tag{2.14}$$

$$z_l^* := \frac{1}{\delta}\left(z_2^M - \sum_i \frac{z_1^M(i)}{\gamma_i} z_i^l\right), \tag{2.15}$$

$$z_e^* := \frac{1}{\delta}\left(z_3^M - \sum_i \frac{z_1^M(i)}{\gamma_i} z_i^e\right), \tag{2.16}$$

where z_i^b, z_i^l, z_i^e are the solutions to (2.5a)-(2.5c) corresponding to the rows of the matrix V and $\delta := 1 - \sum_i \frac{z_1^M(i)}{\gamma_i}$. Note, that $\gamma_i < 0$ for each i. One can check, that (z_b^*, z_l^*, z_e^*) is a feasible solution to (2.11a)-(2.11c) and fulfills the complementary slackness condition, thus (2.11d) also holds.

Finally, the running time of the algorithm is the sum of the time used by $\mathcal{A}'$ itself and the time required by the comparisons made by it. Each comparison involves the same optimization task, but over a set $\mathcal{R}'$, the dimension of which is smaller by at least one than that of $\mathcal{R}$. So, by the induction hypothesis each comparison runs in time $O\left(|L| + |E| + (T + |L| + |E|)\, q^{\dim \mathcal{R}-1}\right)$. and makes at most $q^{\dim \mathcal{R}-1}$ comparisons. Since $\mathcal{A}'$ makes at most q comparisons, the total running time is $O\Big(T + q\left(|L| + |E| + (T + |L| + |E|)\, q^{\dim \mathcal{R}-1}\right)\Big)$ and the algorithm makes at most $q \cdot q^{\dim \mathcal{R}-1}$ comparisons, yielding the required running time. □

Chapter 3

On Resource Constrained Optimization Problems

This chapter shows that a method that has long been used to solve *Resource Constrained Optimization Problems* and found extremely effective in practice, is effective in the theoretical sense as well, it is proved to be strongly polynomial. In the special case of *Resource Constrained Shortest Path Problem* a better running time estimation is also presented.

In order to define this class of problems, first let us consider an underlying set E, a *cost function* $c : E \longrightarrow \mathbb{R}_+$ and an abstract optimization problem

$$\min\{\sum_{e\in P} c(e) : P \in \mathcal{P}\}, \tag{3.1}$$

where $\mathcal{P} \subseteq 2^E$ denotes the set of the feasible solutions. We refer this problem a *basic problem* in this chapter.

The corresponding *constrained optimization problem* is the following. Let $d : E \longrightarrow \mathbb{R}_+$ be another given weighting called *delay*, and $\Delta \in \mathbb{R}_+$ a given constant called *delay constraint*. With these notations we are looking for the value

$$\min\{\sum_{e\in P} c(e) : P \in \mathcal{P}, \quad \sum_{e\in P} d(e) \leq \Delta\}. \tag{3.2}$$

An important example for this is the *Constrained Shortest Path Problem*. Assume that a network is given as a directed, connected graph $G = (V, E)$, where V represents the set of nodes, and E represents the set of directed links. Each link $e \in E$ is characterized by two nonnegative values, a cost $c(e)$ and a delay $d(e)$. With a given delay constraint $\Delta \in \mathbb{R}_+$

and two given nodes $s, t \in V$ the task is to find a least cost path P between s and t with the side constraint that the delay of the path is less then Δ.

One can define the *Constrained Minimum Cost Perfect Matching Problem* and the *Constrained Minimum Cost Spanning Tree Problem* in the same way.

Although their unconstrained versions are easy to solve, the three problems mentioned above are $\mathcal{NP}$-hard (see e.g. [GJ79]). A usual way to find near optimal solutions to these problems is to get rid of the additional constraint by using Lagrangian relaxation. In this way the constrained problem turns into a maximization of a one dimensional concave function (see Section 3.1).

A simple way to find the optimum of the relaxed problem is to use binary search, which is polynomial for integer costs and delays (see [Zie01]).

Instead of using binary search another simple and practically even more effective method — described in Section 3.2 — has been found and applied independently by several authors. After [HZ80] it is sometimes called *Handler-Zang method*. The same method was used in [BG96] making some people call it *BM method*. Other papers aim at further improving either on the running time in practical cases [JSMR01] or on the quality of the found solutions [MZ00].

Although this method has been turned to be particularly efficient in practical applications, the worst case running time of this method was unknown for a long time.

Mehlhorn and Ziegelmann showed that for the Constrained Shortest Path problem the Handler-Zang algorithm is polynomial for integer costs and delays. If $c(e) \in [0, \cdots, C]$ and $d(e) \in [0, \cdots, R]$ for each $e \in E$, then it will terminate after $O\big(\log(|V|RC)\big)$ iterations. They also presented examples showing that this running time is tight for small costs and delays (i.e. if $R \leq |V|$ and $C \leq |V|$).

One may observe that this problem can be transformed to an extension of the LCFO (Least Cost Fractional Optimization) problem discussed by Radzik in [Rad93]. Moreover, the strongly polynomial solution method proposed in [Rad93] turns out to be equivalent with the Handler-Zang algorithm in this case, showing that the number of the iterations made by the Handler-Zang algorithm does not depend on the range of the cost and the delay functions, so this method is actually *strongly polynomial* if the basic problem can be solved in strongly polynomial time. This also shows that Mehlhorn's and Ziegelmann's bound on the running time is not tight for large costs or delays.

In this chapter first we show that the Handler-Zang algorithm takes $O\big(|E|^2 \log(|E|)\big)$

iterations for arbitrary constrained optimization problem. The proof is a more careful application of the technique proposed in [Rad98], and improves the bound obtained from [Rad93] by a factor $O(\log(|E|))$.

For the Constrained Shortest Path problem an even better bound is shown in Section 3.3.2, the number of iterations is proved to be $O\big(|E|\log^2(|E|)\big)$ in this case.

3.1 Lagrangian Relaxation

Lagrange Relaxation [HK70, HK71] is a common technique for calculating lower bounds, and finding good solutions to hard optimization problem. This section shows how it can be applied to resource constrained optimization problems.

From now on we assume that there exists an algorithm $\mathcal{A}(c)$ which runs in time T and solves Problem (3.1) for any given cost function c.

Intuitively the presented Lagrangian relaxation method is based on the heuristic of minimizing $c_\lambda := c + \lambda \cdot d$ modified cost function over $\mathcal{P}$ for an appropriate λ. For a given (fixed) λ we can easily calculate the minimal solution $p_\lambda \in \mathcal{P}$ of the basic problem. If for $\lambda = 0$ we get that $d(p_\lambda) \leq \Delta$, then we found an optimal solution for the constrained problem as well. If $d(p_\lambda) > \Delta$ we must increase λ in order to increase the dominance of delay in the modified cost function until the optimal solution with respect to c_λ suits the delay requirements.

Now, we show how the Lagrangian relaxation helps us to find the value of λ that gives the best result. Moreover, the algorithm will give an upper bound on the badness of the solution as a byproduct, based on the following well known Claim.

Claim 3.1 *Let*

$$L(\lambda) := \min\{c_\lambda(p) : p \in \mathcal{P}\} - \lambda\Delta. \tag{3.3}$$

Then $L(\lambda)$ is a lower bound *to problem (3.2) for any $\lambda \geq 0$.*

Proof. Let p^* denote an optimal solution of (3.2). Then

$$\begin{aligned} L(\lambda) &= \min\{c_\lambda(p) : p \in \mathcal{P}\} - \lambda\Delta \\ &\leq c_\lambda(p^*) - \lambda\Delta \\ &= c(p^*) + \lambda(d(p^*) - \Delta) \leq c(p^*) \end{aligned}$$

proves the claim. □

To obtain the best lower bound we need to maximize the function $L(\lambda)$, that is we are looking for the value

$$L^* := \max_{\lambda \geq 0} L(\lambda), \tag{3.4}$$

and the maximizing λ^*. Now, some more properties of the function $L(\lambda)$ are given.

Claim 3.2 *L is a concave piecewise linear function, namely the minimum of the linear functions $c(p) + \lambda(d(p) - \Delta)$ for all $p \in \mathcal{P}$.* □

Claim 3.3 *For any $\lambda \geq 0$ and c_λ-minimal solution $p_\lambda \in \mathcal{P}$, $d(p_\lambda) - \Delta$ is a subgradient of L in the point λ.*

Proof. By the definition of subgradient one have to show that $L(x) \leq L(\lambda) + (x - \lambda)(d(p_\lambda) - \Delta)$ holds for any x, which is proved by the following calculation.

$$\begin{aligned} L(\lambda) + (x - \lambda)(d(p_\lambda) - \Delta) =& c(p_\lambda) + \lambda(d(p_\lambda) - \Delta) + (x - \lambda)(d(p_\lambda) - \Delta) \\ =& c(p_\lambda) + x(d(p_\lambda) - \Delta) \geq L(x) \end{aligned} \tag{3.5}$$

□

This gives us the following

Claim 3.4 *Whenever $\lambda < \lambda^*$, then $d(p_\lambda) \geq \Delta$ and if $\lambda > \lambda^*$, then $d(p_\lambda) \leq \Delta$ for each c_λ-minimal solution p_λ.* □

Claim 3.5 *A value λ maximizes the function $L(\lambda)$* if and only if *there are solutions p_c and p_d which are both c_{λ^*}-minimal and for which $d(p_c) \geq \Delta$ and $d(p_d) \leq \Delta$. (p_c and p_d can be the same, in this case $d(p_d) = d(p_c) = \Delta$.)* □

The Handler-Zang algorithm will give these solutions along with λ^*.

Claim 3.6 *Let $0 \leq \lambda_1 < \lambda_2$, and $p_{\lambda_1}, p_{\lambda_2} \in P$ λ_1-minimal and λ_2-minimal solutions. Then $c(p_{\lambda_1}) \leq c(p_{\lambda_2})$ and $d(p_{\lambda_1}) \geq d(p_{\lambda_2})$.*

The inequality for d immediately follows from Claim 3.2 and Claim 3.3. The following calculation proves the inequality for c.

$$\begin{aligned} c(p_{\lambda_1}) &= L(\lambda_1) - \lambda_1(d(p_{\lambda_1}) - \Delta) \\ &\leq L(\lambda_2) + (\lambda_1 - \lambda_2)(d(p_{\lambda_2}) - \Delta) - \lambda_1(d(p_{\lambda_1}) - \Delta) \\ &= c(p_{\lambda_2}) + \lambda_2(d(p_{\lambda_2}) - \Delta) + (\lambda_1 - \lambda_2)(d(p_{\lambda_2}) - \Delta) - \lambda_1(d(p_{\lambda_1}) - \Delta) \\ &= c(p_{\lambda_2}) + \lambda_1(d(p_{\lambda_2}) - d(p_{\lambda_1})) \leq c(p_{\lambda_2}) \end{aligned} \tag{3.6}$$

□

These two latter Claims together means that the λ^* that maximizes the function $L(\lambda)$ gives the best modified cost function, that is λ^* is the smallest value for which there exists a c_{λ^*}-minimal solution p_d which satisfies the delay constraint.

3.2 The Handler-Zang method

In this section the Handler-Zang algorithm is described (see Figure 3.1 for the pseudocode).

1. In the first step the algorithm sets $\lambda = 0$. It calculates an optimal solution with respect to the modified cost function c_λ. It means that the algorithm finds the c-minimal solution. If this solution meets the delay requirement Δ, it is an optimal solution of (3.2), and the algorithm stops.

2. Otherwise the algorithm stores the solution as the best solution that does not satisfy the delay requirement Δ (it is denoted by p_c in the following), and checks whether an appropriate solution exists or not: calculates the d-minimal solution. If the obtained solution suits the delay requirement, a proper solution exists, so the algorithm stores it as the best feasible solution found till now (denoted by p_d). Otherwise there is no solution that fulfills the delay requirement, so the algorithm stops.

   In the further steps we obtain the optimal λ, through repeatedly updating either p_c or p_d with a new solution.

3. Let us see the current solutions p_c and p_d. If for a certain λ, both p_c and p_d are c_λ-minimal, then using Claim 3.5, this λ maximizes $L(\lambda)$. But it can be true only if $c_\lambda(p_c) = c_\lambda(p_d)$, from which we get that the only possible λ is

$$\lambda := \frac{c(p_c) - c(p_d)}{d(p_d) - d(p_c)}. \tag{3.7}$$

So, we set it as the new candidate for the optimal solution. Then we find a c_λ-minimal solution r. If $c_\lambda(r) = c_\lambda(p_c)$ then p_c and p_d are also c_λ-minimal, so we are done by setting $\lambda^* = \lambda$ and returning p_d. If $c_\lambda(r) < c_\lambda(p_c)$ then we replace either p_c or p_d with r according to whether it fails or fulfills the delay constraint, and repeat this step.

procedure HanglerZang(s, t, c, d, Δ)
  $p_c^1 := \mathcal{A}(s, t, c)$
  if $d(p_c^1) \leq \Delta$ **then return** p_c^1
  $p_d^1 := \mathcal{A}(s, t, d)$
  if $d(p_d^1) > \Delta$
    then return "There is no solution"
  $\lambda_c^1 := 0; \quad \lambda_d^1 := \infty$
  $i := 1$
  repeat
    $\lambda^{i+1} := \frac{c(p_c^i) - c(p_d^i)}{d(p_d^i) - d(p_c^i)}$
    $r := \mathcal{A}(s, t, c_{\lambda^{i+1}})$
    if $c_{\lambda^{i+1}}(r) = c_{\lambda^{i+1}}(p_c)$ **then return** p_d^i
      else if $d(r) \leq \Delta$ **then** $p_d^{i+1} := r; \quad p_c^{i+1} := p_c^i; \quad \lambda_d^{i+1} := \lambda^{i+1}; \quad \lambda_c^{i+1} := \lambda_c^i$
      else $p_c^{i+1} := r; \quad p_d^{i+1} := p_d^i; \quad \lambda_c^{i+1} := \lambda^{i+1}; \quad \lambda_d^{i+1} := \lambda_d^i$
  end repeat
end procedure,

where $\mathcal{A}(c)$ returns a c-minimal solution to the basic problem.

Figure 3.1: The Handler-Zang algorithm

3.3 Running time of the algorithm

In the following we refer to the solutions and values computed by the algorithm according to Figure 3.1.

First of all, obviously

$$d(p_c^1) \geq d(p_c^2) \geq d(p_c^3) \geq \cdots > \Delta \tag{3.8}$$

and

$$d(p_d^1) \leq d(p_d^2) \leq d(p_d^3) \leq \cdots \leq \Delta, \tag{3.9}$$

and either $d(p_c^i) > d(p_c^{i+1})$ or $d(p_d^i) < d(p_d^{i+1})$ for any i. Since there are only finite number of different solutions, the algorithm finds the optimal λ in finite number of steps.

In this section the following stronger result will be proved.

Theorem 3.7 *The Handler-Zang algorithm terminates after $O(|E|^2 \log |E|)$ iterations, so the running time of the algorithm is $O(T|E|^2 \log |E|)$.* □

The proof presented below is based on the idea of the proof of the strong polynomiality of the so-called Newton-method for fractional optimization problems[Rad98].

3.3.1 Strong Polynomiality

The bound on the running time is based on the following theorem of Goemans (published by Radzik in [Rad98]) stating that a geometrically decreasing sequence of numbers constructed in a certain restricted way cannot be exponentionally long.

Theorem 3.8 *Let $\mathbf{c} = (c_1, c_2, \cdots, c_n)$ be an n dimensional vector with real components, and let $\mathbf{y_1}, \mathbf{y_2}, \cdots, \mathbf{y_q}$ be vectors from $\{-1, 0, 1\}^n$. If for all $i = 1, 2, \cdots, q-1$*

$$0 < \mathbf{y_{i+1}c} \leq \frac{1}{2}\mathbf{y_i c},$$

then $q = O(n \log n)$. □

For sake of completeness let us sketch the elegant proof of this theorem.

Proof. Assume that we are given a sequence $\mathbf{y_1}, \mathbf{y_2}, \cdots, \mathbf{y_q}$ of vectors meeting with the requirement of the theorem and let us consider the following system of linear inequalities.

$$(2\mathbf{y_{i+1}} - \mathbf{y_i})\mathbf{z} \leq 0 \qquad \forall i = 1, 2, \cdots, q-1 \tag{3.10}$$

$$\mathbf{y_q z} = 1, \tag{3.11}$$

where $\mathbf{z}$ is the vector of the variables. This system clearly has a solution, namely the vector

$$\mathbf{z} := \frac{1}{\mathbf{y_q c}}\mathbf{c} \tag{3.12}$$

is an appropriate choice. The theory of linear programming says that then a basic solution $\mathbf{z}'$ must also be exists. It is also well-known that a basic solution is obtained as a solution of linear equations forming from the coefficients of (3.10) and (3.11). From *Cramer's Rule* we get that the components of $\mathbf{z}'$ is obtained as a quotient of two determinant, namely

$$\mathbf{z}'_i := \frac{det(A_i)}{det(A)}, \tag{3.13}$$

where all element of both A_i and A are integer with an absolute value no more than 3. Since $det(A)$ is a nonzero integer, a rough estimation shows that $|\mathbf{z}'_i| \leq n! \cdot 3^n$, thus $\mathbf{y_{i+1}}\mathbf{z}' \leq n \cdot n! \cdot 3^n$. On the other hand, $\mathbf{y_{i+1}}\mathbf{z}' \leq \frac{1}{2}\mathbf{y_i}\mathbf{z}'$ and $\mathbf{y_q}\mathbf{z} = 1$, providing that

$$q \leq \log_2(n \cdot n! \cdot 3^n) = O(n \log n) \tag{3.14}$$

□

For the sake of simplicity let us introduce the following notations. $c_c^i := c(p_c^i)$, $c_d^i := c(p_d^i)$, $d_c^i := d(p_c^i)$, $d_d^i := d(p_d^i)$, $L_c^i := c(p_c^i) + \lambda_c^i(d(p_c^i) - \Delta)$ and $L_d^i := c(p_d^i) + \lambda_d^i(d(p_d^i) - \Delta)$.

