Constrained Shortest Paths and Related Problems

Dissertation
zur Erlangung des Grades
des Doktors der Ingenieurswissenschaften (Dr.-Ing.)
der Naturwissenschaftlich-Technischen Fakultät I
der Universität des Saarlandes

von

Mark Ziegelmann

Saarbrücken
30. Juli 2001

Datum des Kolloquiums: 30. Juli 2001

Dekan der Naturwissenschaftlich-Technischen Fakultät I:
Professor Dr. Schulze-Pillot-Ziemen

Gutachter:
Professor Dr. Kurt Mehlhorn, MPI für Informatik, Saarbrücken
Professor Dr. Ulrich Lauther, Siemens AG, München

Kurzzusammenfassung

Das klassische Kürzeste-Wege-Problem, bei dem man einen Pfad minimaler Kosten zwischen zwei Knoten eines Graphen sucht, ist effizient in polynomieller Zeit lösbar. In vielen praktischen Anwendungen wollen wir jedoch, dass ein Pfad bestimmte Budget- oder Resourcenbedingungen erfüllt. Dies ist das Kürzeste-Wege-Problem mit Nebenbedingungen, das zur Klasse der "schweren" Probleme gehört, für die wir keinen polynomiellen Algorithmus kennen. In dieser Arbeit stellen wir eine 2-Schritt-Methode für das Kürzeste-Wege-Problem mit Nebenbedingungen vor. Wir lösen zuerst eine Relaxierung des Problems und erhalten eine obere und untere Schranke, um dann durch Schließen der Lücke durch geschicktes Auflisten von Pfaden das Optimum zu bestimmen. Wir vergleichen bekannte und neue Verfahren sowohl theoretisch als auch experimentell. Die 2-Schritt-Methode ist auch auf eine allgemeinere Klasse von Netzwerkoptimierungsproblemen mit Nebenbedingungen anwendbar. Wir illustrieren die generische Methode anhand einiger Beispiele. Danach stellen wir unser Softwarepaket CNOP vor, das dieses generische Verfahren implementiert, sowie alle bekannten Verfahren für das Kürzeste-Wege-Problem mit Nebenbedingungen zur Verfügung stellt.

Abstract

The classical shortest path problem, to find a path of minimal cost between two nodes in a graph, is efficiently solvable in polynomial time. However, in many applications we also have additional budget or resource constraints on a path. This problem is known as constrained shortest path problem and unfortunately belongs to the class of "hard" problems for which no polynomial time algorithm is known. In this thesis, we propose a 2-step method for the constrained shortest path problem. We first solve a relaxation to get upper and lower bounds and then close the gap with clever path ranking to obtain the exact solution. We compare different old and new methods both theoretically and experimentally. The 2-step method also works for a more general class of constrained network optimization problems. We illustrate the generic approach using several examples. We have also developed a software package CNOP that provides this generic 2-step approach as well as all state of the art algorithms for constrained shortest paths.

Acknowledgments

The work on this thesis was carried out from 1998–2001 at the Max-Planck-Institut für Informatik in Prof. Dr. Kurt Mehlhorn's research group. I want to thank the Deutsche Forschungsgemeinschaft for granting me a scholarship as a member of the Graduiertenkolleg "Effizienz und Komplexität von Algorithmen und Rechenanlagen".
The working conditions at MPI have been exceptional. Excellent technical facilities made my life easy, generous travel support allowed me to meet researchers from all over the world at workshops and conferences, and the large number of researchers and guests in the institute created a stimulating research atmosphere.
Foremost, I want to thank my advisor Prof. Dr. Kurt Mehlhorn for giving me the opportunity to work in his excellent Algorithms and Complexity Group. He introduced me to the constrained shortest path problem and I have learned a great deal from him. His teaching and research style is really inspirational.
Thank you to Prof. Dr. Ulrich Lauther from Siemens AG for inviting me to an internal workshop on applied discrete optimization. We had interesting discussions on the constrained shortest path problem that were the start of a fruitful collaboration.
I would also like to thank Prof. Dr. Jörg-Rüdiger Sack from SCS Carleton for introducing me to the geodesic shortest path problem and working with me on the geometric version of the constrained shortest path problem.
The contributions of my friends and collegues have been numerous and invaluable. I enjoyed all the discussions of not only scientific content. The cycling and the running group also helped me to keep my mind fresh. Thank you Andreas, Christian, Ernst, Frank, Fritz, Marite, Michael, Nicola, Susan, the Stefans, Sven, Thomas, Uli, Volker and all the others.
Thanks to Andreas Crauser and Bobbye Pernice for proofreading this thesis and for their constructive comments. Stefan Funke and Michael Seel also had numerous suggestions that helped to improve the presentation.
Special thanks go to Andreas Crauser, Stefan Funke, Nicola Geismann, and Michael Seel for always being there during my academic and private rollercoaster ride.
A big thank you to Katrin Börchers for being my soul sister and adding another dimension to my life.
Last, but certainly not least, I want to thank my family for their endless love and support.

Mark Ziegelmann
Saarbrücken, July 2001

Contents

Chapter 1

Introduction

The shortest path problem is one of the fundamental problems in computer science. It has been well studied and efficient polynomial time algorithms are known. Shortest path problems frequently arise in practice since in a variety of application settings we wish to send some material (*e.g.*, a data packet, a telephone call, or a vehicle) between two specified points in a network as quickly, as cheaply, or as reliably as possible.

However, in a practical setting we are often not only interested in a cheapest path or a quickest path but rather in a combination of different criterias, *e.g.* we want to have a path that is both cheap and quick. This is known as the bi- or multicriteria shortest path problem. Since optimizing over all criteria at once is not possible we choose one criteria as the cost function that we want to minimize, the others as resource functions and impose resource (or budget) limits on the maximal resource consumption of a path. The *constrained shortest path problem* is to find a minimum cost path between two nodes whose resource consumptions satisfy the resource limits.

Applications

The constrained shortest path problem has a large number of practical applications:

The first application that comes to mind is *route planning* in traffic networks. We want to go from A to B and for example want to minimize the possibility of traffic congestion while imposing a length constraint on the path (see Figure 1.1). Alternatively, we want to go from A to B as fast as possible but have budget constraints on fuel consumption and road fees.

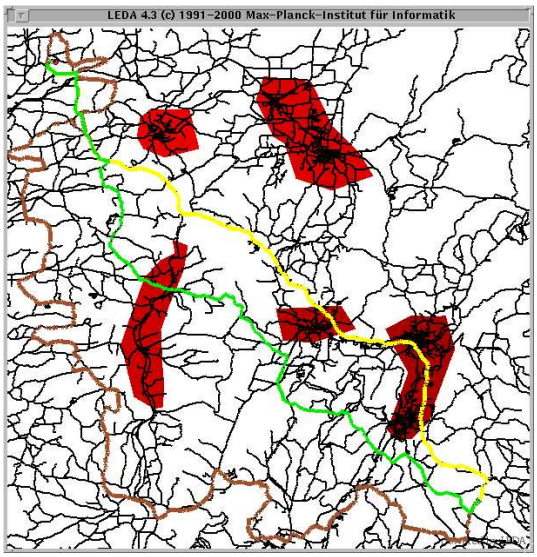

Figure 1.1: Minimum congestion path satisfying length constraint (areas of congestion are shaded). The minimum cost path is brown, the minimum resource path yellow and the constrained shortest path green.

In communication networks, we have the *quality of service (QoS)* routing problem. We are searching for a path of minimum costs that obeys given delay or reliability bounds. This is again a constrained shortest path problem. Further information can be found in the papers of Orda (1998) and Xue (2000).

There are also other applications that can be modeled as constrained shortest path problems:

Elimam and Kohler (1997) show how to model two engineering applications as (multiple resource) constrained shortest path problems: optimal sequences for the treatment of wastewater processes and minimum cost energy-efficient composite wall and roof structures.

Dahl and Realfsen (2000) and Nygaard (2000) studied the *linear curve approximation* problem and showed how to model this as a constrained shortest path problem (see Section 3.5). They find a path that corresponds to a minimal error approximation using a limited number of breakpoints (see Figure 1.2). Applications are data compression in areas like cartography, computer graphics, and image processing. In the area of traffic and communication networks this is known as routing problem with *hop constraints*, *i.e.*, a limit on the number of path links.

There are several applications in operations research that involve constrained shortest paths as a subproblem in column generation methods:

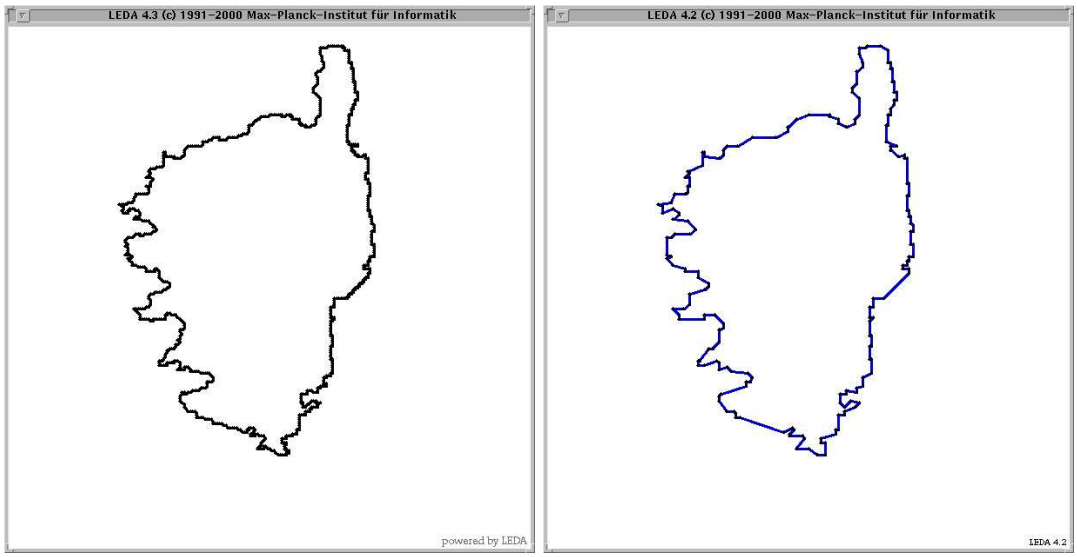

Figure 1.2: Coastline of Corsica (800 points) and minimum error approximation using only 200 points

Borndörfer and Löbel (2001) consider the *duty scheduling* problem and propose an adaptive column generation algorithm that needs to solve multiple resource constrained shortest paths as a subproblem.
Jahn, Möhring, and Schulz (1999) study a *route guidance* problem. They consider the optimal routing of traffic flows with length restrictions in road networks with congestion, and also propose a column generation method with constrained shortest paths as subproblem.
A similar problem in the QoS area was investigated by Holmberg and Yuan (1997) who also need constrained shortest paths as a subproblem in a column generation approach.
Lübbecke and Zimmermann (2000) proposed a column generation method for the *scheduling of switching engines* that also needs to solve constrained shortest paths as subproblem.

We see that the constrained shortest path problem is of immense practical interest in different areas of operations research.

Unfortunately, the introduction of even a single resource constraint turns the problem into a *hard* problem where we do not know a polynomial time algorithm to solve it. However, regarding its huge practical importance we would still like to solve the problem (or at least get an approximation) as efficiently as possible.

In this thesis we study the constrained shortest path problem both theoretically and experimentally. We also consider related problems like constrained minimum span-

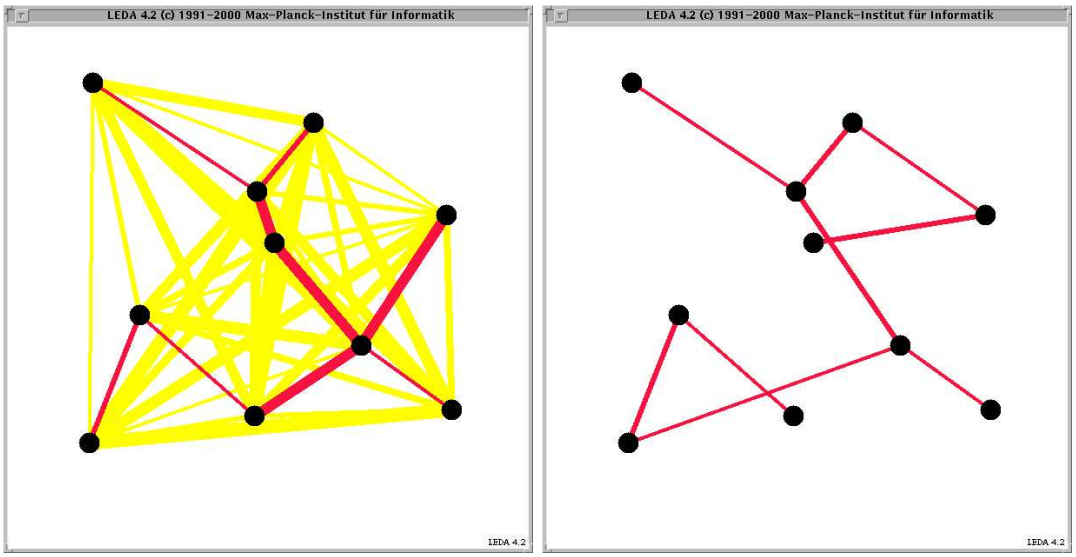

Figure 1.3: Minimum cost spanning tree and Minimum cost reliability constrained spanning tree. Width of edges corresponds to fault probability.

ning trees where we want to find a spanning tree of minimal cost while its resource consumptions satisfy the resource limits (see Figure 1.3).

Our Contribution

There is a variety of work on constrained shortest paths coming from different communities like operations research, algorithms, communication networks, and even signal processing. Almost all papers come up with essentially the same algorithm solving a relaxation of the problem for the single resource case. They only differ in the presentation of the method. Some derive it from geometric intuition, others adopt the Lagrangean relaxation viewpoint. Starting from a new ILP formulation of the problem, we will combine geometric intuition and linear programming theory to obtain a unified understanding of the method. Using this combined view, we are the first to prove a tight polynomial runtime bound for this method. We will show that the relaxation can be solved with $O(\log(nRC))$ parametric shortest path computations, where n is the number of nodes in the network and C and R denote the maximal cost and resource consumption of an edge, respectively. Our reformulation also allows us to extend the method to the multiple resource case, which has been an open problem up to now.