Claim 3.9 *Suppose that $p_c^{i+1} \neq p_c^i$. Then*

$$\frac{c_d^i - c_c^i}{d_c^i - d_d^i} \geq \frac{L^* - c_c^i}{d_c^i - \Delta}. \tag{3.15}$$

Proof. From the definition of the function $L(\lambda)$ we get that for any $\lambda \geq 0$,

$$L(\lambda) \leq c_c^i + \lambda(d_c^i - \Delta) \tag{3.16}$$

and

$$L(\lambda) \leq c_d^i + \lambda(d_d^i - \Delta). \tag{3.17}$$

By maximizing the value "$L(\lambda)$" subject to the above two conditions we get that

$$L^* = \max_{\lambda \geq 0} L(\lambda) \leq c_c^i + \frac{c_d^i - c_c^i}{d_c^i - d_d^i}(d_c^i - \Delta), \tag{3.18}$$

which is equivalent with (3.15). □

Lemma 3.10 *Suppose that $p_c^{i+1} \neq p_c^i$. Then*

$$\frac{L^* - L_c^{i+1}}{L^* - L_c^i} + \frac{d_c^{i+1} - \Delta}{d_c^i - \Delta} \leq 1. \tag{3.19}$$

Proof.

$$\begin{aligned}
L_c^i &= c_c^i + \lambda_c^i(d_c^i - \Delta) \\
&\leq c_c^{i+1} + \lambda_c^i(d_c^{i+1} - \Delta) \\
&= L_c^{i+1} - \lambda_c^{i+1}(d_c^{i+1} - \Delta) + \lambda_c^i(d_c^{i+1} - \Delta) \\
&= L_c^{i+1} + (\lambda_c^i - \lambda_c^{i+1})(d_c^{i+1} - \Delta) \\
&= L_c^{i+1} + (\frac{L_c^i - c_c^i}{d_c^i - \Delta} - \frac{c_d^i - c_c^i}{d_c^i - d_d^i})(d_c^{i+1} - \Delta) \\
&\leq L_c^{i+1} + (\frac{L_c^i - c_c^i}{d_c^i - \Delta} - \frac{L^* - c_c^i}{d_c^i - \Delta})(d_c^{i+1} - \Delta) \\
&= L_c^{i+1} + (\frac{L_c^i - L^*}{d_c^i - \Delta})(d_c^{i+1} - \Delta) \\
&= L_c^{i+1} + (L_c^i - L^*)\frac{d_c^{i+1} - \Delta}{d_c^i - \Delta},
\end{aligned}$$

followed by $L^* - L_c^i \geq (L^* - L_c^{i+1}) + (L^* - L_c^i)\frac{d_c^{i+1} - \Delta}{d_c^i - \Delta}$, which is equivalent with the claim.

□

Corollary 3.11 *Suppose that $p_c^{i+1} \neq p_c^i$. Then*

$$\frac{(L^* - L_c^{i+1})(d_c^{i+1} - \Delta)}{(L^* - L_c^i)(d_c^i - \Delta)} \leq \frac{1}{4}. \tag{3.20}$$

□

By the same token, the following can also be proved.

Corollary 3.12 *Suppose that $p_d^{i+1} \neq p_d^i$. Then*

$$\frac{(L^* - L_d^{i+1})(\Delta - d_d^{i+1})}{(L^* - L_d^i)(\Delta - d_d^i)} \leq \frac{1}{4}. \tag{3.21}$$

□

From Corollary 3.11, we get that

Corollary 3.13 *Suppose that $p_c^{i+1} \neq p_c^i$. Then at least one of the followings are satisfied.*

(a) $L^* - L_c^{i+1} \leq \frac{1}{2}(L^* - L_c^i)$,

(b) $d_c^{i+1} - \Delta \leq \frac{1}{2}(d_c^i - \Delta)$.

□

First note that both sequences $d_c^i - \Delta$ and $L^* - L_c^i$ are monotonically decreasing. So, from Lemma 3.8 we get that (b) can be true at most $O(|E| \log |E|)$ times. To estimate how much times (a) can be true, let

$$s_c^i := (L^* - L_c^i)(d_c^{i-1} - d_d^{i-1}) = (L^* - c_c^i)(d_c^{i-1} - d_d^{i-1}) - (c_d^{i-1} - c_c^{i-1})(d_c^i - \Delta) > 0, \quad (3.22)$$

where $\gamma^i := c_d^i - c_c^i$ and $\delta^i := d_c^i - d_d^i$.

It can be seen from the definition, that s_c^i is monotonically decreasing and if (a) satisfies, then

$$s_c^{i+1} \leq \frac{1}{2} s_c^i. \quad (3.23)$$

The second form of s_c^i shows that there exist vectors $\mathbf{c} \in \mathbb{R}^{2(|E|+1)^2}$ and $\mathbf{y_i} \in \{-1, 0, 1\}^{2(|E|+1)^2}$ such that $s_c^i = \mathbf{y_i c}$ for all i. Indeed, $\mathbf{c}$ is an appropriate choice if it consists of the following constants.

$$\Big\{\alpha \cdot \beta \cdot \zeta : \quad \alpha \in \{1,2\}, \quad \beta \in \{c(e) : e \in E\} \cup \{L^*\}, \quad \zeta \in \{d(e) : e \in E\} \cup \{\Delta\}\Big\}$$

Thus, using Lemma 3.8 we get that (a) can be true at most $O(|E|^2 \log |E|)$ times. So, the sequence $p_c^1, p_c^2, p_c^3, \cdots$ consists of at most $O(|E|^2 \log |E|)$ different solutions.

The same can be proved for the sequence $p_d^1, p_d^2, p_d^3, \cdots$, so we obtained that

Theorem 3.14 *The Handler-Zang algorithm terminates after* $O(|E|^2 \log |E|)$ *iterations.* □□

3.3.2 Better Bound in Case of Constrained Shortest Path Problem

This section shows that in the case of Constrained Shortest Path Problem a better bound can actually be proved.

In this case we are given a directed graph $G = (V, E)$, two predefined nodes $s, t \in V$ and $\mathcal{P}$ denotes the set of s—t paths. Let $n := |V|$ and $m := |E|$.

We use the notations of the previous section with the exception that from now on we consider only the subsequences of $p_c^1, p_c^2, \cdots$ and $p_d^1, p_d^2, \cdots$ that contain each different

path exactly ones. In the hope that it will not be confusing, the same notations are used for these subsequences and also for the corresponding values $c_c^i, c_d^i, d_c^i, d_c^i, \lambda_c^i, \lambda_d^i, L_c^i$ and L_d^i. The monotonicity of these latter sequences still hold true of course, however one has to be careful that λ_c^{i+1} is not necessarily equal to $\frac{c_c^i - c_d^i}{d_d^i - d_c^i}$. With these new notations the following version of Claim 3.11 and Claim 3.12 holds.

Claim 3.15 *For each i,*

$$\frac{(L^* - L_c^{i+1})(d_c^{i+1} - \Delta)}{(L^* - L_c^i)(d_c^i - \Delta)} \leq \frac{1}{4} \tag{3.24}$$

and

$$\frac{(L^* - L_d^{i+1})(\Delta - d_d^{i+1})}{(L^* - L_d^i)(\Delta - d_d^i)} \leq \frac{1}{4}. \tag{3.25}$$

□

The following claim can be easily seen.

Claim 3.16 *For any i and $\lambda \geq \lambda_c^{i+2}$,*

$$(c_c^i - \lambda(d_c^i - \Delta)) - L^* \geq L^* - L_c^{i+1}$$

or equivalently

$$c_c^i - \lambda(d_c^i - \Delta) \geq 2L^* - L_c^{i+1}$$

□

Definition 3.17 *Let edge e be called* i-essential *if $e \in p_c^i \cup p_c^{i+1} \cup p_c^{i+2} \cup \cdots$.*

Lemma 3.18 *Let $k := \lceil \frac{\log_2 n + 1}{2} \rceil$. Then for any i at least one of the followings hold.*

(a) $(d_c^{i+k} - \Delta) \leq \frac{1}{2}(d_c^i - \Delta)$,

(b) there exists an i-essential edge e that is not $(i+k)$-essential.

Proof. Let us suppose that (a) does not hold, i.e.

$$(d_c^{i+k} - \Delta) > \frac{1}{2}(d_c^i - \Delta) > \frac{1}{2}(d_c^{i+1} - \Delta).$$

Since

$$(L^* - L_c^{i+k})(d_c^{i+k} - \Delta) \leq \frac{1}{2n}(L^* - L_c^{i+1})(d_c^{i+1} - \Delta),$$

we get that

$$(L^* - L_c^{i+k}) \leq \frac{1}{n}(L^* - L_c^{i+1}).$$

Now, let us define the following auxiliary cost function on the edges.

$$\tilde{c}(\vec{uv}) := c_{\lambda_c^{i+k}}(\vec{uv}) - \ell(v) + \ell(v),$$

where

$$\ell(u) := \min\{c_{\lambda_c^{i+k}}(p) : p \in \mathcal{P}_{s,u}\}.$$

Note that $\tilde{c}(e) \geq 0$ for each edge e and $\ell(t) = L_c^{i+k} + \lambda_c^{i+k}\Delta$. Moreover, $\tilde{c}(p) = c_{\lambda_c^{i+k}}(p) - \ell(v)$ for any $p \in \mathcal{P}_{s,v}$.

Using Claim 3.16 we get that

$$\begin{aligned}
\tilde{c}(p_c^i) &= c_{\lambda_c^{i+k}}(p_c^i) - \ell(t) = c(p_c^i) + \lambda_c^{i+k} d(p_c^i) - \ell(t) \\
&= c_c^i + \lambda_c^{i+k}(d_c^i - \Delta) + \lambda_c^{i+k}\Delta - \ell(t) \\
&\geq 2L^* - L_c^{i+1} + \lambda_c^{i+k}\Delta - \ell(t) \\
&> n(L^* - L_c^{i+k}) + L^* + \lambda_c^{i+k}\Delta - \ell(t) \\
&= n(L^* - L_c^{i+k}) + L^* - L_c^{i+k} = (n+1)(L^* - L_c^{i+k}).
\end{aligned}$$

Thus, there exists an edge $e \in p_c^i$ such that $\tilde{c}(e) > L^* - L_c^{i+k}$. This particular edge e is of course i-essential. To finish the proof, we show that e is not $(i+k)$-essential. To the contrary, suppose that $e \in p_c^j$ for some $j \geq i + k$. Then

$$\begin{aligned}
L^* &> c_{\lambda_c^j}(p_c^j) - \lambda_c^j\Delta \\
&\geq c_{\lambda_c^{i+k}}(p_c^j) - \lambda_c^{i+k}\Delta \\
&= \tilde{c}(p_c^j) + \ell(t) - \lambda_c^{i+k}\Delta \\
&\geq \tilde{c}(e) + \ell(t) - \lambda_c^{i+k}\Delta \\
&= \tilde{c}(e) + L_c^{i+k} > L^*
\end{aligned}$$

shows the contradiction. □

Corollary 3.19 *The length of sequence $p_c^1, p_c^2, p_c^3, \cdots$ is $O(m \log^2 m)$.*

Proof. Let $k := \lceil \frac{\log_2 n+1}{2} \rceil$ and let us consider the sequence $i = k, 2k, 3k, \cdots$. According to the previous lemma, at these iteration, either the value $d_c^i - \Delta$ at least halves or the number

of the essential edges decreases. Since $d_c^i - \Delta$ is monotonically decreasing, Lemma 3.8 shows that it can be halved at most $O(m \log m)$ times. On the other hand, we have at most m essential edges at the beginning, so the number of these steps is $O(m \log m + m) = O(m \log m)$. From this the statement follows, because $k = O(\log m)$. □

Again in the same way we get

Lemma 3.20 *Let* $k := \lceil \frac{\log_2 n+1}{2} \rceil$. *Then for any* i *at least one of the followings hold.*

(a) $(\Delta - d_d^{i+k}) \leq \frac{1}{2}(\Delta - d_d^i)$,

(b) there exists an i*-d-essential edge* e *that is not* $(i+k)$*-essential,*

where an edge e *is called* i*-d-*essential *if* $e \in p_d^i \cup p_d^{i+1} \cup p_d^{i+2} \cup \cdots$. □

and

Corollary 3.21 *The length of sequence* $p_d^1, p_d^2, p_d^3, \cdots$ *is* $O(m \log^2 m)$. □

To sum up, Corollary 3.19 and Corollary 3.21 together gives the following. (The running time of the Dijkstra algorithm for finding shortest path in a directed graph is $O(m + n \log n)$.)

Theorem 3.22 *In case of Constrained Shortest Path Problem, the Handler-Zang algorithm terminates after* $O(m \log^2 m)$ *iterations, thus the full running time of the algorithm is* $O(m^2 \log^2 m + mn \log^3 m)$. □□

3.4 Constrained Shortest Paths in Telecommunication Networks

The Constrained Shortest Path Problem has a particular importance in telecommunication networks, especially in providing *Quality of Service (QoS)* guaranteed routing. An important requirement of today and future telecommunication networks is that they must be able to carry different kind of traffic — voice, video, web browsing etc. — simultaneously on the same platform. The quality of the transport is described by several QoS parameters such as the available/required *bandwidth*, *delay*, *packet loss ratio*, *jitter* (the measurement of the unevenness of data transport) etc. Different types of traffic require very diverging

transport characteristic. Therefore a fundamental problem of QoS enabled routing is to choose a source-destination path with meets more than one QoS requirements.

The QoS parameters can be categorised into two types. For the *bottleneck type* parameters (e.g. the available bandwidth), the quality of a path is determined by quality of the worst transport component on the path. In case of *additive type* parameters (e.g. the delay), the quality of a path is the sum of the quality parameter of the every transport component on the path. The bottleneck type parameters can be handled quite easily, we can simply remove the inappropriate network elements from the graph representation of the network when the routing path is computed. On the other hand, if we have more than one additive constraints, the problem becomes $\mathcal{NP}$-hard.

We can also formulate this as an optimization problem: we want to minimize one of the QoS parameter while keeping another one under a certain limit. This is just the same problem that was discussed in the sections before. In telecommunication context the abstract parameter to be minimized is called "cost" while the second one is called "delay", therefore this problem is often referred as *Delay Constrained Least Cost Path problem (DCLC)*.

Note that the running time is a crucial point here, because a router has to compute routing paths to a lot of destinations with several QoS constraint, and these paths must be updated whenever the network state changes. Therefore an algorithm is practically acceptable only if its running time is more or less comparable to the Dijkstra algorithm.

As an algorithm for the DCLC problem is a fundamental building block of QoS routing methods, several attempts have been made to develop a method that is able to find good solutions in reasonable time. In the next section we give a short overview on the proposed solutions can be found in the literature — the Lagrange relaxation based method shown in the first part of this chapter is of course one of them. Then, Section 3.4.2 shows two improvements of the Handles-Zang method that can further improve its performance (running time) in practical working scenarios. Finally, Section 3.4.3 compare this method with the other existing solution through simulation.

Following the terminology of [JSMR01], we refer the routing method using Lagrange relaxation and the Handler-Zang method as *Lagrange Relaxation based Aggregated Cost (LARAC)* method.

3.4.1 Related Work on Constrained Shortest Paths

In this section we give a survey on the different routing algorithms proposed in the literature. Up to our best knowledge there is no widely accepted algorithm that can give the optimal solution of the **DCLC** QoS routing problem in polynomial time.

Widyono [Wid94] proposed a routing method that gives the optimal path —having the lowest possible cost without violating the delay constraint—, unfortunately with an exponential running time. The *Constrained Bellman-Ford (CBF)* routing algorithm performs a breadth-first search, discovering paths of monotonically increasing delay while recording and updating lowest cost path to each node it visits. It stops, when the highest constraint is exceeded, or there is no more possibility of improving the paths. Since this extension uses delay instead of hop-count, which is a continuous metric, the routing table which contains entries for all possible delays can be very large, even of unrealizable size. As we mentioned the CBF algorithm has an exponential running time. Widyono have not analyzed it conclusively, but stated that despite of its exponential nature, the performance of the algorithm was reasonable in practice.

As a further development of the CBF algorithm, using a kind of scaling technique, the authors of [Has92], [LO98], [RS00] and [LORS00] gave *fully polynomial time approximation schemes* for the DCLC problem, namely they proved that for any $\varepsilon > 0$ there exists a polynomial time algorithm that is able to find a path satisfying the delay constraint with cost no greater than a factor of $1 + \varepsilon$ from the optimum. The running time of the best known approximation scheme is $O(nm \log n \log\log n + \frac{nm}{\varepsilon})$ [LORS00] . Unfortunately, *in practical cases* the running time of these methods for sufficiently small ε will be worse than even of the CBF algorithm, which makes these results rather theoretical.

All other algorithms try to use some kind of *heuristics*. A very simple method proposed by W. C. Lee [L$^+$94] does not give optimal paths, but provides a simple heuristic solution to the DCLC problem. The *Fallback* routing algorithm assumes that there are ranked metrics in the network. First, the routing algorithm calculates the shortest path for the first metric (i.e. cost), and then checks whether it can guarantee all the other QoS requirements. If the path failed, the algorithm tries to find another one for the next metric, until an appropriate path is found, or the routing fails for all the metrics. This algorithm is very simple, fast and always gives an appropriate solution if it exists, but there is no guarantee of finding the optimal route and we do not know anything about the quality of the found path.