Solving the relaxation gives us upper and lower bounds for our problem. Previous papers suggested different gap closing steps to obtain a 2-step method for constrained

shortest paths. We again give a geometric intuition of the gap closing step and propose a special labeling approach to close the gap.

Then we experimentally compare all state of the art methods for constrained shortest paths on different benchmarks. This is the first detailed experimental runtime comparison for constrained shortest paths.

We then show that the 2-step method can be generalized to a broader class of constrained network optimization problems. All we need is a function returning the unconstrained optimum and a function ranking solutions. The single resource runtime bound for the relaxation extends. We illustrate the generic method using three examples: constrained minimum spanning trees, table layout, and constrained geodesic shortest paths.

We have developed a software package CNOP that implements the generic 2-step approach. A user only has to specify a function solving the corresponding unconstrained problem and a function ranking problem solutions. Additionally, CNOP offers all state of the art methods for constrained shortest paths and can be used as a testbed to see which approach is most suited for a special application. While several implementations of different methods exist, this is the first package that makes all state of the art methods publically available. It also offers the first publicly available implementation for the constrained minimum spanning tree problem. The flexibility of the CNOP package allows the user to experiment with other bi- or multicriteria network optimization problems.

We presented our results at the 8th Annual European Symposium on Algorithms 2000 in Saarbrücken and the 3rd Workshop on Algorithm Engineering and Experiments 2001 in Washington, DC.

Outline

We first review some basic notation, central algorithms and techniques in Chapter 2. Chapter 3 deals with the constrained shortest path problem. We propose a new relaxation formulation that allows us to derive simple combinatorial algorithms to solve the relaxation in the single and the multiple resource case. We discuss gap closing methods to obtain a 2-step approach and finally evaluate the different methods experimentally. In Chapter 4 we extend our 2-step approach to the more general class of constrained network optimization problems and discuss three examples: constrained minimum spanning trees, the table layout problem, and constrained geodesic shortest paths. In Chapter 5 we present our software package CNOP that implements the generic 2-step approach. Finally, we close with a discussion of our results and open problems in Chapter 6.

Chapter 2

Preliminaries

In this chapter we give a short and succinct overview of all the mathematical tools, terminology, and basic results we are going to use. The reader familiar with the notations introduced here may skip this chapter and refer to it when necessary.

2.1 Basic Graph Theory and Network Algorithms

In this section we give several basic definitions from graph theory and review the ideas of some important network algorithms. Further details can be found in Ahuja, Magnanti, and Orlin (1993).

2.1.1 Definitions

A *directed graph* $G = (V, E)$ consists of a set V of nodes and a set E of edges whose elements are ordered pairs of distinct nodes. We denote an edge e from node u to node v by $e = (u, v)$. A *directed network* is a directed graph whose edges and/or nodes have associated numerical values (*e.g.*, costs, capacities, etc).

We define an *undirected graph* in the same manner as we define a directed graph except that edges are unordered pairs of distinct nodes. We write $e = \{u, v\}$ for an undirected edge connecting nodes u and v. In undirected graphs, an edge can be viewed as a "two-way" street, whereas in directed graphs, an edge is only "one-way". An *undirected network* is an undirected graph whose edges and/or nodes have associated numerical

values.

Throughout this thesis we often make no distinction between graphs and networks, so we use the terms synonymously. We also often speak of graphs without specifying whether the graph is directed or not. In that case, the meaning can be derived from the context or is simply valid for both types.

We let $|V| = n$ denote the number of nodes and $|E| = m$ the number of edges in a graph G. For an edge $e = (u, v)$ we call u the *source* and v the *target* of the edge. Both u and v are called *endpoints* of the edge. Two edges are called *adjacent* if they share a common endpoint. The *degree* of a node is the sum of the number of its incoming edges and the number of its outgoing edges.

A graph $G' = (V', E')$ is a *subgraph* of $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$. It is a *spanning subgraph* if $V' = V$ and $E' \subseteq E$.

A *path* between two nodes u and v in a graph G is an alternating tuple $(u, (u, w_1), w_1, \ldots, w_r, (w_r, v), v)$ of nodes of V and edges of E starting in u and ending in v so that every node is adjacent to its neighbouring edges. If all nodes are pairwise distinct we have a *simple* or *loopless* path.

A *cycle* is a simple path $(i_1, (i_1, i_2), i_2, \ldots, (i_{r-1}, i_r), i_r)$ together with the edge (i_r, i_1). A graph is called *acyclic* if it contains no cycle.

We will say that two nodes i and j are *connected* if the graph contains at least one path from node i to node j. A graph is *connected* if every pair of nodes is connected.

A *tree* is a connected graph that contains no cycle. A tree T is a *spanning tree* of G if T is a spanning subgraph of G.

A *cut* is a partition of the node set V into two parts S and $\bar{S} = V - S$. Each cut defines a set of edges consisting of those edges that have one endpoint in S and the other in $\bar{S}$. An *s-t-cut* is defined with respect to two distinguished nodes s and t and is a cut $[S, \bar{S}]$ satisfying the property that $s \in S$ and $t \in \bar{S}$.

Paths, spanning trees and cuts will play an important role in the following chapters. We now briefly review network algorithms for these structures.

2.1.2 Shortest Paths

In the *shortest path problem (SP)* in a network $G = (V, E)$ with cost function c defined on the edges, we want to compute the minimum cost (or shortest) path from a node s to all other nodes. Shortest paths have the following properties:

Property 1: If the path $(s = i_1, (i_1, i_2), i_2, \ldots, (i_{h-1}, i_h), i_h = k)$ is a shortest path

from node s to node k, then for every $q = 2, 3, \ldots, h-1$, the subpath $(s = i_1, (i_1, i_2), i_2, \ldots, (i_{q-1}, i_q), i_q)$ is a shortest path from node s to node i_q.

Property 2: Let $d(j)$ denote the shortest path distance from node s to node j. Then a directed path P from s to k is a shortest path if and only if $d(j) = d(i) + c_{ij}$ for every edge $(i, j) \in P$.

There are two general approaches for solving the shortest path problem efficiently: label-setting and label-correcting methods.

Label-Setting Algorithms: The label-setting algorithms assign tentative distance labels to the nodes and then iteratively identify a true shortest path distance (a permanent label) to one or more nodes at each step. They work for non-negative cost functions.

algorithm *label-setting* :

$PERM = \emptyset$
$TENT = V$
$d(i) = \infty$ for each node $i \in V$
$d(s) = 0$
while $|PERM| < n$ **do** {

choose $i \in TENT$ with $d(i) = \min\{d(j) : j \in TENT\}$
$PERM = PERM \cup \{i\}$
$TENT = TENT \setminus \{i\}$
for_all_adj_edges (i, j) of i **do**
if $d(j) > d(i) + c_{ij}$ **then** $d(j) = d(i) + c_{ij}$

}

This label-setting algorithm is known as Dijkstra's algorithm. Using Fibonacci heaps, it can be shown to run in $O(n \log n + m)$ time.

Label-Correcting Algorithms: Label-correcting algorithms can also deal with negative edge lengths and make use of the shortest path optimality conditions:

Theorem 2.1.1 (Shortest Path Optimality Conditions): For every node $j \in V$, let $d(j)$ denote the length of some directed path from the source node s to node j.

Then the numbers $d(j)$ represent shortest path distances if and only if they satisfy the following optimality conditions:

$$d(j) \leq d(i) + c_{ij} \qquad \forall (i,j) \in E.$$

The label-correcting algorithms maintain a distance label with each node and iteratively update these labels until the distance labels satisfy the optimality conditions.

algorithm *label-correcting* :

$d(i) = \infty$ for each node $i \in V$
$d(s) = 0$
$LIST = \{s\}$

while $LIST \neq \emptyset$ **do** {

remove node i from $LIST$
forall_adj_edges (i,j) of i **do**
if $d(j) > d(i) + c_{ij}$ {
$d(j) = d(i) + c_{ij}$
if $j \notin LIST$ **then** add j to $LIST$
}

}

Identifying an edge violating the optimality condition is the key point here. A FIFO implementation of LIST leads to a polynomial running time of $O(nm)$; a dequeue implementation as proposed by Pape (1974) can be exponential in the worst case but is very efficient in practice, especially on road networks.

More details can be found in Ahuja, Magnanti, and Orlin (1993).

2.1.3 Minimum Spanning Trees

The *minimum spanning tree problem (MST)* in a network $G = (V, E)$ with cost function c defined on the edges is to find a spanning tree of minimal cost in G. As for the shortest path case, optimality conditions play a central role in developing algorithms for the problem.

Theorem 2.1.2 (Cut Optimality Conditions): A spanning tree T is a minimum spanning tree if and only if it satisfies the following cut optimality conditions: For every tree edge $(i,j) \in T$, we have $c_{ij} \leq c_{kl}$ for every edge (k,l) that is contained in the cut formed by deleting edge (i,j) from T.

Theorem 2.1.3 (Path Optimality Conditions): A spanning tree T is a minimum spanning tree if and only if it satisfies the following path optimality conditions: For every nontree edge (k,l) of G, we have $c_{ij} \leq c_{kl}$ for every edge (k,l) that is contained in the path in T connecting nodes k and l.

There is a simple algorithm for MST using the path optimality conditions, known as Kruskal's algorithm:
We first sort all edges in nondecreasing cost order and define a set, LIST, that is the set of edges we have chosen as part of a minimum spanning tree. Initially, LIST is empty. We examine the edges in sorted order one by one and check whether adding the edge we are currently examining to LIST creates a cycle with the edges already in LIST. If it does not, we add the edge to LIST; otherwise we discard it. We terminate when $|\text{LIST}| = n - 1$. At termination, the edges in LIST constitute a minimum spanning tree. Kruskal's algorithm can be implemented in $O(m + n \log n)$ time, plus time for sorting the edges[1].

There is another simple algorithm for MST using the cut optimality conditions, known as Prim's algorithm:
This algorithm builds a spanning tree from scratch by fanning out from a single node and adding edges one at a time. It maintains a spanning tree on a subset S of nodes and adds a nearest neighbour to S. It does so by identifying an edge (i,j) of minimum cost in the cut $[S, \bar{S}]$. Prim's algorithm can be implemented in $O(m + n \log n)$ time using Fibonacci heaps.

For more details we again refer to Ahuja, Magnanti, and Orlin (1993).

2.1.4 Maximum Flows and Minimum Cuts

Maximum Flows: Given a directed graph $G = (V, E)$, a source node $s \in V$, a sink node $t \in V$, and a non-negative capacity cap_e for every edge of e, the goal is to find a maximum flow from s to t. For a node v, we use $In(v)$ and $Out(v)$ to denote the set of edges ending in and emanating from v, respectively, and for an edge e we use f_e to denote the flow along this edge.

$$
\begin{array}{lrcll}
\text{maximize} & \sum_{e \in Out(s)} f_e & & & \\
\text{subject to} & \sum_{e \in Out(v)} f_e & = & \sum_{e \in In(v)} f_e & \text{for every node } v \in V \setminus \{s,t\} \\
& f_e & \leq & cap_e & \text{for every edge } e \in E \\
& f_e & \geq & 0 &
\end{array}
$$

[1] We can even reach $O(m\alpha(n,m))$ using an improved union-find implementation (where $\alpha(n,m)$ is the inverse Ackermann function as defined in Cormen, Leiserson, and Rivest (1993)).

The first family of constraints states that we have flow conservation at all nodes distinct from s and t and the other constraints state that flows are non-negative and bounded by the edge capacities. The goal is to maximize the flow leaving node s subject to these constraints.

Minimum s-t-Cut: In the setting of the maximum flow problem we may also be interested in a minimum capacity cut C separating s and t. A set C of edges is called an s-t-cut if every path from s to t contains at least one edge in C. The capacity of C is the sum of the capacities of the edges in C. The minimum s-t-cut problem can be easily formulated as an ILP. We have a zero-one variable for every edge e with the intended semantics that $e \in C$ iff $y_e = 1$.

$$\begin{array}{lrcll} \text{minimize} & \sum_{e \in E} cap_e y_e & & & \\ \text{subject to} & \sum_{e \in P} y_e & \geq & 1 & \text{for every } s\text{-}t\text{-path } P \\ & y_e & \in & \{0,1\} & \text{for every edge } e \in E \end{array}$$

There is an important duality relationship between maximum flows and minimum cuts:

Theorem 2.1.4 (Max-Flow Min-Cut Theorem): The maximum value of the flow from a source node s to a target node t in a capacitated network equals the minimum capacity among all s-t cuts.

There is a preflow push algorithm that solves the maximum flow problem and hence minimum cut problems in $O(n^3)$ time. We again refer to Ahuja, Magnanti, and Orlin (1993) for a detailed explanation.

2.2 Basic Complexity Theory

When we analyze the performance of an algorithm in this thesis we assume the common RAM-model and use asymptotic analysis, the Big-Oh notation, to bound the worst case complexity (refer to Cormen, Leiserson, and Rivest (1993) for details).

2.2.1 $\mathcal{NP}$-completeness

Complexity theory allows us to classify a problem into two broad classes:

1. Easy problems that can be solved by polynomial-time algorithms,
2. Hard problems that are not likely to be solved in polynomial time and for which all known algorithms require exponential running time.

Complexity theory requires the problems to be stated in such a way that we can answer them with yes or no; we refer to this yes-no version of a problem as the *recognition version.*

Now we introduce the complexity classes:
We say that a recognition problem R belongs to the complexity class $\mathcal{P}$ if some polynomial-time algorithm solves problem R. The recognition versions of the shortest path problem or the minimum spanning tree problem belong to the class $\mathcal{P}$, for example.

We say that a recognition problem R is in the complexity class $\mathcal{NP}$, if for every yes instance I of R, there is a short (polynomial-length) verification that the instance is a yes instance. Note that it is much easier to verify that, for example, a given assignment satisfies a boolean formula, than deciding that there is no satisfying assignment.

A recognition problem R is said to be *$\mathcal{NP}$-hard* if all other problems in the class $\mathcal{NP}$ polynomially transform to R.