The algorithm proposed by Pornavalai [PCS98] improves the previous idea by *combining paths* calculated on the different metrics, by first calculating the paths based on the metrics from the source node to the others and from all the nodes to the destination, and then trying all the possible combinations of them. There are two more algorithms improving the idea of Fallback. Ishida et al. [IAK98] proposed a distributed algorithm in which the nodes always choose the least cost path until it fulfills the delay requirements and from that point they choose the path with the least delay. The *Delay Constrained Unicast Routing* (DCUR) algorithm proposed by Salama [SRV97] is similar to it, but it can choose between the least cost and least delay paths independently from the choice of the previous node. These three algorithms have higher but still reasonable running time, and they are more likely to find a solution closer to the optimum than it was in case of the Fallback algorithm, but neither these algorithms have guarantees for the optimality of the path.

Cheng and Nahrstedt [CN98] give an algorithm to find a path that meets two requirements in polynomial time. The algorithm *reduces* the original problem to a simpler one by modifying the cost function, based on which the problem can be solved using an *extended* shortest path algorithm. The shortcoming of this method is that the algorithm has to use high granularity in approximating the metrics, so it can be very costly in both time and space, and it can not guarantee that the simpler problem has a solution if the original problem has.

The last group of algorithms [Jaf84, BG96, dNvM98, GM03] is based on calculating a *simple metric* from the multiple requirements. To do so, one can use a simple shortest-path algorithm based on a single cost aggregated as a combination of weighted QoS parameters. The main drawback of this solution is that the result is quite sensitive to the selected aggregating weights, and there is *no clear guideline on how the weights should be chosen.* To handle the problem of determining the weights for the aggregated cost solution, several routing algorithms has been proposed. For example the algorithm proposed in [I$^+$96] tries to find an appropriate path based on a single cost of weighted link metrics. If it fails, it tries to find another path by iterative changing the values of weights so as to get more possible paths, until an appropriate path that guarantees QoSs can be found. The problem with this solution is that if the path is correct for the aggregated cost, it does not mean that it is correct for all of the user's QoS requirements. In other words, *information is lost in the aggregation process.*

Wang and Crowcroft [WC95] say that single (mixed) metric approach can not be suffi-

cient for QoS routing. It can at best be an indicator in path selection but does not provide enough information for making reliable quantitative estimation of whether the path meets the requirements or not. For example, he proposes a mixed metric from delay, bandwidth and loss probability. A path with a large value of $\frac{B(p)}{D(p)\cdot L(p)}$ is likely to be a better choice in terms of bandwidth, delay and loss probability. However, it is not sure that a path, which minimizes the above expression, is suitable for any of the constraints (bandwidth, delay or loss).

On the contrary we state that this problem can be solved, and there are algorithms based on aggregated costs, which can find paths that meet the QoS constraints. Instead of a static combination of cost and delay the presented Handler-Zang method is able to find an aggregated cost adaptively for each route calculation.

3.4.2 Practical Improvements to LARAC algorithm

In this section we mention two important possibilities of improving LARAC.

Storing Former Results

In reality a router computes paths for a lot of destinations, instead of doing it for a single one. The original LARAC algorithm can be improved by utilizing this fact, by reusing results of previous calculations.

So, whenever we call Dijkstra algorithm with a certain λ we store its whole result, that is, we store the c_λ-minimal paths to all destinations. These paths form a tree, so we need to store only m edges. Moreover we store the c-cost and the d-cost of these path.

The new method is very simple. The only difference is that when we are looking for a path to a new destination, we are looking for the largest *previously calculated* λ_1 for which $d(p_{\lambda_1}) \geq \Delta$ and the smallest λ_2 for which $d(p_{\lambda_2}) \leq \Delta$, and we initialize p_c and p_d with these paths instead of c- and d-minimal ones.

Improved Stop Condition

We optimize the algorithm to gain faster running time, by giving up the requirements on the minimality of the found path's cost. When we compute a path for a given source-destination pair, we calculate with different λ values. As the iteration number increases,

so decreases the difference between the subsequent λ values, and the difference between the cost of the paths also decreases.

So, let $p_c^0, p_c^1, p_c^2, \cdots, p_c^j$ be the series of paths (say from source s to destination t) that do not satisfy the delay constraint, and let be $p_d^0, p_d^1, p_d^2, \cdots, p_d^k$ the series of paths that satisfy the delay constraint.

Our aim is to stop the algorithm before it finds the optimal solution, but only when can assure that the result is not far from the optimum.

The algorithm gives p_d^k as the final solution, so we are interested in the difference between cost of the tags of the p_d-series and the last tag p_d^j. Let us examine the running algorithm in a certain intermediate iteration and let suppose that the current paths at the end of this iteration are p_c^a and p_d^b. Because $c(p_c^a) < c(p_d^k)$, we get that

$$c(p_d^b) - c(p_d^k) < c(p_d^b) - c(p_c^a). \tag{3.26}$$

So, if the difference between $c(p_d^b)$ and $c(p_c^a)$ is small enough, then the difference between $c(p_d^b)$ and $c(p_d^k)$ will be possibly negligible, in which case there is no need to continue the running of the iteration.

We define an upper bound named *Maximal Difference* (MD), which shows in percentage that how much difference is tolerated between the cost of the original solution (that would be given by the original algorithm) and the cost of the new solution's path (that is given by the improved algorithm). Thus, if

$$c(p_d^b) \leq (1 + MD) * c(p_c^a) \tag{3.27}$$

stands at the end of an iteration, then the algorithm can stop, since no further significant improvement on the resulting path's cost can be earned. Our tolerance toward the minimality of cost is represented in the value of MD.

3.4.3 Practical Evaluations

We used simulations to evaluate the performance of the LARAC algorithm and other QoS routing algorithms proposed in the literature. This section is structured as follows: we survey the implemented algorithms and measurements in Section 3.4.3, then in Section 3.4.3 we give a description of the simulation environment. Finally in Section 3.4.3 we introduce our simulation experiments.

Implemented Algorithms and Measurements

We investigated the performance of the *LARAC* algorithm and compared it to the most promising three algorithms selected from the ones surveyed in Section 3.4.1. The *CBF* algorithm has been selected, because it gives the optimal solution, which can be a useful measure for the other algorithms. The *Fallback* algorithm has been selected for its simplicity; *DCUR*, because it was the most promising among the algorithms that compose the constrained path from other paths obtained by simple shortest path algorithms. Since LARAC can give the best solution that can be obtained with aggregated costs, we did not implement any other algorithms of this category.

We compared the behavior of the implemented algorithms with the help of the following measures:

- *Average Number of Unreachable Nodes*: It can happen that with a given delay constraint Δ there is no path in the network to a certain destination. We need this measure to know whether all the algorithms can find paths to the same number of destination nodes.

- *Average Cost*: this attribute measures the average cost of the paths from a source node to all the reachable destination nodes.

- *Average Delay*: It is the average delay of the paths from a source node to all the reachable destination nodes.

- *Average Number of Steps*: This attribute measures the running time of the algorithms. Since both Dijkstra and CBF works with a heap, the number of step represents the events when the algorithm changes the contents of the heap. It will not give exact results, since the calculation of the number of steps is different for CBF and the other algorithms that use Dijkstra, however complexity measurements based on these calculations still provide us valuable feedback. For those algorithms that execute Dijkstra several times during its calculations, the complexity measure gives comparable results.

- *Cost Inefficiency*: this measure can be calculated from the optimal cost measure of the CBF algorithm [GM03, SRV97]. It can be defined as:

$$\delta_A := \frac{C(p_A) - C(p_{CBF})}{C(p_{CBF})} \tag{3.28}$$

where A stands for the algorithm for which the cost inefficiency is calculated. We will use $\overline{\delta}_A$ and $max\delta_A$ to characterize comprehensively the results of an algorithms provided for different delay constraints.

Simulation Environment

For the simulation we built 100 random networks with 40 nodes and an average node degree of 4 (reflecting real network values [SRV97]). The results of the measurements have been averaged. The topology of the networks was generated by a random graph generator program [JO99]. The cost value on links varies from 1 to 15 based on uniform distribution. The propagation delay on links is selected from three ranges to resemble delay characteristics of a nationwide network e.g. in the US. The first range (1-5 ms) represents short local links, the second range (5-8) represents longer local links, while the third range (20-30 ms) represents continental links. The ratio of long local links was configured to be 20 percent, while the ratio of continental connections in networks was configured to be 5 percent.

In the simulations we changed the delay constraint Δ and investigated its effect on the found paths of different algorithms. Since the average number of unreachable nodes is the same for all algorithms, the other measures of the routing algorithms are comparable.

Experiments

In Section 3.4.3 we first investigate the basic LARAC algorithms without the proposed improvements described in details in Section 3.4.2, then Experiments of the improved LARAC algorithm are presented in Section 3.4.3.

Comparison of the four algorithms As it was mentioned before an important result of the simulation is that the average number of unreachable nodes is the same for all algorithms. This means that all algorithms managed to find a path that satisfies the delay constraint. However, in minimizing the other additive metric (i.e., cost) the algorithms provide different results. In Figure 3.2 we can see the average cost of the paths found by the algorithms. The LARAC algorithm found almost the same paths as CBF, while DCUR found paths with higher cost, and the worst cost was achieved by Fallback. The result of Fallback meets our expectations, however we could not tell before that DCUR is worse

than LARAC, although it could be seen that DCUR can easily find paths with higher cost than the optimal. Finally, LARAC's lower bound for the optimal path cost is very close to the results of CBF.

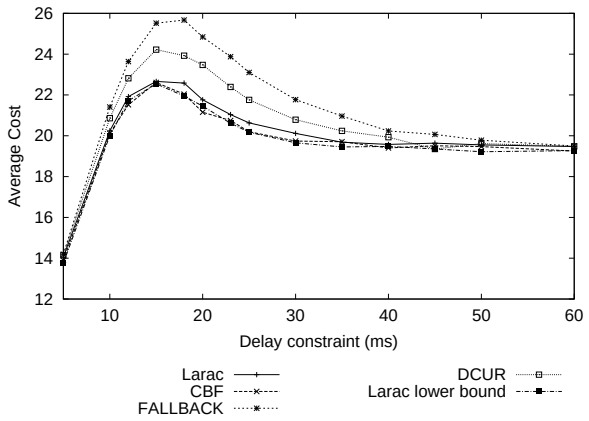


Figure 3.2: Average cost

The maximum worst case cost $\max \delta_A$, and the average worst case cost $\overline{\delta}_A$ of the algorithms compared to the cost of the optimal CBF are the following (Table I):

Table 3.1: Maximum and Average cost inefficiency

Algorithm	$\max \delta_A$	$\overline{\delta}_A$
LARAC	102.9%	101.3%
DCUR	110.9%	104.7%
Fallback	117.4%	108.7%
LARAC lower bound	99.27%	99.7%

It can be seen in Figure 3.2 that at the first section of the curves ($\Delta < 15$) the found paths have very low cost, and after the cost of the path increases with the delay bound. The explanation of this phenomenon is that it is impossible to find path for all source destination pairs with a small Δ, and the found paths are short, thus the costs of them are also small. As the delay bound increases the algorithms find more and more (longer) paths, therefore the average cost of the paths increases. After the point where a path can be found for all source destination pairs, the average cost of the paths will decrease with

the delay bound, because the algorithms are able to find the paths with lower costs and higher delay bounds.

This effect can be noticed on the delay curves (Figure 3.3) as well. We expected CBF to have the highest delay, then LARAC, DCUR and Fallback. The result we got is almost meets our expectations, but for surprise DCUR has the highest delay.

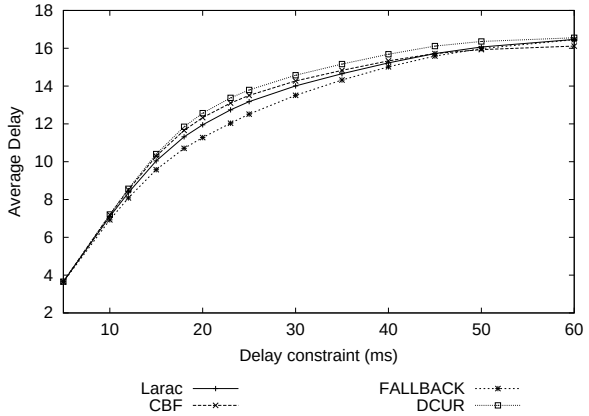


Figure 3.3: Average delay

In Figure 3.4 the average number of steps of the algorithms are compared. As we can see all algorithms have a very characteristic curve for the number of steps.

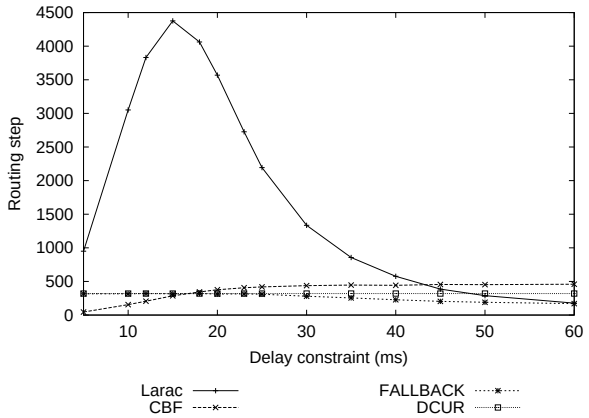


Figure 3.4: Average number of step

For the small delay constraints LARAC has an increasing running time, since as it can find paths to more and more destinations, the number of Dijkstra executions increases. As

it reaches the delay constraint that allows paths to all the nodes in the network, the number of steps starts to decrease since minimum cost paths computed first can more likely satisfy the looser delay bounds, thus eliminating the execution of the second Dijkstra algorithm to get the minimum delay path. This is the reason why the average number of steps for the LARAC algorithm approximates the number of step of a single Dijkstra algorithm for high delay values.

Fallback has a decreasing characteristic as well, the difference is that it is linear in the first section, and equals to the running time of 2 Dijkstra algorithms. After this first section, the number of steps decreases to the number of steps of 1 Dijkstra, for similar reasons as it was explained for LARAC.

We implemented the DCUR algorithm in a centralized manner for the simulations. For this reason the running time of it would be huge ($2n$ times the running time of a Dijkstra algorithm), so in the figure an estimation is plotted for a distributed implementation. We used the execution time of 2 Dijkstra algorithms for the running time estimation of the distributed DCUR.

CBF has a totally different characteristic. Since the algorithm exit condition depends only on the delay constraint, it increases with Δ. The last horizontal section shows, that the algorithm stopped because all the paths from the heap has been already used.

Improvements to LARAC algorithm We investigated both optimization proposal's effect on the performance of the LARAC algorithm. The first optimization (storing former results) aims at improving the running time of the basic LARAC algorithm. Based on our simulations we can say that the gain on the routing step is quite significant. Under the critical value of delay bound (which is 15 ms) up to 39% gain is earned.

The second optimization (improved stop condition) affects not only the running time but also the cost of the found paths, since it modifies the exit condition of the algorithm.

Figure 3.5 and 3.6 show the average routing step of the algorithms and average cost of the computed paths. As we can see on figure 3.5, the running time greatly decreases as we increase the value of Maximal Difference (or shortly MD). Four different results are presented, one for each of the different MD values. Table II summarizes the improvement of the algorithms' running time with the help of *average iteration number*. This provides us the average number of times the Dijkstra algorithm was executed with a given algorithm. As we can notice, the original LARAC algorithm had an average iteration number of 7.94,

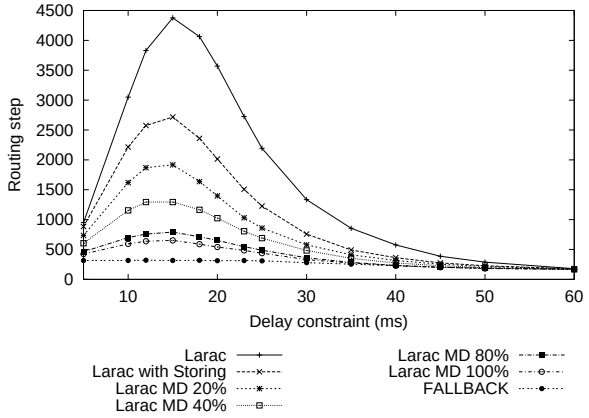


Figure 3.5: Average number of step

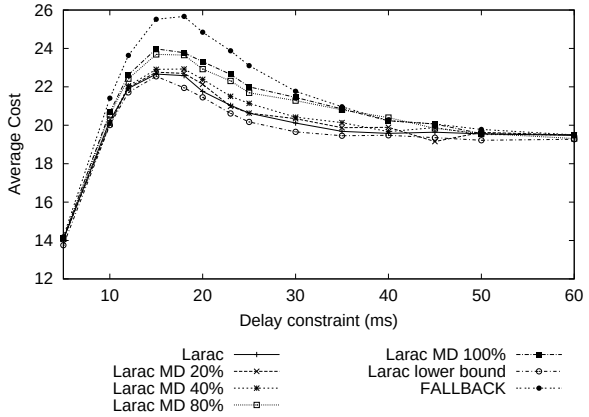


Figure 3.6: Average cost

while the improved one with e.g., MD 40% this value decreased to 4.34.