A recognition problem R is said to be *$\mathcal{NP}$-complete* if (1) $R \in \mathcal{NP}$, and (2) R is *$\mathcal{NP}$-hard.*

Examples for $\mathcal{NP}$-complete problems are the well-known *Traveling Salesperson Problem*, the *Knapsack Problem*, and the *constrained shortest path problem* that we consider in this thesis. The book by Garey and Johnson (1979) is an excellent source for $\mathcal{NP}$-completeness results.

Sometimes the running time of an algorithm is of the form $O(n^k C)$ where C is a number given in the problem instance (*e.g.*, the maximum edge weight). Such a running time is not polynomial since we would like to have $\log C$ instead of C. We call running times of that kind *pseudo-polynomial.*

If the input instance satisfies the similarity assumption, *i.e.*, $C = O(n^k)$, for some fixed integer k, the running time becomes polynomial. If a problem is $\mathcal{NP}$-complete even when the similarity assumption is satisfied, we say that the problem is *strongly $\mathcal{NP}$-complete.* A problem that is $\mathcal{NP}$-complete but fails to stay $\mathcal{NP}$-complete if the similarity assumption is satisfied is called *weakly $\mathcal{NP}$-complete.*

We will see that the constrained shortest path problem belongs to the weakly $\mathcal{NP}$-complete problems, whereas the Traveling Salesperson Problem is strongly $\mathcal{NP}$-complete.

2.2.2 Approximation Schemes

Since many optimization problems are "hard nuts", *i.e.*, we do not know polynomial-time algorithms for them, we are often interested in algorithms that provide a feasible, but suboptimal solution in polynomial time, together with a provable guarantee regarding its degree of suboptimality.

We call an algorithm A an *ϵ-approximation* algorithm for a minimization problem with optimal cost Z^*, if for each instance of the problem, algorithm A runs in polynomial time and returns a solution with cost Z_A, so that

$$Z_A \leq (1+\epsilon)Z^*.$$

Symmetrically, for a maximization problem we require

$$Z_A \geq (1+\epsilon)Z^*.$$

We say that we have a *polynomial time approximation scheme (PTAS)*, if for every fixed $\epsilon > 0$, there exists an ϵ-approximation algorithm.

A polynomial time approximation scheme is rather powerful from a theoretical viewpoint, given that we do not know an exact polynomial-time algorithm for the problem, as it allows an arbitrarily close approximation the problem.

One of the most commonly used techniques for deriving PTAS for weakly $\mathcal{NP}$-hard problems is *rounding and scaling*. We will see examples of this technique later.

2.3 Linear Optimization

In linear optimization we are interested in optimizing a linear function subject to a set of linear constraints. Many interesting problems fall into this category and can be modeled as linear optimization problems: production planning, scheduling problems, network flow problems, and many more.

We give a short overview of the mathematical background, main results and solution techniques for linear optimization. This overview is based on Chvátal (1983), Schrijver (1986), and Bertsimas and Tsitsiklis (1997). These books can also be used for further reading.

2.3.1 Linear Programming

In the *Linear Programming Problem* we are given a system $Ax \leq b$ of linear inequalities and a linear objective function $f(x) = c^T x$. The goal is to find a *feasible* solution $\bar{x}$

(which means $A\bar{x} \leq b$) that maximizes (or minimizes) the objective function. Given a matrix $A \in \mathbb{R}^{m \times n}$, a vector $b \in \mathbb{R}^m$ and a vector $c \in \mathbb{R}^n$, the corresponding *linear program (LP)* is denoted by

$$\begin{aligned} \max \quad & c^T x \\ \text{s.t.} \quad & Ax \leq b \end{aligned}$$

or abbreviated as

$$\max\{c^T x \mid Ax \leq b\}.$$

Problems of a similar form can be easily converted into this standard form. A feasible solution x^* is called an *optimal solution* if $c^T x^* \geq c^T \bar{x}$ for all feasible solutions $\bar{x}$. If a linear program has no feasible solution, it is called *nonfeasible.*

The concept of *duality* is of fundamental importance in the field of linear programming. For every linear program

$$(P) : \max\{c^T x \mid Ax \leq b\},$$

another linear program, called the *dual problem* to (P), can be associated, which is defined as

$$(D) : \min\{y^T b \mid y^T A = c^T, y \geq 0\}.$$

The problem (P) is called the *primal problem.*

We now state several important results about connections between the primal and dual problem.

Lemma 2.3.1: The dual of the dual is the primal.

Lemma 2.3.2 (Weak duality): If $\bar{x}$ is primal feasible and $\bar{y}$ is dual feasible then $c^T \bar{x} \leq \bar{y}^T b$.

Theorem 2.3.1 (Strong duality): If a linear programming problem has an optimal solution, so does its dual, and the respective optimal costs are equal.

Theorem 2.3.2 (Complementary slackness): Let x and y be feasible solutions to the primal and the dual problem, respectively. The vectors x and y are optimal solutions for the two respective problems if and only if

$$\begin{aligned} y_i(A_i^T x - b_i) &= 0 \qquad \forall \text{ constraints } i, \\ (c_j - y^T A_j)x_j &= 0 \qquad \forall \text{ variables } j. \end{aligned}$$

Duality often enables us to form a different, sometimes simpler view of a problem. We will see an application of duality in the next chapter.

2.3.2 Polyhedra

Now we want to give the geometric interpretation of linear programming and need some polyhedral theory. A good source for further information is Schrijver (1986).

A *polyhedron* $P \subseteq \mathbb{R}^n$ is the set of points defined by a finite number of inequalities. A bounded polyhedron is called a *polytope*.

Now we relate polytopes to convex hulls. A set C is called *convex* if $x, y \in C$ implies $\lambda x + (1-\lambda)y \in C$ for all $0 \leq \lambda \leq 1$. The *convex hull* of a set of points S is the smallest convex set containing S.

Polytopes are precisely those sets in $\mathbb{R}^n$ that are the convex hull of finitely many points in $\mathbb{R}^n$. If $a \in \mathbb{R}^n \setminus \{0\}$ and $a_0 \in \mathbb{R}$, then the polyhedron $\{x \in \mathbb{R} \mid a^T x \leq a_0\}$ is called a *halfspace*. Every polyhedron is the intersection of finitely many halfspaces.

An inequality $a^T x \leq a_0$ is called *valid* with respect to a polyhedron P if $P \subseteq \{x \in \mathbb{R} \mid a^T x \leq a_0\}$. A set $F \subseteq P$ is called a *facet* of P, if there exists a valid inequality $a^T x \leq a_0$ of P so that $F = \{x \in P \mid a^T x = a_0\}$. In this case we say that F is the face of P defined (respectively induced) by the valid inequality $a^T x \leq a_0$. Note that every face of a polytope is itself a lower-dimensional polytope.

Especially important faces are those of minimum and maximum dimension. The 0-dimensional faces of P are called *vertices* of P and the faces of dimension $dim(P) - 1$ are called *facets* of the polytope.

In this thesis we will restrict ourselves to *rational polytopes* P, *i.e.*, $P \subseteq \mathbb{Q}^n$. A rational polytope is called *integral* if all its vertices are integral. The applicability of polyhedral methods in combinatorial optimization often comes down to the ability to prove that polyhedra are integral.

Proving integrality of polyhedra is a very difficult task. We will state two of the most important results giving us conditions for integrality.

Theorem 2.3.3 (Hoffmann 1974): A rational polytope P is integral if and only if for all integral vectors w the optimal value of $\max\{w^T x \mid x \in P\}$ is an integer.

Another sufficient condition is the *total unimodularity* of the constraint matrix. A matrix A is called *totally unimodular* if each subdeterminant of A is $-1, 0$, or 1.

Theorem 2.3.4 (Schrijver 1986): Let A be a totally unimodular matrix and let B be an integral vector. Then the polyhedron $P = \{x \mid Ax \leq b\}$ is integral.

2.3.3 Solving Linear Programs

We briefly sketch two well-known methods for solving linear programs: The *simplex method* which is the most commonly used method in practice due to its efficiency, although it is not polynomial in the worst case, and the *ellipsoid method* that was the first method guaranteeing polynomial running time but lacking efficient implementations. A detailed description of these algorithms can be found in Chvátal (1983), Schrijver (1986) and Bertsimas and Tsitsiklis (1997).
We do not present interior point methods which combine practical efficiency and polynomial running time. An overview of these methods can be found in Bertsimas and Tsitsiklis (1997).

The Simplex Method

The simplex method requires an LP in the following form

$$\begin{aligned} \max \quad & c^T x \\ \text{s.t.} \quad & Ax = b \\ & x \geq 0 \end{aligned}$$

for an $m \times n$ matrix A and a m-vector b. Let $B = \{B_1, \ldots, B_m\}$ be a set of column indices and $N = \{1, \ldots, n\} \backslash B$. Let A_B be the $m \times m$-matrix consisting of columns given by B. A_B is called a *basis* of A if A_B is non-singular. The solution $x_B = A_b^{-1} b, x_N = 0$ is called the *basic solution* of $Ax = b$. We call a basic solution *primal feasible*, if $A_B^{-1} b \geq 0$.

In the following lemma we characterize the extreme points of a polyhedron by bases of the matrix A.

Lemma 2.3.3: Let x be an extreme point of a polyhedron $P = \{x \mid Ax = b, x \geq 0\}$, for a matrix A with full rank. Then there exists at least one basis A_B of A, so that $x = A_B^{-1} x_B$. If x is a primal feasible solution, then x is an extreme point.

Geometrically speaking, the simplex method starts with a feasible extreme point $x^{(0)}$. It then iteratively computes a feasible extreme point $x^{(i+1)}$ that provides a better objective function value. The algebraic counterpart to extreme points are the bases of the matrix A. If the current solution is not degenerate, *i.e.*, has a unique basis, there is a simple calculation that provides the information whether the addition of an index in the current basis leads to a basis with a better optimal solution. If there is no such variable then the current solution is optimal. A further calculation gives the indices that we can remove from the basis to preserve primal feasibility. If there is no

such variable, the problem is unbounded (see Chvátal (1983) for details of the so-called tableau method).

Geometrically, an exchange of an index B_i of a basis B corresponds to a move along the line defined by $A_{B\setminus B_i} x_B = 0$ until a further constraint is met (the corresponding index enters the basis). Such a basis exchange is also called a *pivoting step.*
To avoid cycling in degenerate cases when an extreme point has several bases there are different pivoting strategies that are described in Chvátal (1983).

For the primal simplex method, we need a feasible basis to start with. Often, a feasible basis is known for a specific problem. If not, we have to solve the *auxiliary problem*, an LP for which a feasible basis is known, and whose optimal solution is feasible for the original problem. Details are again found in Chvátal (1983).

We have just described the *primal simplex method.* In every iteration, we have a primal feasible basis and we iterate until there is no basis update possible that improves the objective function value. There is also a *dual simplex method.* In every iteration we have a basis, so that there is no index whose addition could improve the objective function value, but it is not necessarily primal feasible, *i.e.*, there can be indices i with $x_i < 0$. In an iteration we move in a direction that increases the value of a variable whose value is negative, thus preserving the optimality of the new extreme point if it is primal feasible.

Although Klee and Minty gave an example that shows that the running time of the simplex method (regardless which pivoting rule is used) can be exponential (see also Chvátal (1983)), the practical performance is usually good and the simplex method is the method most commonly used in practice.

The Ellipsoid Method

The ellipsoid method of Khachian is a method that polynomially solves linear programs. The heart of its procedure solves the membership problem of finding a point in the interior of a polytope $P_< = \{x \mid Ax < b\}$ or deciding that $P_< = \emptyset$.

Starting with an initial ellipsoid enclosing $P_<$, this method iteratively computes ellipsoids that cover $P_<$. The volume of these ellipsoids decreases geometrically. In an iteration step we try to find a hyperplane h that separates the center of the current ellipsoid from the polytope, so we can construct a new enclosing ellipsoid with smaller volume. Eventually, the center of the new ellipsoid must lie in $P_<$ or the volume will be so small that we can conclude $P_< = \emptyset$. Khachian has shown that this can be done in polynomial time provided that a separating hyperplane can be found in polynomial time. Computing the optimal LP value from a point $x_t \in P_<$ can be done by cutting

off that point with a new constraint $c^T x < c^T x_t$ and reiterating the process until the new polytope contains no more feasible point. The interested reader should refer to Schrijver (1986) and Bertsimas and Tsitsiklis (1997) for more details.
The ellipsoid method is an important theoretical result but it is not competitive in practice.

An interesting fact concerns linear programs with an exponential (or even infinite) number of constraints that are impossible to be generated and stored implicitly. The running time of the Ellipsoid method does not depend on the number of constraints, the only problem dependent part of this method is deciding whether a point x is in the polyhedron and, if not, finding a hyperplane separating x from the polyhedron. If this *separation problem* can be solved in polynomial time then the Ellipsoid method is also polynomial.

If we want to solve LPs of that type in a practically efficient way, we use the so-called *cutting plane generation* method: Instead of considering all the constraints at once, we start off with a small subset of the constraints and solve the relaxed problem with the simplex method. Let x be the optimal solution. We now solve the separation problem for x. If x is in the original polyhedron then no violating constraint was found and x is indeed the optimal solution of the original problem. Otherwise we add a violating constraint to our constraint subset and iterate. The optimal basis of the old problem is dually feasible for the new problem, thus we can start the dual simplex method with this basis. One usually observes that the dual simplex needs only very few iterations to solve the new problem.

A surprising observation is that this method usually terminates after a rather small number of iterations. We will see an application of this method in the next chapter.

Applying the cutting plane generation method to the dual problem is identical to applying the *delayed column generation* method to the primal which is used when we have an exponential number of variables.

2.3.4 Integer Linear Programming

In discrete optimization problems, we seek to find a solution x^* in a discrete set X that optimizes an objective function $f(x)$ defined for all $x \in X$.

Discrete optimization problems often arise in a variety of contexts in science and engineering. A natural and systematic way to study a broad class of discrete optimization problems is to model them as integer linear programming problems.

The *integer linear programming problem*, or *integer linear program (ILP)* is defined as

$$\max\{c^T x \mid Ax \leq b, x \text{ integral}\}.$$

So integer linear programming is the same as linear programming except that (some) variables are restricted to take integer values. Solving ILPs to optimality is an $\mathcal{NP}$-complete problem (Garey and Johnson 1979) as opposed to LP solving that can be done in polynomial time using the Ellipsoid method.