Table 3.2: Summary of average iteration number and optimality of costs

Algorithm	Average Iteration Number	$\max \delta_A$	$\overline{\delta}_A$
LARAC	7.94	102.9%	101.3%
MD 20%	5.80	104.7%	101.5%
MD 40%	4.34	105.8%	102.4%
MD 80%	2.92	108.4%	104.6%
MD 100%	2.57	110.2%	105.4%

These values are very impressive, but the gain on routing steps is only one side of the coin. We have to examine the average cost of the found paths as well. In Figure 3.7 and in Table 3.4.3 the increase in cost is presented. When the MD value 40% was used, the average cost inefficiency $\overline{\delta}_A$ increased to 102.4%. Similarly at the MD value 80%, $\overline{\delta}_A$ equals to 104.6%. The gain on running time compared to the loss on the optimality of path costs seems to be worthy. Moreover, our algorithm provides a good way of controlling the trade-off between optimality of the path, and running time of the algorithm.

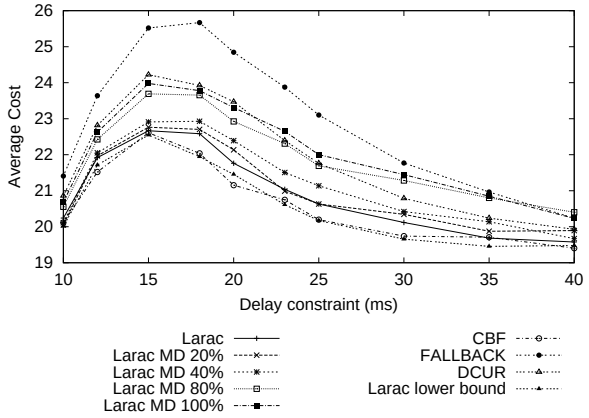


Figure 3.7: Average cost

Chapter 4

Fractional Optimization Problems

This chapter shows that some ideas in Chapter 3 can also be used to improve on the running time estimation of the Newton-Dinkelbach method for the Fractional Optimization Problems. The results presented here are joint work with Tomasz Radzik.

To define *Fractional Optimization Problem*, let X be a (finite) set of feasible solutions, and $c, w : X \longrightarrow \mathbb{R}$ given functions. We assume that $w(x) > 0$ for all x and there exists x for which $c(x) > 0$. With these notation, the problem is the following.

Problem 4.1 ($\mathcal{F}$) *Compute the value*

$$\alpha := \max\left\{\frac{c(x)}{w(x)} : x \in X\right\} \tag{4.1}$$

and a maximizing solution $x^* \in X$.

It is well know that the problem above is equivalent with solving Problem *(P)* below.

Problem 4.2 ($\mathcal{P}$) *Compute the value*

$$\alpha = \min\left\{\lambda : c(x) - \lambda w(x) \leq 0 \quad \forall x \in X\right\}. \tag{4.2}$$

The parametric version of this problem is

Problem 4.3 ($\mathcal{P}(\lambda)$) *Compute the value*

$$h(\lambda) := \max\left\{c(x) - \lambda w(x) : x \in X\right\} \tag{4.3}$$

and find a maximizing $x_\lambda \in X$.

If is easy to see that function $h : \mathbb{R} \longrightarrow \mathbb{R}$ is continuous (piecewise linear), monotonically decreasing, convex, $h(0) > 0$ and $\alpha = \min\{\lambda : h(\lambda) = 0\}$.

There are two common ways to find this value in strongly polynomial time. First, Megiddo's parametric search method described in Section 1.1.2 can be applied (see [Meg79]). In general, this provides an $O(T^2)$ algorithm, where T is the running time required to compute $h(\lambda)$. In running time can often be significantly improved in special cases using the technique presented in [Meg83, Rad98]. In fact, this technique provides best worst case running time algorithm for most of the common fractional optimization problems. unfortunately, the practical performance of this approach is less attractive.

Is is easy to see that $w(x_\lambda)$ is a subgradient of function h in the point λ. Therefore, as an alternative approach, one can apply the Newton method to function h to solve fractional optimization problems if there exists an efficient way to compute the value $h(\lambda)$ and the solution x_λ for any given $\lambda \geq 0$. The obtained algorithm called *Newton-Dinkelbach method* [Rad98] is shown in Figure 4.1.

This algorithm is shown to be very efficient both theoretically and in practice. It is shown in [Rad98] that if X consists of subsets of an underlying set E of size n and both c and w are a linear function, then the Newton-Dinkelbach method terminates in $O(n^2 \log n^2)$ iterations. In the special case when X is the set of s—t paths in a directed acyclic graph, the number of iteration is $O(n \log n)$.

This chapter improves the running estimation to $O(n^2 \log n)$ in case of linear objective functions and extends the latter result to the case when X is given as the 0-1 vertices of the integer polyhedron $\mathcal{Q} := \{x \in \mathbb{R}^n : Ax = b, x \geq 0\}$.

4.1 General properties of Newton-Dinkelbach Method

This section lists some basic properties of Newton-Dinkelbach Method. The first three lemmas directly follows the convexity of the function $h(\lambda)$.

Lemma 4.4 ([Rad98]) $h_1 > h_2 > \cdots > h_t = 0$. □

Lemma 4.5 ([Rad98]) $0 = \lambda_1 < \lambda_2 < \cdots < \lambda_t = \lambda^*$. □

Lemma 4.6 ([Rad98]) $w_1 > w_2 > \cdots > w_t = 0$. □

procedure NewtonDinkelbach$(c, w, \mathcal{P}(\lambda))$
  $\lambda_1 := 0$
  $i := 0$
  do
    $i := i + 1$
    $x_i := x_{\lambda_i}$
    $c_i := c(x_i)$
    $w_i := w(x_i)$
    $h_i := h(\lambda_i) = c_i - \lambda_i w_i$
    $\lambda_{i+1} := \dfrac{c_i}{w_i}$
  while $h_i > 0$
end procedure.

Figure 4.1: The Newton-Dinkelbach algorithm

Lemma 4.7 ([Rad98]) $\dfrac{h_{i+1}}{h_i} + \dfrac{w_{i+1}}{w_i} \leq 1.$

Proof. The following calculation shows the inequality.

$$\begin{aligned} h_i &= c_i - \lambda_i w_i = c(x_i) - \lambda_i w(x_i) \geq c(x_{i+1}) - \lambda_i w(x_{i+1}) = \\ &= c_{i+1} - \lambda_i w_{i+1} = h_{i+1} + \lambda_{i+1} w_{i+1} - \lambda_i w_{i+1} = \\ &= h_{i+1} + \frac{c_i}{w_i} w_{i+1} - \frac{h_i - c_i}{w_i} w_{i+1} = h_{i+1} + \frac{h_i w_{i+1}}{w_i} \end{aligned}$$

□

A trivial consequence of the Lemma above is the following.

Corollary 4.8 ([Rad98]) $\dfrac{h_{i+1} w_{i+1}}{h_i w_i} \leq \dfrac{1}{4}.$

Lemma 4.9 ([Rad98]) *$c_i - \lambda w_i \leq -h_{i+1}$ holds for all $i = 1, 2, \cdots, t-2$ and $\lambda \geq \lambda_{i+2}$.*

Proof. Because $w_i > 0$, it is enough to proof the claim for $\lambda = \lambda_{i+2}$. So,

$$h_{i+1} = c_{i+1} - \lambda_{i+1} w_{i+1} = (\lambda_{i+2} - \lambda_{i+1}) w_{i+1} \geq (\lambda_{i+2} - \lambda_{i+1}) w_i = \lambda_{i+2} w_i - c_i$$

finishes the proof. □

4.2 The Case of Linear Objective Functions

Let E be a finite underlying set, $n := |E|$ and assume that $X \subseteq \{0,1\}^E$ and c and w are linear functions, i.e.

$$c(x) := \sum_{e \in E} c_e, x_e \tag{4.4}$$

and

$$w(x) := \sum_{e \in E} w_e, x_e \tag{4.5}$$

for some given vectors $c, w \in \mathbb{R}^E$. In this section we prove that the Newton-Dinkelbach method terminates in $O(n^2 \log n)$ iterations in this case.

Theorem 4.10 *In the linear case, the Newton-Dinkelbach method terminates in* $O(|E|^2 \log |E|)$ *iterations.*

Proof. As in Section 3.3, the bound on the running time is based on Theorem 3.8. Consider the sequence

$$s_i := h_{2i} w_{2i-1}. \tag{4.6}$$

Because of the monotonicity of h_i and w_i and Corollary 4.8,

$$s_{i+1} = h_{2i+2} w_{2i+1} \leq h_{2i+1} w_{2i+1} \leq \frac{1}{4} h_{2i} w_{2i} \leq \frac{1}{4} h_{2i} w_{2i-1} \leq \frac{1}{4} s_i. \tag{4.7}$$

On the other hand

$$s_i = h_{2i} w_{2i-1} = (c_{2i} - \lambda_{2i} w_{2i}) w_{2i-1} = (c_{2i} - \frac{c_{2i-1}}{w_{2i-1}} w_{2i}) w_{2i-1} = c_{2i} w_{2i-1} - c_{2i-1} w_{2i}, \tag{4.8}$$

showing that there exist $\mathbf{c} \in \mathbb{R}^{|E|^2}$ and $\mathbf{y_i} \in \{-1,0,1\}^{|E|^2}$ such that $s^i = \mathbf{y_i c}$ for all i. Indeed, $\mathbf{c}$ is an appropriate choice if it consists of the following constants.

$$\Big\{\alpha \cdot \beta \colon \quad \alpha \in \{c(e) : e \in E\}, \quad \beta \in \{w(e) : e \in E\}\Big\}.$$

Finally, applying Theorem 3.8 to the vector $\mathbf{c}$, we get that the length of the sequence s_i is $O(|E|^2 \log |E|)$, resulting in the same bound on the number of iterations in Newton Dinkelbach method. □

4.3 The Case of LP Defined Basic Problems

This sections shows that if $X \subseteq \{0,1\}^E$ is defined as a set of the basic solutions of a linear program, then Theorem 4.10 can be further improved.

Theorem 4.11 *Let E be a finite underlying set, $n := |E|$ and assume that $X \subseteq \{0,1\}^E$ is the set of vertices of the polyhedron $\mathcal{Q} := \{x \in \mathbb{R}^E : Ax = b, \quad x \geq 0\}$. Then the Newton-Dinkelbach method terminates in $O(n \log n)$ iterations.*

Proof. By definition, x_λ is an optimal solution of the problem

$$h(\lambda) = \max\{c_\lambda x : x \in \mathbb{R}^E, Ax = b, \quad x \geq 0\}, \tag{4.9}$$

where $c_\lambda := c - \lambda w$. Let y_λ be an optimal solution of the corresponding dual linear program

$$\min\{yb : y \in \mathbb{R}^q, \quad yA \geq c_\lambda\}, \tag{4.10}$$

Let $x_1, x_2, \ldots, x_t$ be the solutions found by the algorithm. An coordinated $e \in E$ is called *essential in the iteration i* if there exists $j \geq i$ such that $x_j(e) = 1$. Theorem 4.11 follows from the following.

Claim 4.12 *After making $k := \lceil \log n \rceil + 1$, the number of the essential coordinates strictly decreases.*

Proof. From Corollary 4.8 we get that

$$h_{i+k} w_{i+k} \leq \frac{1}{n^2} h_{i+1} w_{i+1} \tag{4.11}$$

It is enough to prove that there exist $e \in E$ such that $x_i(e) = 1$ but $x_j(e) = 0$ for all $j > i + k$. There are two cases.

$\boxed{w_{i+k} \leq \frac{1}{n} w_i}$ In this case there exists $e \in E$, $x_i(e) = 1$ such that $w(e) \geq \frac{1}{n} w x_i$. From this we get that

$$w(e) \geq \frac{1}{n} w_i \geq w_{i+k} > w_j = w x_j, \tag{4.12}$$

from which $x_j(e) = 0$ follows.

$\boxed{w_{i+k} > \frac{1}{n} w_i}$ From this we get that $w_{i+k} > \frac{1}{n} w_{i+1}$, therefore $h_{i+k} < \frac{1}{n} h_{i+1}$. Let $\tilde{c}_\lambda := c_\lambda - y_\lambda A$. The linear complementary condition shows that $\tilde{c} \leq 0$.

$$c_{\lambda_{i+k}} x_i = c x_i - \lambda_{i+k} w x_i = c_i - \lambda_{i+k} w_i \leq -h_{i+1} < -n h_{i+k}, \tag{4.13}$$

thus

$$\tilde{c}_{\lambda_{i+k}} x_i = c_{\lambda_{i+k}} x_i - y_{\lambda_{i+k}} A x_i < -n h_{i+k} - y_{\lambda_{i+k}} b = -(n+1) h_{i+k}. \tag{4.14}$$

Then there exists $e \in E$, $x_i(e) = 1$ such that $\tilde{c}_{\lambda_{i+k}}(e) < -h_{i+k}$. Assume that $x_j(e) = 1$. Then

$$0 < h_j = c_{\lambda_j} x_j \leq c_{\lambda_{i+k}} x_j = \tilde{c}_{\lambda_{i+k}} x_j + y_{\lambda_{i+k}} A x_j \leq c_{\lambda_{i+k}} + h_{i+k} < 0, \tag{4.15}$$

therefore we get that $x_j(e) = 0$, which finishes the proof.

□

□

Chapter 5

UMTS Access Network Topology Design

The present chapter introduces two network design algorithms for planning UMTS (Universal Mobile Telecommunication System) access networks. The task here is to determine the cost-optimal number and location of the *Radio Network Controller* nodes and their connections to the *Radio Base Stations* (RBS) in a tree topology according to a number of planning constraints. First, a global algorithm to this general problem is proposed, which combines a metaheuristic technique with the solution of a specific b-matching problem. It is shown how a relatively complex algorithm can be performed within each step of a metaheuristic method still in reasonable time. Then, another method is introduced that is able to plan single RBS-trees. It can also be applied to make improvements on each tree created by the first algorithm. This approach applies iterative local improvements using branch-and-bound with Lagrangian lower bound. Section 5.6 demonstrates through a number of test cases that these algorithms are able to reduce the total cost of UMTS access networks, also compared to previous results. Finally — for the sake of completeness — Section 5.7 gives a short proof of the $\mathcal{NP}$-hardness of the problem in the special case when only a single tree with a given RNC node is planned.

5.1 UMTS Architecture and Cost Model

UMTS [UMT98], [OP98] stands for Universal Mobile Telecommunication System; it is a member of the ITU's[1] IMT-2000[2] global family of *third-generation* (3G) mobile communication systems. UMTS will play a key role in creating the future mass market for high quality wireless multimedia communications, serving expectedly 2 billion users worldwide by the year 2010.

One part of the UMTS network will be built upon the transport network of today's significant 2G mobile systems [Kal98], but to satisfy the needs of the future Information Society new network architectures are also required. Since UMTS aims to provide high bandwidth data, video or voice transfer, the radio stations communicating with the mobile devices of the end-users should be placed more densely to each other, corresponding to the higher frequency needed for the communication. Using the traditional star topology of the radio stations, this would increase the number of expensive central stations as well. To solve this problem, UMTS allows a constrained tree topology of the base stations, permitting to connect them to each other and not only to the central station. This new architecture requires new planning methods as well.

As shown in Figure 5.1 the UMTS topology can be decomposed into two main parts: the core network and the access network. The high-speed core network based on e. g. ATM or IP technology connects the central stations of the access network. These connections are generally realized with high-capacity optical cables. The focus is on planning the access network, the design of the core network is beyond the scope. The reader is referred to e. g. [Mar97] on this topic.

The access network consists of a set of *Radio Base Station*s (RBS nodes) and some of them are provided with *Radio Network Controller*s (RNC nodes). The RBSs communicate directly with the mobile devices of the users, like mobile phones, mobile notebooks, etc., collect the traffic of a small region and forward it to the RNC they belong to using the so-called *access links*, which are typically microwave radio links with limited length (longer length connections can be realized using repeaters, but at higher cost) and capacity. In

[1]ITU is the abbreviation of *International Telecommunication Union*, an international organization within the United Nations System where governments and the private sector coordinate global telecom networks and services.

[2]IMT-2000 stands for *International Mobile Telecommunications-2000* and it is the global standard for third generation (3G) wireless communications, defined by a set of interdependent ITU Recommendations.

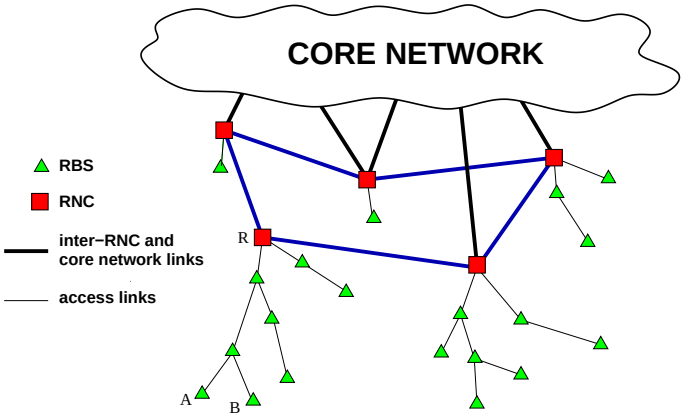


Figure 5.1: UMTS topology

traditional GSM configuration the RBSs are connected directly to an RNC station limiting the maximum number of RBSs belonging to a certain RNC. To overcome this limitation the UMTS technology makes it possible for an RBS to be connected to another RBS instead of its RNC. However RBSs have no routing capability, they simply forward all the received data toward their corresponding RNC station, therefore all traffic from an RBS goes first to the RNC controlling it. For example if a mobile phone in region A on Figure 5.1 wants to communicate with a device in region B, their traffic will be sent trough the RNC R. Only the RNC station is responsible for determining the correct route for that amount of traffic. It follows that the RBSs should be connected to the RNC in a tree topology. (There are research initiatives to provide some low-level routing capability for the RBSs and to allow additional links between RBSs, which may increase the reliability of the network. These developments are quite in early stage, hence we only deal with tree topology.) As a consequence, the access network can be subdivided into independent trees of RBSs, each rooted at an RNC. These trees are called *Radio Network Subsystem*s (RNS). Further advantage of the tree topology compared to the star topology is that the links can become shorter, which on one hand reduces the cost of the links, on the other hand it may require less repeaters in the network, thus increase its reliability.

Moreover, there are some additional links connecting the RNCs to each other (*inter-RNC link*s) and to the core network. The planning of these links are beyond the scope of this thesis, though a rough estimation of its cost can be built into the objective function using the RNC cost.

For technical reasons the following strict topology constraints have to be taken into account:

- The limited resources of the RBS stations and the relatively low bandwidth of the access links cause considerable amount of delay in the communication. In order to reduce this kind of delay, the maximum number of the access links on a routing path is kept under a certain amount by limiting the depth of a tree to a small predefined value. This limit is denoted by l_{tree} in our model. The usual value of l_{tree} is 3.

- The degree of an RBS is also constrained. One simple reason is that the commercial devices have only limited number of ports. Another reason is that too many close RBSs can cause interference problems in their radio interface if their connections are established through microwave links. Moreover the capacity of an RBS device also limits the number of connectible RBSs. Therefore in our model there can be at most d_{RBS} RBSs connected directly to another RBSs on a lower level. It is typically a low value, $d_{RBS} = 2$ in the devices existing at the time this research was made.

- The degree of RNCs (the number of RBSs connected directly to a certain RNCs) is also limited for similar reasons, it is at most d_{RNC}, but d_{RNC} is generally much larger than d_{RBS}.

The planning task investigated in present chapter is to plan *cost-optimal access network*, that is to determine the optimal *number* and *location* of the RNCs and to find the *connections of minimal cost* between RNCs and RBSs satisfying all the topological restrictions.

The cost of a UMTS access network is composed of two variable factors: the cost of the RNC stations and the cost of the links. For the exact definitions see Section 5.2.

As one may expect, this planning problem is $\mathcal{NP}$-hard. The problem of finding a minimal weight two-depth rooted tree in a weighted graph can be reduced to this problem. Moreover, the problem remains $\mathcal{NP}$-hard even in the special case of planning a single tree with a fixed RNC node. See Section 5.7 for the proof of this claim.

A general UMTS network may contain even about 1000 RBSs, a powerful RNC device controls approximately at most 200-300 RBSs. This large number of network nodes indicates that the planning algorithms have to be quite fast in order to get acceptable results.

Solving methods. One possible approach to the problem described above would be to divide the set of input nodes into clusters and then create a tree with one RNC in each cluster. However, this "two-stage" solution has some significant drawbacks: first, it is very hard to give a good approximation to the cost of a cluster without knowing the exact connections. Second, it would be also a strong restriction to the algorithm searching for connections to work with static clusters of nodes created in the beginning.

For these reasons a "one-stage" approach is introduced: an algorithm which creates a number of independent trees with connections simultaneously. The proposed method called *Global* is based on a widely used metaheuristic technique called Simulated Annealing. However, it is not straightforward to apply Simulated Annealing to this problem, since the state space is very large and a reasonable neighborhood-relationship is hard to find. To overcome these problems a combination of the Simulated Annealing and a specific b-matching algorithm is proposed. To make this method efficient it will be shown how the relatively complex and time-consuming b-matching method can be performed within each step of a metaheuristic process in still acceptable time.

Then, a Lagrangian relaxation based lower bound computation method is presented for the problem of designing a single tree with predefined RNC node. Using this lower bound a branch-and-bound method is proposed to compute the theoretical optimal solution to this problem for smaller but still considerable large number of nodes.

For bigger single-tree design tasks, a second heuristic method called *Local* is proposed based on an effective combination of a local search technique and the branch-and-bound procedure above. This *Local* algorithm can be used effectively either in circumstances when only a single tree should be designed or to improve each trees provided by the *Global* algorithm.

5.2 Problem Definition and Notation

This section gives a formal description of the problem investigated.

The access network is modeled as a directed graph $G = (N, E)$, where N is the set of the RBSs. Each edge in E corresponds to a link between its ends and directed toward the corresponding RNC. On the other hand, this set E of directed edges determines the set of RNCs as well: a node is RNC if and only if it has no outgoing edge.

In order to illustrate the logical structure of the network the notion of the *level of nodes*

is introduced (Figure 5.2). Let all RNC nodes be on level 0, and let the level of an RBS station be the number of edges of the path that connects it to its controlling RNC. The level of a link is defined as the level of its end on the greater level-number. Other important notation used in this chapter is shown in Table 5.1.

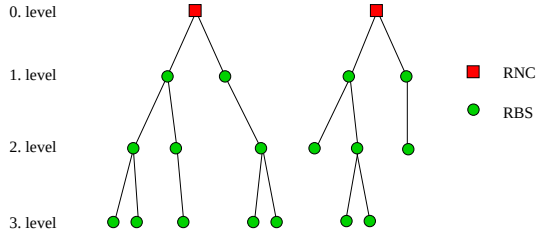


Figure 5.2: Logical structure of the network

N	the set of RBSs
E	the set of links
n	the number of network nodes
l_{tree}	the maximal depth of the trees
L_i^E, L_i	the set of nodes on the i-th level of the graph
E_i	the set of links on the i-th level of the graph
$l^E(v), l(v)$	the level of the node v
$l^E(e), l(e)$	the level of the edge e
d_{RBS}	the maximal degree of an RBS
d_{RNC}	the maximal degree of an RNC
$cost_{RNC}$	the cost of an RNC

Table 5.1: Some important notation

The input of the examined planning problem consists of the set N of RBSs, the cost function c_{link} of the links (described later), the installation cost $cost_{RNC}$ of the RNCs and the constraints d_{RNC}, d_{RBS} and l_{tree}. Moreover we are given the set R_R of places of required RNCs and the set R_P of places of possible RNCs. (It is useful since in many practical cases already existing networks should be extended.)

The set E of links is a *feasible solution* to this input if

- E forms a set of disjoint rooted trees, which cover the whole set N,
- the depth of each tree is at most l_{tree},
- the degree of the root of each tree is at most d_{RNC},
- the in-degree of each other node is at most d_{RBS},
- $R_R \subseteq L_0^E \subseteq R_P$. (By definition, L_0^E is the set of the RNC nodes.)

The *total cost* of the network is composed of the following factors.

- The cost of the RNC stations. The cost of one RNC, $cost_{RNC}$, means the installation cost of that particular RNC. This constant may contain other factors as well, e. g. the cost of the links between the RNCs can be included, assuming that it has a nearly constant additional cost for every RNC.
- The cost of the access links. In this model the cost $c_{link}(i, j, l)$ of a link depends on its endpoints i and j and its level l. A possible further simplification is to assume that $c_{link}(i, j, l) = f_l \cdot c_{link}(i, j)$, where f_l is a constant factor representing the weight of level l.

  This kind of link-cost function has two applications. First, since access links closer to the RNC aggregate more traffic, this is an elementary way to model the capacity dependent costs by giving higher cost on the lower levels. Second, it can be used to prohibit the usage of some links on some levels by setting their cost to an extremely large value. Moreover, it makes it possible to force a node to be on a predefined level.

So, the task is to find a feasible connection E minimizing the total cost

$$|L_0^E| \cdot cost_{RNC} + \sum_{(ij)\in E} c_{link}\Big(i, j, l^E\big((ij)\big)\Big) \tag{5.1}$$

5.3 Related work

Similar problems have already been examined by several authors, using different notation according to the origin of their optimization tasks.

By adding a new virtual root node r to the underlying graph and connecting the possible RNC nodes to r the problem transforms to the planning problem of a single rooted spanning tree with limited depth and with *inhomogeneous* degree constraints. There are some papers in the literature dealing with planning of a minimal cost spanning tree with either of the above constraints (see e. g. [DK97b, DK97a, ACG92]), but finding an algorithm handling both requirements is still a challenge.

On the other hand, our problem can be considered as a version of facility location problems. The problem of facility location is to install some "facilities" at some of the possible places, so that they will be able to serve a number of "clients". The objective is usually to minimize the sum of distances between the clients and the corresponding facilities while satisfying some side constraints, e. g. the number or capacity of facilities is limited. See e. g. [Das95, Dre95] for more details on facility location problems and on the known algorithmic approaches.

Facility location problems give a good model to the planning task of traditional GSM access network topology, for it requires the design of one-level concentrators (for the star concentrator location problem see e.g. [Gav91]). The case of multilevel concentrators is also studied in the literature, though less extensively. In [SZ82, GSD01] the authors examined a problem similar to ours, but without depth and degree constraint and with capacity dependent cost functions. In [Cha98], the case of two-level concentrators is discussed, also without degree constraints.

Finally, our problem can also be considered as an extension of the so-called hub location problem. In this scenario we have a given set of nodes communicating with each other and the amount of the traffic between the node pairs is given by a traffic matrix. Instead of installing a link between every pair of nodes, each node is connected to only a single special node called hub. The hubs are fully interconnected, so the traffic flow between two nodes is realized on a path of length three, where the two intermediate nodes are hubs. The task is to find an optimal placement of hubs, where the cost is composed of the cost of the hubs and the capacity dependent cost of the links. There are several papers examining hub location problems with various side constraints and optimization objectives. A good review on this topic can be found e.g. in [Kli98].

A previous method that is able to a give solution to the problem presented in this chapter can be found in [HSG00]. This algorithm called *TreePlan* finds the number and the places of the RNC devices by Simulated Annealing metaheuristic and connects the

nodes to the RNSs using an extension of Prim's spanning tree algorithm which respects the topological requirements on the tree. This algorithm was used as a reference method in the experimental tests.

5.4 The *Global* Algorithm

The aim of this algorithm is to find the optimum places of the RNC nodes (i.e. to decide which RBSs should be equipped with RNC devices) and to connect each RBS to an RNC directly or indirectly as described in Section 5.2. The basic idea of the proposed approach is the following.

Assuming that the level $l(v)$ is known for each node v, the theoretically minimal cost can be determined for that given level-distribution (see Section 5.4.2). The algorithm considers a series of such distributions, determines the cost for each of them and uses some metaheuristic method to reach the final solution. From among the wide range of metaheuristic methods existing in the literature, Simulated Annealing was chosen for this purpose, but some other local search methods could also be used, especially the so-called Tabu Search scheme [Glo89, Glo90, AL97].

5.4.1 Using Simulated Annealing

In this section the application of Simulated Annealing ([SGJ83, AL97]) to the problem is illustrated. To use Simulated Annealing to a specific optimization problem, an appropriate state space $\mathcal{S}$ corresponding to the possible feasible solutions, a neighborhood-relation between the states and a cost function of each state should be selected. The role of neighborhood-relation is to express the similarity between the elements of the state space. The neighborhood of a state s is typically defined as the set of the states that can be obtained by making some kind of local modifications on s.

Then the Simulated Annealing generates a sequence of feasible solutions $s_0, s_1, \cdots$, approaching to a suboptimal solution as follows. It starts with an arbitrary initial state s_0. In each iteration it chooses a random neighbor s_{i+1} of the last solution s_i and calculates its cost $c(s_{i+1})$, then decides whether it accepts this new solution or rejects it (i.e. $s_{i+1} := s_i$). This iteration is repeated until a certain stop condition fulfills.

If the cost of the new state $c(s_{i+1})$ is lower than $c(s_i)$, the new state is always accepted.

If it is higher then it is accepted with a given probability P_{accept} determined by the value of the deterioration and a global system variable, the so called *temperature* T of the system. In this case P_{accept} is also positive, however, it is an exponentially decreasing function of cost deterioration.

In general, the state s_{i+1} is accepted with probability

$$P_{accept} = min\left\{1, exp\left(-\frac{c(s_{i+1}) - c(s_i)}{T_i}\right)\right\}.$$

The temperature decreases exponentially during the execution, i.e the temperature T_i in the i-th iteration is given by:

$$T_i = T_{i-1} * fact, \quad T_0 = const.$$

where $fact$ is the so called *annealing factor*, which is a number close to 1, typically 0.99-0.9999. The values T_0 and $fact$ are declared in the beginning of the algorithm. The short pseudo-code in Figure 5.3 illustrates the framework of Simulated Annealing.

For an effective Simulated Annealing the following criteria should be met. The state space to be searched should be possibly small; each state should have lot of (meaningful) neighborhoods, allowing to reach the optimum in a low number of steps; the cost of a state should be determined relatively fast.

None of these criteria can be fulfilled easily in case of this planning problem. The most obvious idea for the state space would be the set of all feasible connections. Two such connections would be neighbors, if they can be reached from each other by changing one edge. This solution violates the first two criteria: the state space is very large, and because of the topological constraints, each state has only few neighbors and there are a lot of local minima. Although the cost of a state can be calculated easily, since the exact connections are given in every state, this solution is not usable. Instead, we propose following idea.

The state-space of Simulated Annealing is only the set of all possible distributions of the nodes on the different levels of the graph. Thus a given distribution can be represented as an n-dimensional integer vector s in $\mathcal{S}$.

$$D = \{0, 1, 2, .., l_{tree}\}, \quad \mathcal{S} = D^n$$

The state space with this selection will be much smaller than in the previous case.

```
procedure Sim_Ann(steps, factor, temp, s_0)
{
   curcost :=cost(s_0);
   oldcost :=curcost;
   while (steps > 0)
   {
         Choose a random neighbor s_{i+1} of s_i;
         curcost := cost(s_{i+1});
         if(acceptable(curcost, oldcost, temp))
           oldcost := curcost;
         else s_{i+1} := s_i;
         i := i+1;
         steps := steps - 1;
         temp := temp * factor;
   }
   return s_n;
}
```

Figure 5.3: Structure of the Simulated Annealing algorithm

Two level-distributions are neighbors if they can be reached from each other trough one of the following slight modifications of the current distribution:

- *moving* an arbitrary node onto an adjacent level upwards or downwards
- *swapping* an arbitrary RNC with an arbitrary RBS, that is, moving an RNC to another site

The price for the smaller state space is that the calculation of the cost of a state becomes more difficult. Section 5.4.2 shows how the cost $c(s_i)$ of a given state $s_i \in \mathcal{S}$ can be calculated.

The initial state s_0 of the Simulated Annealing algorithm can be any feasible state. Such a state can be constructed easily by setting each possible nodes to RNC and connecting

the other nodes arbitrarily fulfilling the criteria defined in Section 5.2. The fact that there exists no feasible solution at all can be detected, as well.

5.4.2 Finding the Exact Cost for a Given Level-distribution s_i

Let us assume that the distribution of the nodes on different levels is known. In order to find the optimal connections for this given distribution, the connections between the adjacent levels have to be determined. The main observation is that the connections of the different levels to their parents are independent, so the task of connecting the nodes of a certain level can be performed *separately* for each level. (As there are l_{tree} adjacent level-pairs, the algorithm described now has to be run l_{tree} times in each step of the Simulated Annealing process.)

Generally, for each adjacent level-pair (L_i and L_{i+1}, $i = 0, \cdots, l_{tree} - 1$) a connection has to be found so that

- all nodes in set L_{i+1} are covered
- the maximal degree k of the nodes in set L_i is given. As already mentioned, $k = d_{RNC}$ if $i = 0$, $k = d_{RBS}$ if $i > 0$.

This can be formulated as a special b-matching problem, which can be solved in strongly polynomial time [AMO93, CCPS97].

Definition 5.1 (Bipartite b-matching problem) *Let $G(V, E)$ be a bipartite graph and let $low, upp : V \longrightarrow \mathbb{N}$ be two predefined functions on the set of nodes. A subgraph M of G is called* b-matching *if $low(v) \leq deg_M(v) \leq upp(v)$ for each $v \in V$. We are also given a cost function $c : E \longrightarrow \mathbb{R}$ on the edges. Then the* bipartite b-matching problem *is to find a minimum cost b-matching in G.*

Acceleration of the Algorithm

The quality of the solution found by the Simulated Annealing largely depends on the number of iterations made by the algorithm. However we want to solve a b-matching problem in each iteration, which is — although polynomially solvable — a very time consuming task. Here follows a little trick that can accelerate a lot on the running time

of the algorithm, actually this trick makes our Simulated Annealing/b-matching based algorithm a competitive solution in practice.

The observation is that the b-matching problems we have to solve in each iteration is quite similar to those solved at the previous iteration. Namely, the b-matching problems to be solved is obtained from the problem instances solved at the previous iteration by adding or deleting a node or by moving a node from one partite to the other.

The usual b-matching algorithm is a kind of primal-dual method, i.e. it maintains a feasible dual solution, "partial" primal solution satisfying the so-called slackness condition and at each iteration it improves either of them until the primal side becomes a "full" solution. Instead of starting from scratch, we create a primal and dual solution with the conditions above from the previous optimal solution. First we make the dual feasible by increasing its values in a greedy way if necessary, then we remove the primal solution's edges that would break the duality slackness conditions. Since these solutions are hopefully quite close to the optimum, the algorithm will stop much sooner. The details should be straightforward for the readers familiars with the b-matching algorithms, let us skip then.

5.5 The Single-Tree Problem

In practical planning problems it is often the case that — for geographical, political or economical reasons — the exact number and location of the RNC nodes and the set of RBSs belonging to them is already known in the beginning.

For these reasons this section discusses the special problem where only one *fixed* RNC and a set of RBS nodes are given. Of course the *Global* algorithm can solve this special case, as well, but an alternative method called *Local* algorithm, which finds remarkably better solutions, is introduced. Although this algorithm is slightly slower than the *Global* algorithm, it is efficient for about 200-300 nodes, which is the typical number of RBSs controlled by one RNC.