In comparison to linear programming, integer linear programming is significantly richer in modeling power (*e.g.*, introduction of binary choice variables). Many network optimization problems can be elegantly modeled as ILPs as we will see in the following chapters.

There is a variety of methods to solve ILPs, such as cutting plane methods or branch and bound methods. We again refer to Bertsimas and Tsitsiklis (1997) for further reading.

2.3.5 Lagrangean Relaxation

To describe the general form of the Lagrangean relaxation technique, suppose that we consider the following integer programming problem

$$\begin{aligned} z^* = \min \quad & c^T x \\ \text{s.t.} \quad & Ax = b \\ & Cx = d \\ & x \text{ integer}, \end{aligned}$$

and assume that A, C, b, c, d have integer entries. Let $X = \{x \text{ integer} \mid Cx = d\}$. In order to motivate the method, we assume that optimizing over the set X can be done efficiently. However, adding the constraints $Ax = b$ to the problem makes the problem difficult to solve. The Lagrangean relaxation technique uses the idea of relaxing these complicating constraints by bringing them into the objective function with associated *Lagrange multipliers* μ. We refer to the resulting problem

$$\begin{aligned} z^* = \min \quad & c^T x + \mu^T (Ax - b) \\ \text{s.t.} \quad & x \in X \end{aligned}$$

as *Lagrangean subproblem* of the original problem and refer to the function

$$L(\mu) = \min\{c^T x + \mu^T (Ax - b) \mid x \in X\}$$

as the *Lagrangean function.*

Note that a solution of the Lagrangean subproblem does not need to be feasible for the original problem since we have eliminated the constraints $Ax = b$. The following lemma tells us what information we can get from a solution of the Lagrangean subproblem.

Lemma 2.3.4: For any vector μ of the Lagrangean multipliers, the value $L(\mu)$ of the Lagrangean function is a lower bound on the objective value z^* of the original problem.

As we have seen, for any value of the Lagrangean multiplier μ, $L(\mu)$ is a lower bound on the optimal objective function value of the original problem. To obtain the sharpest lower bound, we would need to solve the following optimization problem

$$L^* = \max_{\mu} L(\mu)$$

which we refer to as the *Lagrangean multiplier problem* or *Lagrangean dual problem* associated with the original problem.

It is clear from Lemma 2.3.4 that weak duality holds:

Theorem 2.3.5 (ILP duality): We have $L^* \leq z^*$.

We now state the general theorem that tells us that the optimum of the Lagrangean dual is equal to the integer relaxation of the ILP.

Theorem 2.3.6 (Lagrangean Relaxation): If we apply the Lagrangean relaxation technique to a linear programming problem (P) defined as $\min\{c^T x \mid Ax = b, Cx = d, x \geq 0\}$ by relaxing the constraints $Ax = b$, then the optimal value L^* of the Lagrangean dual equals the optimal objective function value of (P).

The preceding theorem tells us that Lagrangean relaxation provides an alternative method for solving a linear programming problem. In some situations the relaxed problem is easy to solve, but the original problem is not; in these situations, a Lagrangean-based algorithm is an attractive solution approach. We will see an example in the following chapter.

We now briefly review a standard procedure for solving the Lagrangean dual, the *subgradient procedure.*
Let us first give a geometric illustration of the Lagrangean dual. The function $L(\mu)$ is concave and piecewise linear, since it is the minimum of a finite collection of linear functions of μ (see Figure 2.1). As a consequence, the problem of computing L^* can be recast as a linear programming problem, but with a large number of constraints. If we could assume that the function $L(\mu)$ is differentiable then the classical steepest ascent

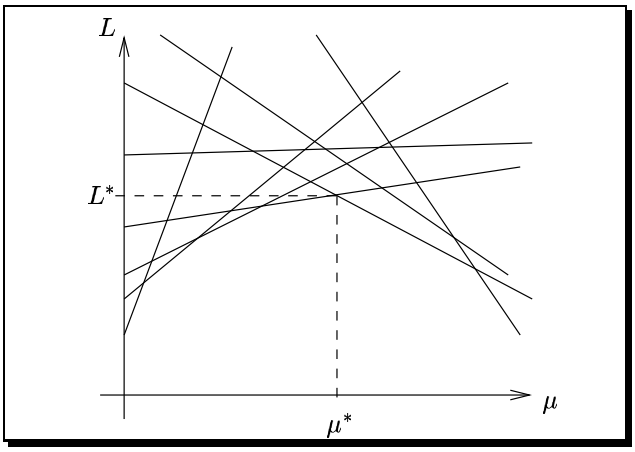

Figure 2.1: Illustration of Lagrangean Relaxation.

method for maximizing $L(\mu)$ is given by the sequence of iterations

$$\mu^{(i+1)} = \mu^{(i)} + \delta_i \nabla L(\mu^{(i)}), \quad i = 1, 2, \ldots$$

In our case, the function $L(\mu)$ is not differentiable and thus the gradient $\nabla L(\mu^{(i)})$ does not always exist. For this reason we need to extend the notion of the gradient to nondifferentiable concave functions.

Definition 2.3.1: Let f be a concave function. A vector s so that

$$f(x) \leq f(x^*) + s^T(x - x^*),$$

for all $x \in \mathbb{R}^n$, is called a *subgradient* of f at x^*. The set of all subgradients of f at x^* is denoted by $\partial f(x^*)$.

The following lemma establishes a necessary and sufficient condition for the maximum of a concave function.

Lemma 2.3.5: Let $f : \mathbb{R}^n \to \mathbb{R}$ be a concave function. A vector x^* maximizes f over $\mathbb{R}^n$ if and only if $0 \in \partial f(x^*)$.

The following *subgradient optimization algorithm* generalizes the steepest ascent algorithm and can be used to maximize a nondifferentiable concave function such as the Lagrangean function $L(\mu)$.

1. Choose a starting vector $\mu^{(i)}$.
2. Given $\mu^{(i)}$ choose a subgradient $s^{(i)}$ of the function $L(\cdot)$ at $\mu^{(i)}$. If $s^{(i)} = 0$ then $\mu^{(i)}$ is optimal and the algorithm terminates, otherwise continue.

3. Let $\mu^{(i+1)} = \mu^{(i)} + \delta_i s^{(i)}$, where δ_i is a positive stepsize parameter. Increment i and go to step 2.

Typically, only the extreme subgradients $s^{(i)}$ are used. A popular stepsize is given by

$$\delta_i = \frac{\hat{L}_D - L(\mu^{(i)})}{||s^{(i)}||^2}\alpha^{(i)}$$

for $0 < \alpha^{(i)} < 1$ and $\hat{L}_D$ being an estimate of the optimal value.

In practice, the stopping criterion $0 \in \partial L(\mu^{(i)})$ is rarely met. Typically, the algorithm is stopped after a fixed number of iterations. In some cases when the relaxed constraints are in the form $Ax \leq b$ instead of $Ax = b$, we also have to take care that $\mu^{(i)} \geq 0$. This adds to the disadvantage of not being able to guarantee monotonic convergence. In order to find a near-optimal solution, several iterations are necessary. Usually, after ten iterations the stepsize is too small and the algorithm is loses the ability to make rapid progress. On the other hand, this is a very simple procedure that is applicable to a wide range of problems and often is the only way to obtain reasonably good bounds efficiently. The selection of step sizes also leaves room for finetuning for special applications. We will see an application of the subgradient method in the next chapter.