The *Local* algorithm is based on a branch-and-bound method that finds the *exact optimal solution* for smaller inputs (up to 50-60 nodes). To sum up, the *Local* algorithm can be used for the following purposes.

- Planning a tree of RBSs belonging to a fixed RNC.
- Improving the trees created by an arbitrary previous algorithm.

- Determining the theoretical optimum for smaller inputs.

The simplified single-tree problem can be formulated as follows.

A single RNC and $n-1$ RBSs are given by their coordinates. Note that the problem of positioning the RNC is omitted in this model. The task of the algorithm is to connect all the RBSs directly or indirectly to the RNC. These connections should build a tree rooted in the RNC. This tree has the following characteristics.

- The depth of the tree can be at most l_{tree}.
- The in-degree of RBSs can be at most d_{RBS}.
- The degree of the RNC is at most d_{RNC}.

We remark that this simplified optimization problem is still $\mathcal{NP}$-hard (see Section 5.7 for the proof).

In the next section the branch-and-bound algorithm is described that finds the optimal solution for smaller inputs and then it will be shown how it can help to solve the problem for larger inputs.

5.5.1 Finding the Optimal Solution

The well-known branch-and-bound method is used to find the optimal solution to the problem. The efficiency of the branch-and-bound method mainly depends on the algorithm that computes a lower bound to the problem. It must run fast and — what is more substantial — it should give tight bounds.

The most commonly used techniques are based on some kind of relaxation. The problem is formulated as an ILP problem, and some conditions are relaxed.

One possible way to relax the problem is LP-relaxing. In this case we work with the LP version of the given ILP-formulation by allowing also non-integer solutions, and compute a lower bound for the original ILP optimization problem. The LP problem can be solved efficiently, e.g. using the simplex method. A lot of software packages can do it automatically, the bests (e.g. CPLEX) use many deep techniques to improve the running time of the branch-and-bound algorithms [ILO01]. Unfortunately, in our case these packages fail to find an optimal solution even for inputs having only 15 nodes. The reason is twofold. Firstly, the ILP-formulation of the problem needs too many variables. Secondly, in this

case, there is quite a large gap between the optimal solution and the lower bound provided by the LP-relaxation.

In order to solve the problem for larger inputs two substantial improvements of this technique are presented.

- First, Lagrange-relaxation is used instead of LP-relaxation. In this case it provides significantly tighter lower bounds.
- The other improvement is that we seek with branch-and-bound not for the exact connections but only for the levels of nodes. This significantly reduces the number of the decision variables. After that the optimal connections can be computed as described in Section 5.4.2.

These techniques enable us to find *the optimal* solution of practical examples having up to 50–60 nodes. In case of 20–30 nodes the algorithm is very fast, it terminates usually within some seconds and at most in some minutes.

In the following the problem is formulated as an ILP program, its Lagrange-relaxation is introduced and finally the branch-and-bound technique is described. For the sake of simplicity we examine the case when $l_{tree} = 3$ and $d_{RBS} = 2$, but it is straightforward to extend it to the general one.

ILP-formulation

Before the formulation of the problem, some notation have to be introduced. Let N denote the set of the nodes. Since $l_{tree} = 3$, the nodes except the RNC can be classified as first-, second- or third-level nodes. Let us introduce three sets of binary variables. The variable r_i indicates that the node i is connected directly to the RNC. The variable x_{ij} indicates that the node j on the first level is connected to the node i on the second one. Finally, y_{ij} indicate that the node j on the second level is connected to the node i sitting on the third level.

Furthermore, the cost functions c_i, c_{ij} and d_{ij} are given, where c_i is the cost of a link between node i and the RNC. The cost of a link between the nodes i and j is c_{ij} or d_{ij}, depending on whether it is on the second or third level, respectively.

Thus, the ILP-formulation is the following:

$$\min \sum_{i \in N} c_i r_i + \sum_{i,j \in N} c_{ij} x_{ij} + \sum_{i,j \in N} d_{ij} y_{ij} \tag{5.2a}$$

subject to

$$r_i, x_{ij}, y_{ij} \in \{0,1\} \qquad \forall i,j \in N \qquad (5.2b)$$

$$r_i + \sum_j x_{ij} + \sum_j y_{ij} = 1 \qquad \forall i \in N \qquad (5.2c)$$

$$\sum_i r_i \leq d_{RNC} \qquad (5.2d)$$

$$\sum_i x_{ij} \leq 2r_j \qquad \forall j \in N \qquad (5.2e)$$

$$\sum_i y_{ij} \leq 2\sum_k x_{jk} \qquad \forall j \in N \qquad (5.2f)$$

(5.2a) is the object function to be minimized, (5.2c) expresses that each node resides on exactly one level and that it is connected to the node above itself by exactly one edge. (5.2d) is the degree constraint of the RNC, (5.2e) means that the degree of first-level nodes is at most 2, whereas (5.2f) indicates that the degree of second-level nodes is at most 2.

Lower Bound Computation

For getting lower bound to Problem (5.2a)-(5.2f), the most straightforward way would be to relax the integrality constraints (5.2b) and solve the obtained linear program. However, experiments showed that this approach does not give sufficiently tight bounds to perform a branch-and-bound scheme. Therefore Lagrangian-relaxation (see e.g. [Wol98, AMO93, Das95] for more detail) is applied to obtain a better lower bound.

The straightforward way to do that would be to relax constraints (5.2d), (5.2e) and (5.2f). Then the relaxed problem becomes very easy: for each node i, one must choose the "cheapest" one from the variables r_i, x_{ij} and y_{ij} according to a certain *modified cost function* and set this to 1. However this approach would not lead to success because of two reasons. Firstly, we had to introduce quite a lot ($3|N|$) Lagrange multipliers slowing down the solution of the Lagrange dual problem. Secondly, the relaxed problem is also an (integer) linear program with the property that optimal solutions are automatically integers[3]. However, an easy observation shows that in this case the lower bound obtained by the Lagrange relaxation is equal to that the LP-relaxation gives us. As we mentioned above this lower bound is too weak for our purposes.

[3]More exactly, they can be chosen to be integer

Therefore another kind of Lagrange relaxation is proposed, which needs less multipliers, gives much tighter bounds, but the relaxed problem is more difficult to be solves.

Namely, the constraint (5.2c) is relaxed and built into the object function by using Lagrange-multipliers ($\underline{\lambda}$). Unfortunately the relaxed problem still does not seem to be solvable efficiently, so we first modify the ILP formalization: we relax the binary assumptions on the variables y_{ij}, they are only expected to be *non-negative integers*. In order to reduce the effect of this modification two redundant constraints are added to our formulation.

$$y_{ij} \leq \sum_k x_{jk} \qquad \forall i, j \in N \tag{5.2g}$$

$$y_{ji}^2 y_{ki} = y_{ji} y_{ki}^2 \qquad \forall i, j, k \in N \tag{5.2h}$$

Inequalities (5.2c) and (5.2g) together imply that each y_{ij} is a binary variable. Equation (5.2h) forces that the values y_{ji} of the links incoming to the node i are either equal or zero. (If $y_{ji} \neq 0$ and $y_{ki} \neq 0$ then $y_{ji} = y_{ki}$.) Although equations (5.2h) are not linear, it will not cause a problem because Claim 5.2 still holds.

Now, we are ready to relax equation (5.2c) and build it into the object function. Introducing a new variable λ_i called the Lagrange-multiplier for all nodes, the relaxed version of our problem for a fixed multiplier is the following.

$$L(\underline{\lambda}) := \min \sum_{i \in N} c_i r_i + \sum_{i,j \in N} c_{ij} x_{ij} + \sum_{i,j \in N} d_{ij} y_{ij} + \sum_{i \in N} \lambda_i \left(r_i + \sum_{j \in N} x_{ij} + \sum_{j \in N} y_{ij} \right) - \sum_{i \in N} \lambda_i \tag{5.3a}$$

$$r_i, x_{ij} \in \{0,1\}, \quad y_{ij} \in \mathbb{Z} \qquad \forall i,j \in N \tag{5.3b}$$

$$y_{ij} \geq 0 \qquad \forall i,j \in N \tag{5.3c}$$

$$\sum_i r_i \leq d_{RNC} \tag{5.3d}$$

$$\sum_{i \in N} x_{ij} \leq 2r_j \qquad \forall j \in N \tag{5.3e}$$

$$\sum_{i \in N} y_{ij} \leq 2 \sum_{k \in N} x_{jk} \qquad \forall j \in N \tag{5.3f}$$

$$y_{ij} \leq \sum_k x_{jk} \qquad \forall i,j \tag{5.3g}$$

$$y_{ji}^2 y_{ki} = y_{ji} y_{ki}^2 \qquad \forall i,j,k \tag{5.3h}$$

In spite of the non-linear constraint (5.3h), the following statement, which is well-known for the linear case, still holds with the usual proof.

Claim 5.2 *For any vector $\underline{\lambda}$ of the Lagrange multipliers, $L(\underline{\lambda})$ is a lower bound of the original problem.*

Proof. Let $r_i^\lambda, x_{ij}^\lambda, y_{ij}^\lambda$ be an optimal solution of system (5.3b)–(5.3h) and $r_i^*, x_{ij}^*, y_{ij}^*$ be an optimal solution to system (5.2b)–(5.2f). Then

$$\begin{aligned} L(\underline{\lambda}) &= \sum_i c_i r_i^\lambda + \sum_{ij} c_{ij} x_{ij}^\lambda + \sum_{ij} d_{ij} y_{ij}^\lambda + \sum_i \lambda_i \left(r_i^\lambda + \sum_j x_{ij}^\lambda + \sum_j y_{ij}^\lambda - 1 \right) \\ &\leq \sum_i c_i r_i^* + \sum_{ij} c_{ij} x_{ij}^* + \sum_{ij} d_{ij} y_{ij}^* + \sum_i \lambda_i \left(r_i^* + \sum_j x_{ij}^* + \sum_j y_{ij}^* - 1 \right) \\ &= \sum_i c_i r_i^* + \sum_{ij} c_{ij} x_{ij}^* + \sum_{ij} d_{ij} y_{ij}^* \end{aligned}$$

proves the claim. □

An easy computation shows that the object function (5.3a) is equal to

$$\min \sum_i c_i^{\underline{\lambda}} r_i + \sum_{ij} c_{ij}^{\underline{\lambda}} x_{ij} + \sum_{ij} d_{ij}^{\underline{\lambda}} y_{ij} + \Lambda^{\underline{\lambda}}, \tag{5.4}$$

where $c_i^{\underline{\lambda}} := c_i + \lambda_i$, $c_{ij}^{\underline{\lambda}} := c_{ij} + \lambda_i$, $d_{ij}^{\underline{\lambda}} := d_{ij} + \lambda_i$ and $\Lambda^{\underline{\lambda}} := -\sum_i \lambda_i$.

Claim 5.3 *The optimal solution can be chosen in such a way that there are at most two incoming edges with nonzero y variable for each node $j \in N$.*

Proof. Obviously, it is not worth setting the y_{ij} nonzero if its cost is not less than zero. If there is only one edge with negative cost then the best we can do is to set the corresponding y variable to $\sum_k x_{jk}$, otherwise the best solution is to set the y variables of the two edges having most negative cost to $\sum_k x_{jk}$. □

Now let us discuss the problem of computing $L(\underline{\lambda})$ for a fixed $\underline{\lambda}$ value. First let us assume that $d_{RNC} = \infty$. Observe that

- if the modified cost $c_i^{\underline{\lambda}}$ of a first-level node is negative[4], we can connect it in order to minimize the object function,
- even if its modified cost $c_i^{\underline{\lambda}}$ is positive, it may be advantageous to connect it assuming that we can link it with two second-level nodes, so that the modified total cost of these three nodes is negative
- finally, we can connect it if we find two second-level nodes so that there are third-level nodes, where the modified cost of the whole set of nodes is negative

This idea leads to the following *dynamic programming* algorithm. (Note that here we work in the reverse direction.)

Theorem 5.4 *For any given vector λ, the following algorithm finds an optimal solution to (5.3a)-(5.3h) in linear time.*

1. *For each node j let i_1 and i_2 be the two nodes for which the values $d_{i_1 j}^{\underline{\lambda}}$ and $d_{i_2 j}^{\underline{\lambda}}$ are the two most negative ones.*

   Let $\alpha_j := \min\{0, d_{i_1 j}^{\underline{\lambda}}\} + \min\{0, d_{i_2 j}^{\underline{\lambda}}\}$.

2. *Again, for each node j let i_1 and i_2 be the two nodes for which the values $c_{i_1 j}^{\underline{\lambda}} + \alpha_{i_1}$ and $c_{i_2 j}^{\underline{\lambda}} + \alpha_{i_2}$ are the two most negative ones.*
   Then let $\beta_j := \min\{0, c_{i_1 j}^{\underline{\lambda}} + \alpha_{i_1}\} + \min\{0, c_{i_2 j}^{\underline{\lambda}} + \alpha_{i_2}\}$.

[4]Note that we just relaxed the constraint that guarantees that every node will be connected to another node or to the RNC, so we can reach the cost $\Lambda^{\underline{\lambda}}$ if none of them are connected. Therefore $L(\underline{\lambda})$ should be possibly less than $\Lambda^{\underline{\lambda}}$.

3. The cost of the optimal solution to Problem (5.3a)–(5.3h) is

$$L(\underline{\lambda}) := \sum_i \min\{0, \beta_i + c_i^{\underline{\lambda}}\} + \Lambda^{\underline{\lambda}}. \tag{5.5}$$

If $d_{RNC} < n$, we have to take only the sum of the d_{RNC} smallest (most negative) values of $\min\{0, \beta_i + c_i^{\underline{\lambda}}\}$.

□

To obtain the best lower bound we have to look for the vector $\underline{\lambda}$ which maximizes the function $L(\underline{\lambda})$. That is, we are looking for the value

$$L^* := \max_{\underline{\lambda}} L(\underline{\lambda}) \tag{5.6}$$

Since the function $L(\underline{\lambda})$ is concave (it can be seen from (5.3a) that $L(\underline{\lambda})$ is the minimum of some $\underline{\lambda}$-linear functions), the well-known *subgradient method* can be used to find its maximum. The details are omitted, good surveys of this method can be found for example in [AMO93, BSS93, Das95].

In order to be able to apply the branch-and-bound method we need to solve a modified version of this problem where the depth of some nodes are predefined. Even this case can be handled easily, because for example if the depth of a node i must be equal to 2, then it means that the variables r_i, y_{ik} and x_{ki} must be avoided for all k. This can be performed by setting their corresponding costs to a sufficiently large value. The other cases can be handled similarly.

It is also mentioned that *if all node levels are fixed*, then the lower bound given by the previous algorithm is tight, that is, it is actually *equal* to the optimal solution to Problem (5.2a)–(5.2f). It follows from the fact that in this case the problem reduces to two separate b-matching problems, for which total unimodularity holds [Sch86], so the optimal object function value is equal to the value of the LP-relaxed version of Problem (5.2a)–(5.2f) and in case of LP-programs the Lagrange-relaxation gives the optimal solution.

Branch-and-Bound

This section describes our implementation of the branch-and-bound method. Readers who are not familiar with this technique are referred to e.g. [Wol98, AMO93, Das95].

A set system (N_1, N_2, N_3) is called *partial level-distribution* if $N_i \subseteq N$, $i \in \{1, 2, 3\}$ are pairwise disjoint subsets of the nodes. For this partial level-distribution, let L^{N_1,N_2,N_3} denote the computed Lagrangian lower bound, when each node in N_i is fixed on level i.

The core of this method is the recursive procedure called *SingleTree*, the input of which are the subsets N_1, N_2, N_3 and a bounding number B. The procedure either states that there is no solution cheaper than B for this partial level-distribution or it returns an optimal solution.

Of course, the optimal solution can be obtained by calling *SingleTree* with the empty level-distribution $(\emptyset, \emptyset, \emptyset)$ and $B = \infty$.

The procedure *SingleTree* works as follows. First, we choose a node $j \in N \setminus (N_1 \cup N_2 \cup N_3)$, we fix it onto every level and compute the corresponding lower bounds, that is we determine the values $L_1 = L^{N_1 \cup j, N_2, N_3}, L_2 = L^{N_1, N_2 \cup j, N_3}$ and $L_3 = L^{N_1, N_2, N_3 \cup j}$. Then we examine these partial level-distributions *in increasing order according to their lower bounds*. For each level-distribution, if the lower bound is less than B, we fix the node j on the corresponding level and call *SingleTree* with the new N_i sets. If it returns a new solution, then we update B with the cost of this solution and repeat this with the other level-distributions. Finally, if some of these three calls of *SingleTree* return a solution, we return the best one, otherwise we state that there is no solution with cost less than B.

We also keep an eye on the number of nodes on each level, and if the topological constraints are not fulfilled, we go to the next step automatically.

An easy modification of this method enables to give an approximation algorithm to our problem. The idea is to skip a possible partial fixing not only if the lower bound is worse than the best solution but also if they are close enough to each other. Namely an error factor err is introduced and a partial fixing is examined only if the corresponding lower bound is less than $\frac{1}{1+err}B$. This ensures that the resulted solution is worse at most by the factor err than the optimum.

5.5.2 The *Local* Algorithm

The heart of the algorithm is the optimizer routine of the previous section which is able to find the *theoretical optimum* for small networks. So in each step we select a subset of nodes, find the *optimal* solution for this subproblem and get near the global optimum

```
function SingleTree(N_1, N_2, N_3, B)
{
  if (2 · |N_1| < |N_2| or 2 · |N_2| < |N_3|) then return ∞
  else {
    if (N = N_1 ∪ N_2 ∪ N_3) then return L^{N_1,N_2,N_3}
    else {
      j ∈ N \ (N_1 ∪ N_2 ∪ N_3)
      C_1 := (N_1 + j, N_2, N_3)
      C_2 := (N_1, N_2 + j, N_3)
      C_3 := (N_1, N_2, N_3 + j)
      L_1 := L^{C_1}, L_2 := L^{C_2}, L_3 := L^{C_3}
      ((L'_1, C'_1), (L'_2, C'_2), (L'_3, C'_3)) := sort-by-L((L_1, C_1), (L_2, C_2), (L_3, C_3))
      B* := B
      if (L'_3 < B*) then B* = min{B*, SingleTree(C'_3, B*)}
      if (L'_2 < B*) then B* = min{B*, SingleTree(C'_2, B*)}
      if (L'_1 < B*) then B* = min{B*, SingleTree(C'_1, B*)}
      if (B* < B) then return  B*
                  else return ∞
    }
  }
}
```

Figure 5.4: The SingleTree algorithm

iteratively.

This idea is also motivated by the geometrical fact that the connections of distant nodes do not affect each other significantly, i.e. every node is probably linked to a nearby one. Therefore it is a good approximation of the optimum if the connections of small subsets of close RBSs are determined independently.

The algorithm begins with an initial solution and in every step it forms a group of RBSs (H), finds the *optimal* connections for H and approaches the global optimum by cyclical local optimization. In Figure 5.5 the result of the first step can be seen. In the general step we choose the next subset according to a strategy described in Section 5.5.2 and make corrections to it. The algorithm terminates if it could not improve the network through a

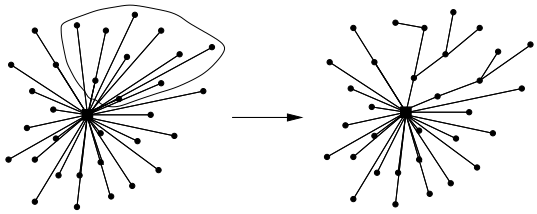


Figure 5.5: The first step of the *Local* algorithm. In this example it is assumed that d_{RNC} is high enough to put every node in L_1 in the initial solution.

whole cycle.

This method is an effective implementation of the so-called *best local improvement method* [AL97]. In this problem the state space is the possible set of connections. Two states A and B are adjacent, if we can reach B from A by changing the connections inside the set H. Note that this local search method does not use elementary steps, it makes rather complex improvement to the graph. Since the typical size of H is 20–25 nodes, the number of possible connections inside H is about 10^{20}, i.e. each state has a huge number of adjacent states. Therefore a local optimum can be reached from every state in few steps. The power of the algorithm is that in spite of the large number of neighbors the optimizer function chooses the *best* one in each step.

Finding an Initial Solution

The algorithm can be utilized to improve the solution given by a previous one, hence in this case the starting solution is given. However, the algorithm can also work independently, so it should be capable of finding an initial solution satisfying the conditions.

- The easiest way to do that is to put d_{RNC} randomly chosen RBSs in the set L_1, $d_{RNC} \cdot d_{RBS}$ randomly chosen ones in set L_2, etc. This trivial solution satisfies the constraints.

- A more sophisticated way to find an initial solution is the following: as described in Section 5.5.1 the core optimizer can be configured to return the theoretical optimum or work with a predefined error rate (err). If err is high enough, the optimization becomes very fast, but of course less effective. So by setting the error rate high in the

first cycle (i.e. until every RBS is at least once picked in H) we could quickly find a considerably better starting solution than the previously mentioned trivial one.

Selection of H

In each step of the algorithm a new set H is selected and given as input for the optimizer. On the one hand, H should consist of close RBSs, on the other hand, it should contain whole subtrees only[5], each rooting in a first level RBS. This is important because during optimization the connections in H will be changed, so otherwise it would be possible to make unconnected RBSs.

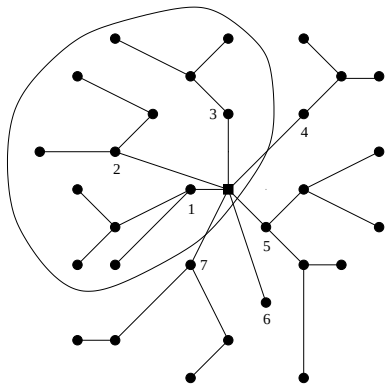


Figure 5.6: The selection of H

To satisfy these conditions we select adjacent subtrees of the current network rooting in a first-level node until a predefined upper bound (h_{max}) for the size of H has been reached. The first-level RBSs are ordered according to their location (clockwise around the RNC). An example for such a selection can be seen in Figure 5.6. The selection starts in every step with another first-level node, which guarantees the variety of H.

Since the core optimizer solves an $\mathcal{NP}$-hard problem, its speed is very sensitive to the size of its input, i.e. $|H|$. Choosing the parameter h_{max} at the beginning of the algorithm, we can set an upper bound for $|H|$. Of course the higher $|H|$ is, the better the result will be, but the slower the algorithm becomes.

[5]Of course the RNC is always in H.

n	*TreePlan*	*Global*	Iteration	*Global* vs. *TreePlan*	*Global+Local*	*Global+Local* vs. *TreePlan*
100	4736	4363	19880	7.9%	4168	12.0%
200	9048	8757	20905	3.2%	8485	6.2%
300	13534	13274	20879	2.0%	12747	5.8%
400	18389	18040	19439	1.9%	17005	7.5%
500	23037	21943	22772	4.7%	21293	7.6%
600	27533	26662	29872	3.2%	25759	6.4%
1000	45686	43916	28596	3.9%	42645	6.7%

Table 5.2: Results of the algorithms (with $d_{RNC} = \infty$) compared with *TreePlan*.

5.6 Numerical Results

In our empirical tests we used the *TreePlan* algorithm [HJS99, HSG00] as a reference method since to our knowledge this is the only method in the literature that is able to solve problems similar to ours.

First, the *Global* algorithm was executed on several test cases, then in a second phase the *Local* algorithm was used to improve the results of the *Global* algorithm. (Note that the *Local* algorithm can also be used to re-design the trees of any planning algorithms, e.g. the *Global* algorithm.) Both results were compared to *TreePlan*.

Although our algorithms can work with arbitrary parameters, the values $d_{RBS} = 2$ and $l_{tree} = 3$ are fixed in our test cases, for these values were the accepted values in the UMTS architecture at the time this research was made. The cost of an RNC node, $cost_{RNC}$ was set to 500 during all tests, and length proportional link cost function was used. In order to be able to compare the numerical results to those of *TreePlan*, d_{RNC} was set to ∞ in the comparative test cases. After these we present some tests on the *Global* algorithm with different d_{RNC} values.

The *Global* algorithm terminates if there is no new state accepted during a predefined number of iterations (K) in the simulated annealing. This frees the tester to explicitly set the number of iterations in the simulated annealing for different initial temperature and annealing factor values. In the following results the annealing factor $fact$ was set to 0.9995, the initial temperature T_0 was 20000 and K was set to 2000.

The results of our tests can be seen in Table 5.2. The first column is the size of the network, then the total network cost planned by *TreePlan* and by the *Global* algorithm is compared. The fourth column contains the number of iterations made by the *Global* algorithm. In the sixth column the improvement of the *Local* algorithm on the *Global* algorithm can be seen, and finally the total improvement vs. *TreePlan*.

In *all* of the test cases the *Global* algorithm alone can reduce the cost of the UMTS network; moreover the *Local* algorithm was able to make further improvements on the trees in *every* test. The size of these trees varied from 35 to 200, showing that the *Local* algorithm is appropriate for practical sized problems. As one may expect the best results can be found by the combination of the two new methods, which gave an improvement of 6-7%, in extreme cases even of 12% compared to the former approach, *TreePlan*.

Table 5.3 contains the cost of the same network of size 700 with different d_{RNC} values planned by the *Global* algorithm. Naturally the cost of the network should be monotonously increasing with the decrease of d_{RNC}, as the problem is more constrained. The results clearly reflect this characteristic.

The execution times of the *Global* algorithm on a 400MHz PC[6] running SuSE Linux 7.1 can be seen in Figure 5.7. (On the x axis the number of nodes, on the y axis the execution time of an iteration in milliseconds can be seen.)

[6]Note that a PC with ten times higher computational capacity is widely available today. Moreover, the C++ compilers has also shown a dramatic improvement considering the efficiency of the generated executable machine code.

n	d_{RNC}	Cost	Time
700	100	30658	21m03s
700	50	30658	18m38s
700	25	30737	18m37s
700	10	31506	19m24s
700	5	36956	17m16s
700	3	45263	15m13s
700	2	56437	14m21s
700	1	80860	15m25s

Table 5.3: Results of the *Global* algorithm with different d_{RNC} values.

The number of iterations highly depends on the initial parameters and the termination criterion of the Simulated Annealing. In case of the above tests the number of iterations was in the range of 20-30.000. Of course, our algorithms are slower than the former approach according to the more complex computation. However, in a typical network design process one would run the planning algorithm a couple of times only, hence an execution time of about 2.5 hours for a network with 1000 nodes (which is the size of usual UMTS networks) should be acceptable.

We also tested the *Local* algorithm separately. The subgradient method applied in the tests was as in e.g. [Das95]. Our objective was to estimate the difference of the results of the *Local* algorithm and the optimum. Therefore we calculated the lower bound of several test instances as described in Section 5.2 and compared it to the result of the *Local* algorithm. We considered 4 problem sizes each with 10 different networks. Table 5.4 presents the average/minimum/maximum difference between the lower bound and the *Local* algorithm with $h_{max} = 24$. This means, that the *Local* algorithm gives around 5-7% approximation of the optimum value in average for practical examples of 25-200 nodes.

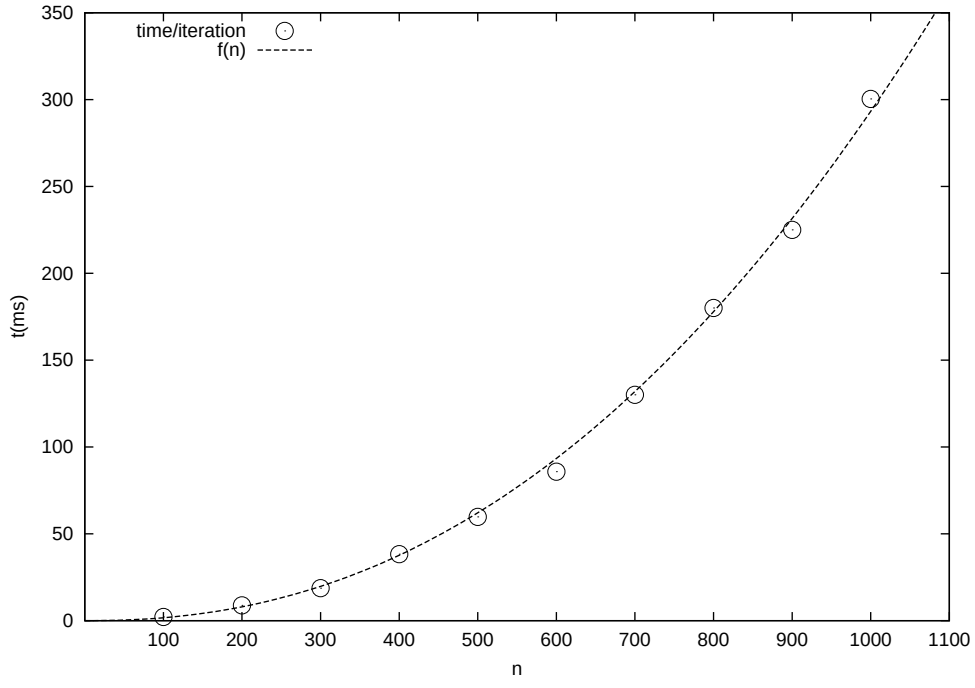


Figure 5.7: The circles indicate the execution times of a single iteration of the *Global* algorithm. The dotted line shows the function $f(n) = 5.588 \cdot 10^{-5} \cdot n^{2.24}$.

n	AVG(%)	MIN(%)	MAX(%)
25	2.60	1.01	8.04
50	5.88	4.35	7.65
75	5.42	3.79	8.55
100	6.44	5.35	8.36
150	6.87	5.89	8.35
200	6.79	5.87	7.83

Table 5.4: The percentage of the difference between the lower bound and the result of the *Local* algorithm.

n	Optimum	*Local* algorithm
40	1190.8	1190.8
40	1244.9	1244.9
40	1232.3	1232.3
40	1209.9	1216.3
50	1668.0	1677.1

Table 5.5: Comparing the *Local* algorithm with $h_{max} = 21$ to the optimum for small inputs.

Following test shows, that in small problems the actual difference is even smaller. Namely one may sets h_{max} to n to get the global optimum. This is of course feasible for smaller n values only. Table 5.5 shows the results of comparing the *Local* algorithm with $h_{max} = 21$ with the optimum value—also computed with the *Local* algorithm with $h_{max} = n$. The table shows 4 different networks with 40 nodes and one network with 50 nodes. One can clearly see that for small inputs the *Local* algorithm gives almost always optimum.

5.7 NP-hardness of the single-tree problem

Claim 5.5 *Let $G(V, E)$ denote an undirected connected complete graph, $RNC \in V$ be a given vertex, and $d_{RNC} \in \mathbb{N}^+$. Moreover we are given the function $c : E \to \mathbb{R}^+$ determining the cost of the edges. Then the problem of finding a minimal cost spanning tree with the*

following properties is $\mathcal{NP}$-hard:

- *every node can be reached from the RNC in a path of length at most 3,*
- *every node except the RNC has degree at most 3,*
- *RNC has degree at most d_{RNC}.*

Proof. We show a Karp-reduction of the well-known $\mathcal{NP}$-complete 3–*uniform set covering* problem (referred as Problem [SP5] in [GJ79]) to this one. The input of the 3–uniform set covering problem is an integer number l and a class of sets $\mathcal{H} := \{X_1, X_2, \cdots, X_n\}$ over an underlining set H $(|H| \geq 3)$, such that

$$X_i \subseteq H, \quad |X_i| = 3 \quad \forall i \in \{1, ..., n\}$$

and

$$\bigcup_{i=1}^{n} X_i = H.$$

The problem is to determine whether there exist at most l sets in $\mathcal{H}$ still covering H.

This problem is reduced to the single-tree problem as it follows. First, $|H|$ nodes are defined corresponding the elements of H and six additional nodes $a_i, b_i, c_i, d_i, e_i, f_i$ are defined for each element X_i of $\mathcal{H}$. Moreover, an RNC node is also defined.

The link cost function c is defined as follows: only the edges illustrated in Figure 5.8 have finite cost, the others have infinite, i. e. so large that they will surely not be in the minimal spanning tree (because there exists a spanning tree without these edges). The thin edges in every subgraph have cost 0, and the thick edge has cost 1. There is a natural one to one mapping between the sets X_i and the thick edges. Obviously the cost of a spanning tree in G arises only from thick edges, i. e. it is always an integer number.

It is going to be shown that the cost of the minimal cost spanning tree satisfying the above criteria is equal to the minimal number of sets covering H completely.

A trivial observation is that every path of length at most 3 between a vertex $v \in H$ and the RNC must contain exactly one thick edge. Furthermore, through a particular thick edge at most the vertices of its set X_i can be reached through paths of length at most three. As a consequence of these, the sets corresponding to the thick edges of a given feasible tree will cover all vertices in H.

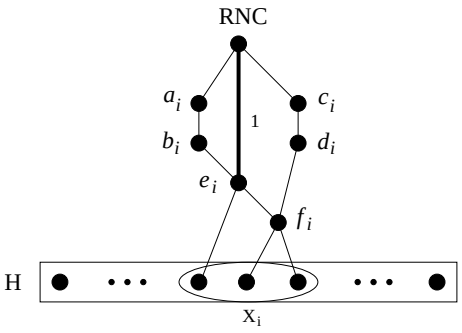
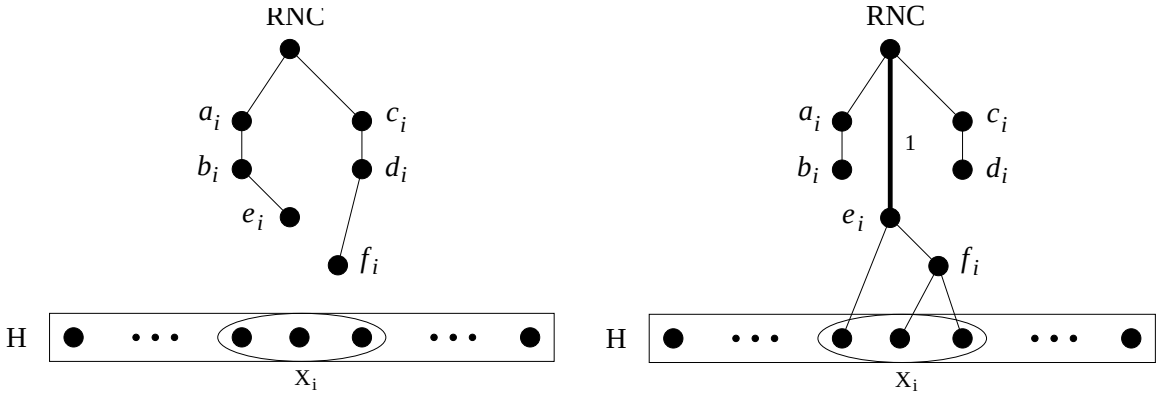


Figure 5.8: The subgraph associated with each set X_i



(a) The part of the spanning tree belonging to the X_i sets *not* in the minimal covering set. The cost is 0.

(b) The part of the spanning tree belonging to the X_i sets in the minimal covering set. The cost is 1.

Figure 5.9: Spanning tree with cost r.

On the other hand, let us suppose that $X_{i_1} \cup X_{i_2} \cup \cdots \cup X_{i_r} = H$. To prove the other direction, we show a spanning tree with cost r. In each subgraph belonging to a set X_i a subtree is constructed according to Figure 5.9(a) or Figure 5.9(b) with cost 0 or 1 depending on whether the set X_i is among the covering ones.

The union of these subtrees covers every vertex in G, since the covering set covers every vertex in H and the $V \setminus H$ vertices are covered by the subtrees in both cases. It may happens that this union contains cycles, but by simply neglecting the superfluous edges a spanning tree is obtained with cost at most r.

□

Bibliography

The Author's Publications

[GHJ04] István Gódor, János Harmatos, and Alpár Jüttner. Inverse shortest path algorithms in protected UMTS access networks. *Computer Communications*, 28(7):765–772, 2004. on-line available: `http://authors.elsevier.com/sd/article/S0140366404003779`.

[HJ05] János Harmatos and Alpár Jüttner. Resilient and qos aware setting of OSPF parameters in multi domain networks. In J.E. Mitchell and D.W. Faulkner, editors, *10th European Conference on Networks and Optical Communications*, pages 241–248. University College London, Communications Engineering Doctorate Centre, july 2005.

[HJS99] János Harmatos, Alpár Jüttner, and Áron Szentesi. Cost-based UMTS transport network topology optimisation. In *ICCC'99 International Conference on Computer Communication*, Tokyo, Japan, September 1999.

[JHO+00] Alpár Jüttner, János Harmatos, Dániel Orincsay, Balázs Szviatovszki, and Áron Szentesi. On-demand optimization of label switched paths in MPLS networks. In *9th International Conference on Computer Communications and Networks, ICCCN 2000*, pages 107–113, Las Vegas, USA, October 2000.

[JM05] Alpár Jüttner and Ádám Magi. Tree based broadcast in ad hoc networks. *Mobile Networks and Applications (MONET)*, Special Issue on "WLAN Optimization at the MAC and Network Levels", 2005. on-line available: `http://dx.doi.org/10.1007/s11036-005-3368-5`.

[JOF05] Alpár Jüttner, András Orbán, and Zoltán Fiala. Two new algorithms for UMTS access network topology design. *European Journal of Operational Research*, 164(2):456–474, July 2005.

[JSHP00] Alpár Jüttner, Áron Szentesi, János Harmatos, and M. Pióro. On solvability of an OSPF routing problem. In *15th Nordic Teletraffic Seminar*, pages 1–9, Lund, Sweden, August 2000.

[JSMR01] Alpár Jüttner, Balázs Szviatovszki, Ildikó Mécs, and Zsolt Rajkó. Lagrange relaxation based method for the QoS routing problem. In *Infocom*. IEEE, April 2001.

[JSS03] Alpár Jüttner, István Szabó, and Áron Szentesi. On bandwidth efficiency of the hose resource management model in virtual private networks. In *Infocom*. IEEE, April 2003.

[Jüt01] Alpár Jüttner. Optimization with additional variables and constraints. EGRES report TR-2001-17, Egerváry Research Group, Hungary, 2001.

[Jüt03] Alpár Jüttner. On budgeted optimization ploblems. In *3th Hungarian-Japanese Symposium on Discrete Mathematics and Its Applications*, Tokyo, Japan, January 2003.

[Jüt05a] Alpár Jüttner. On budgeted optimization ploblems. *SIAM Journal on Discrete Matemathics*, 2005. to appear.

[Jüt05b] Alpár Jüttner. On resource constrained optimization problems. In *4th Japanese-Hungarian Symposium on Discrete Mathematics and Its Applications*, Budapest, Hungary, June 2005.

[Jüt05c] Alpár Jüttner. Optimization with additional variables and constraints. *Operations Research Letters*, 33(3):305–311, May 2005.

[PJH+01a] M. Pióro, A. Jüttner, J. Harmatos, Á. Szentesi, P. Gajowniczek, and A. Myslek. Topological design of telecommunication networks — nodes and links localization under demand constraints. In *17th ITC (International Teletraffic Congress)*, Salvador da Bahia, Brazil, September 2001.

[PJH+01b] M. Pióro, Alpár Jüttner, János Harmatos, Áron Szentesi, and A. Myslek. Topological design of MPLS networks. In *GLOBECOM*, San Antonio, Texas, USA, November 2001.

[PSH+02] Michał Pióro, Áron Szentesi, János Harmatos, Alpár Jüttner, P. Gajowniczek, and S. Kozdorowski. On open shortest path first related network optimisation problems. *Journal of Performance Evaluation*, 48(1-4):201–223, May 2002.

[PSHJ00] M. Pióro, Áron Szentesi, János Harmatos, and Alpár Jüttner. On OSPF related network optimization problems. In *8th IFIP Workshop on Performance Modelling and Evaluation of ATM & IP Networks*, pages 70/1–70/14, Ilkley, UK, July 2000. IFIP.

[SSJ01] Balázs Szviatovszki, Áron Szentesi, and Alpár Jüttner. Minimizing rerouting in MPLS networks with preemption-aware constraint-based routing. In *SPECTS 2001*, 2001. 2001 International Symposium on Performance Evaluation of Computer and Telecommunication Systems.

[SSJ02] Balázs Szviatovszki, Áron Szentesi, and Alpár Jüttner. On the effectiveness of restoration path computation methods. In *International Conference on Communications*, New York, apr 2002. IEEE.

[SSJ03] Balázs Szviatovszki, Áron Szentesi, and Alpár Jüttner. Minimizing re-routing in MPLS networks with preemption-constraint-based routing. *special issue of the Computer Communications Journal on Advances in Performance Evaluation of Computer and Telecommunications Networking*, 25(11-12):1076–1084, May 2003.

[XTXJ05] Ying Xiao, Krishnaiyan Thulasiraman, Guoliang Xue, and Alpár Jüttner. The constrained shortest path problem: Algorithmic approaches and an algebraic study with generalization. *AKCE International Journal of Graphs and Combinatorics*, 2(2):63–86, December 2005.

Other References

[ACG92] N.R. Achuthan, L. Caccetta, and J.F. Geelen. Algorithms for the minimum weight spanning tree with bounded diameter problem. *Optimization: Techniques and Applications*, pages 297–304, 1992.

[AL97] Emile Aarts and Jan Karel Lenstra. *Local Search in Combinatorial Optimization*. John Wiley & Sons, Inc., 1997.

[AMO93] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows*. PRENTICE HALL, 1993.

[AO95] R.K. Ahuja and J.B. Orlin. A capacity scaling algorithm for the constrained maximum flow problem. *Networks*, 25:89–98, 1995.

[BDK93] R.E. Burkard, K. Dlaska, and Bettina Klinz. The quickest flow problem. *ZOR Methods and Models of Operations Research*, 37:1–58, 1993.

[BG96] David Blokh and George Gutin. An approximation algorithm for combinatorial optimization problems with two parameters. *Australasian J. Combin.*, 14:157–164, 1996.

[BP95] R.E. Burkard and U. Pferschy. The inverse-parametric knapsack problem. *European Journal of Operations Research*, 83:376–393, 1995.

[BSS93] Mokhtar S. Bazaraa, Hanif D. Sherali, and C.M. Shetty. *Nonlinear Programming - Theory and Algorithms*. John Wiley & Sons, Inc, 1993.

[CCPS97] W.J. Cook, W.H. Cunningham, W. Puleyblank, and A. Schrijver. *Combinatorial Optimization*. Series in Discrete Matehematics and Optimization. Wiley-Interscience, Dec 1997.

[Cha98] P. Chardaire. Upper and lower bounds for the two-level simple plant location problem. *Annals of Operations Research*, 1998.

[CM93] E. Cohen and N. Megiddo. Strongly polynomial-time and nc algorithms for detecting cycles in dynamic graphs. *Journal of the Association for Computing Machinery*, 40:791–832, 1993.

[CN98] Shigang Cheng and Klara Nahrstedt. On finding multi-constrained paths. In *ICC'98*, Atlanta, Georgia, 1998.

[Das95] Mark S. Daskin. *Network and Discrete Location*. Jon Wiley and Sons Inc., 1995.

[DK97a] N. Deo and N. Kumar. Computation of constrained spanning trees: A unified approach. In P.M. Pardalos, D.W. Hearn, and W.W. Hager, editors, *Network Optimization*. Springer-Verlag, 1997.

[DK97b] N. Deo and N. Kumar. Constrained spanning tree problems: Approximate methods and parallel computation. In *Network Design: Connectivity and Facilities Location*, pages 191–219. DIMACS, April 1997.

[dNvM98] H. de Neve and P. van Mieghem. A multiple quality of service routing algorithm for PNNI. In *ATM'98 Workshop*, pages 306–314, Fairfax, Virginia, May 1998. IEEE.

[Dre95] Zvi Drezner, editor. *Facility Location: A Survey of Applications and Methods*. Spinger Series in Operational Research. Springer-Verlag, 1995.

[EG77] J. Edmonds and R. Giles. A min-max relation for submodular functions on graphs. *Annals of Discrete Mathematics*, 1:185–204, 1977.

[EK72] J. Edmonds and R.M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the Association for Computing Machinery*, 19:248–264, 1972.

[FH75] D.R. Fulkerson and G.C. Harding. Maximizing the minimum source-sink path subject to a budget constraint. *Mathematical Programming*, 13:116–118, 1975.

[FSO97] G.N. Frederickson and R. Solis-Oba. Efficient algorithms for robustness in matroid optimization. In *Proceedings of the Eight Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 659–668, 1997.

[FSO98] G.N. Frederickson and R. Solis-Oba. Algorithms for measuring perturbability in matroid optimization. *Combinatorica*, 18:503–518, 1998.

[FSO99] G.N. Frederickson and R. Solis-Oba. Increasing the weight of minimum spanning trees. *Journal of Algorithms*, 33:244–266, 1999.

[Fuj91] S. Fujishige. *Submodular Functions and Optimization.* Elsevier Science Publishers B.V., 1991.

[Ful59] D.R. Fulkerson. Increasing the capacity of a network: the parametric budget problem. *Management Science*, pages 472–483, 1959.

[Gav91] B. Gavish. Topological design of telecommunication networks — local access design methods. *Annals of Operations Research*, 1991.

[GJ79] M.R. Garey and D.S. Johnson. *Computers and Intractability — A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, New York, 1979.

[Glo89] F. Glover. Tabu search - part i and ii. *ORSA Journal on Computing*, 1(3), 1989.

[Glo90] F. Glover. Tabu search - part ii. *ORSA Journal on Computing*, 2(1), 1990.

[GLS81] M. Grötschel, L. Lovász, and A. Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1:169–197, 1981.

[GM03] Liang Guo and Ibrahim Matta. Search space reduction in QoS routing. *Computer Networks*, 41(1):73–78, Jan 2003.

[GSD01] A. Girard, B. Sanso, and L. Dadjo. A tabu search algorithm for access network design. *Annals of Operations Research*, 106:229–26, 2001.

[GT89] A.V. Goldberg and R.E. Tarjan. Finding minimum-cost circulations by canceling negative cycles. *Journal of the Association for Computing Machinery*, 36:873–886, 1989.

[Har77] G.C. Harding. *Some budgeted optimization problems and the edge disjoint branchings problem.* PhD thesis, Cornell University, 1977.

[Has92] R. Hassin. Approximation schemes for the restricted shortest path problem. *Mathematics of Operations Research*, 17(1):36–42, Feb 1992.

[HK70] M. Held and R. Karp. The traveling salesman problem and minimum spanning trees. *Operations Research*, 18:1138–1162, 1970.

[HK71] M. Held and R. Karp. The traveling salesman problem and minimum spanning trees, part ii. *Mathematical Programming*, 6:62–88, 1971.

[HSG00] János Harmatos, Áron Szentesi, and István Gódor. Planning of tree-topology UMTS terrestrial access networks. In *PIMRC*, England, London, Sept 2000.

[HZ80] G. Handler and I. Zang. A dual algorithm for the constrained shortest path problem. *Networks*, 10:293–310, 1980.

[I+96] Atsushi Iwata et al. ATM routing algorithms with multiple QoS requirements for multimedia internetworking. *IEICE Transactions on Communication*, E79-B:999–1007, August 1996.

[IAK98] Kenji Ishida, Kitsutaro Amano, and Naoki Kannari. A delay-constrained least-cost path routing protocol and the synthesis method. In *Proceedings of the 5th International Conference on Real-Time Computing Systems and Applications*, pages 58–65. IEEE, 1998.

[ILO01] ILOG. *CPLEX 7.1 User's Manual*, 2001.

[Jaf84] Jeffrey Jaffe. Algorithms for finding path with multiple constraints. *Networks*, 14:95–116, 1984.

[JO99] Balázs Gábor Józsa and Dániel Orincsay. Random graph generator (for telecommunication networks). Technical report, Ericsson Research, Hungary, April 1999.

[Kal98] Paul Kallenberg. Optimization of the fixed part GSM networks using simulated annealing. In *Networks 98*, Sorrento, Oct 1998.

[Kha80] L.G. Khachian. Polynomial algorithms in linear programming. *Zhurnal Vychislitelnoi Matematiki i Matematicheskoi Fiziki*, 20:53–72, 1980.

[Kli98] J.G. Klincewicz. Hub location in backbone/tributary network design: a review. *Location Science*, 6:307–335, 1998.

[KP82] R.M. Karp and C.H. Papadimitriou. On linear characterization of combinatorial problems. *SIAM Journal of Computing*, 11:620–632, 1982.

[L+94] W.C. Lee et al. Multi-criteria routing subject to resource and performance constraints. *ATM Forum*, 94(0280), March 1994.

[LO98] D.H. Lorenz and A. Orda. QoS routing in networks with uncertain parameters. *IEEE/ACM Transactions on Networking*, 6(6):768–778, Dec 1998.

[LORS00] Dean H. Lorenz, Ariel Orda, Danny Raz, and Yuval Shavitt. Efficient QoS partition and routing of unicast and multicast. In *IWQoS 2000*, June 2000.

[Mar97] Brian G. Marchent. Third generation mobile systems for the support of mobile multimedia based on ATM transport. In *5th IFIP Workshop*, Ilkley, July 1997.

[Meg79] N. Megiddo. Combinatorial optimization with rational objective functions. *Mathematics of Operations Research*, 4:414–424, 1979.

[Meg83] N. Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *Journal of the Association for Computing Machinery*, 30:852–865, 1983.

[Meg84] N. Megiddo. Linear programming in linear time when the dimension is fixed. *Journal of the Association for Computing Machinery*, 31:114–127, 1984.

[MZ00] K. Mehlhorn and M. Ziegelmann. Resource constrained shortest paths. In *8th Annual European Symposium on Algorithms (ESA)*, pages 326–337, LNCS 1879, 2000.

[NPT92] Carolyn H. Norton, S.A. Plotkin, and Éva Tardos. Using separation algorithms in fixed dimension. *Journal of Algorithms*, 13(1):79–98, 1992.

[OP98] Tero Ojanperä and Ramjee Prasad. *Wideband CDMA for Third Generation Mobile Communication.* Artech House Publishers., 1998.

[PCS98] C. Pornavalai, G. Chakraborty, and N. Shiratori. Routing with multiple QoS requirements for supporting multimedia applications. *Journal of High Speed Networks*, 1998.

[PR81] M.W. Padberg and M.R. Rao. The Russian method for linear programming III.: Bounded integer programming. Technical Report 81-29, Graduate School of Business Administration, New York University, 1981.

[Rad93] T. Radzik. Parametric flows, weighted means of cuts, and fractional combinatorial optimization. In P. Pardalos, editor, *Complexity in Numerical Optimization*, pages 351–386. World Scientific, 1993.

[Rad98] Tomasz Radzik. Fractional combinatorial optimization. In DingZhu Du and Panos Pardalos, editors, *Handbook of Combinatorial Optimization*. Kluwer Academic Publishers, dec 1998.

[RS00] Danny Raz and Yuval Shavitt. Optimal partition of QoS requirements with discrete cost functions. In *INFOCOM 2000*, 2000.

[Sch86] A. Schrijver. *Theory of Linear and Integer Programming*. J. Wiley & Sons, 1986.

[SGJ83] S.Kirkpatrick, C.D. Gelatt, and M.P. Vecchi JR. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[SM88] F. Shahrokhi and D.W. Matula. The maximum concurrent flow problem. Technical Report CSR-183, Dept. of Computer Science, New Mexico Tech., 1988.

[SRV97] Hussein F. Salama, Douglas S. Reeves, and Yannis Viniotis. A distributed algorithm for delay-constrained unicast routing. In *Infocom'97*, Kobe, Japan, April 1997. IEEE.

[SZ82] G.M. Schneider and Mary N. Zastrow. An algorithm for design of multilevel concentrator networks. *Computer Networks*, 6:1–11, 1982.

[Tar86] Éva Tardos. A strongly polynomial algorithm for solving combinatorial linear programs. *Operations Research*, pages 250–256, 1986.

[UMT98] UMTS Forum. *The Path towards UMTS - Technologies for the Information Society*, 1998.

[WC95] Zheng Wang and Jon Crowcroft. Bandwidth-delay based routing algorithms. In *GlobeCom'95*, Singapore, nov 1995. IEEE.

[Wid94] R. Widyono. The design and evaluation of routing algorithms for real-time channels. Technical Report TR-94-024, University of California at Berkeley, June 1994.

[Wol98] Laurence A. Wolsey. *Integer Programming.* John Wiley & Sons, Inc., 1998.

[Zie01] Mark Ziegelmann. *Constrained Shortest Paths and Related Problems.* PhD thesis, Universität des Saarlandes, July 2001.

Index