Chapter 3

The Constrained Shortest Path Problem

In this chapter we consider a variant of the classical shortest path problem, the (resource) constrained shortest path problem (CSP). Unlike the original shortest path problem, CSP is $\mathcal{NP}$-complete. Since there are important applications that can be modeled by CSP, we are interested in solving the problem as efficiently as possible.

We start the chapter with a discussion of previous work. As with many difficult problems, we will first look at a simpler one, an LP relaxation of CSP. Starting from a new ILP formulation, we derive a combinatorial method for solving the relaxation by combining simple geometric intuition with optimization theory. In the single resource case our approach is equivalent to previously proposed methods, however, we are the first to prove a tight polynomial bound on the running time. We also obtain the first combinatorial method to solve the relaxation exactly in the multiple resource case. Solving the relaxation gives us upper and lower bounds. We will then review old methods and present new methods for closing the possible duality gap to obtain an exact 2-step method for CSP. We close the chapter with a detailed experimental comparison of the different methods.

3.1 Problem Definition

The *(resource) constrained shortest path problem (CSP)* asks for the computation of a least cost path obeying a set of resource constraints. More precisely, we are given a graph $G = (V, E)$ with $|V| = n$ and $|E| = m$, a source node s and a target node t, and

k resource limits $\lambda^{(1)}$ to $\lambda^{(k)}$. Each edge e has a cost c_e and uses $r_e^{(i)}$ units of resource i, $1 \leq i \leq k$. Costs and resources are assumed to be nonnegative. They are additive along paths. The goal is to find a least cost path from s to t that satisfies the resource constraints.

The special case $k = 1$ is called the *single resource case* which has previously been studied more extensively (see Section 3.2) and we will consider this case in Section 3.3.2. The *multiple resource case* for $k > 1$ will be discussed in Section 3.3.4.

3.2 Previous Work

We give an overview of the previous work on the constrained shortest path problem.

3.2.1 Complexity

Whereas the (unconstrained) shortest path problem can be efficiently solved in polynomial time, we will see that the introduction of a single additional resource constraint makes the problem $\mathcal{NP}$-complete.

CSP is listed as problem `ND30` (shortest weight-constrained path) in Garey and Johnson (1979) and reported to be $\mathcal{NP}$-complete by a transformation from PARTITION. Handler and Zang (1980) give an interesting connection to the well known *knapsack problem* that has a very similar flavour:

We are given a set of $n-1$ items, each having a value v_j and a weight w_j, for $j = 1, \ldots, n-1$. The goal is to pack items into a knapsack so that the weight limit λ is not exceeded and so that the value of the chosen items is maximized. The knapsack problem (KP) is formally given as follows:

$$\begin{aligned} \max \quad & \sum_{j=1}^{n-1} v_j x_j \\ \text{s.t.} \quad & \sum_{j=1}^{n-1} w_j x_j \leq \lambda \\ & x_j \in \{0,1\} \quad j = 1, \ldots, n-1 \end{aligned}$$

where v_j, w_j, λ are positive integers. Now we set up an n-node network with two parallel arcs from every node j to node $j+1$ for $j = 1, \ldots, n-1$. Let $c_{j,j+1}^{(1)} = M - v_j, r_{j,j+1}^{(1)} = w_j$ and $c_{j,j+1}^{(2)} = M, r_{j,j+1}^{(2)} = 0$ be the parameters for the first and second arcs for

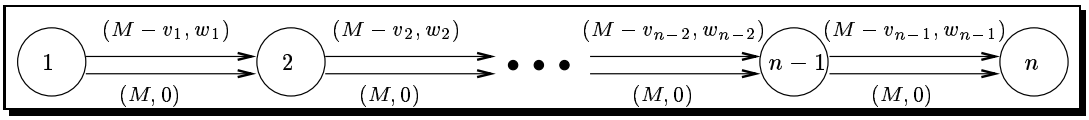

Figure 3.1: Network for the knapsack problem.

$j = 1, \ldots, n-1$, respectively, where $M = \max\{v_j : j = 1, \ldots, n-1\}$ (see Figure 3.1). Then it is evident that KP may be solved by finding a shortest path (with respect to parameter c) from node 1 to node n, subject to a resource constraint (with respect to parameter r), with right hand side λ. Since KP is $\mathcal{NP}$-complete it can be deduced that CSP is also $\mathcal{NP}$-complete.

3.2.2 Dynamic Programming Recursion

Early work dealing with CSP in the case of a single resource was done by Joksch (1966) who presented an algorithm based on dynamic programming (see also Lawler (1976)).

We call a path from node i to node j an r-path if the resource consumption of the i-j-path is less than or equal to r. Hence we are looking for a shortest[1] λ-path from 1 to n. Let $c_j(r)$ be the cost of the shortest r-path from node 1 to node j. Then the following recursive definition holds:

$$c_j(r) = \min\{c_j(r-1), \min_{(i,j)\in E, r_{ij}\leq r} \{c_i(r - r_{ij}) + c_{ij}\}.$$

Setting $c_1(r) = 0$ for $0 \leq r \leq \lambda$ and $c_j(0) = \infty$ for all $j = 2, \ldots, n$, we may recursively compute the optimum of CSP which is given by $c_n(\lambda)$.

The running time of this method is $O(m\lambda)$ and the space consumption is $O(n\lambda)$. Hence dynamic programming gives us a pseudopolynomial algorithm for CSP, as for the related Knapsack problem.

As in the case of the Knapsack problem, the dynamic programming approach can also be used to obtain a PTAS for CSP by rounding and scaling, which is discussed the next subsection.

The labeling approaches discussed in Section 3.2.4 build on this dynamic programming recursion making use of the fact that $c_j(r)$ is a step function.

[1] minimum cost

3.2.3 ϵ-Approximations

CSP is weakly $\mathcal{NP}$-complete since we have just seen a simple pseudo-polynomial dynamic programming formulation. Hassin (1992) has applied the standard technique of *rounding and scaling* to obtain a fully polynomial ϵ-approximation scheme (PTAS) for CSP. We briefly review his method now.

In the previous subsection we have seen a simple dynamic programming procedure. For our purposes another recursive algorithm is more suitable:
Let $g_j(c)$ denote the resource consumption of a minimum cost 1-j-path whose cost is at most c. Then the following recursive definition holds:

$$g_j(c) = \min\{g_j(c-1), \min_{(i,j)\in E, c_{ij}\leq c}\{g_i(c-c_{ij}) + r_{ij}\}\}.$$

Setting $g_1(c) = 0$ for $c = 0, \ldots, OPT$ and $g_j(0) = \infty$ for $j = 2, \ldots, n$, we may iteratively compute the optimum.

Note that OPT is not known a priori, but it satisfies $OPT = \min\{c \mid g_n(c) \leq \lambda\}$. Thus, $g_j(c)$ is first computed for $c = 1$ and $j = 2, \ldots, n$, then for $c = 2$, and so on, until we have $g_n(c) \leq \lambda$ for the first time, meaning that OPT is set to this value of c. The complexity of this algorithm is $O(|E|OPT)$.

Now let V be a given value, and suppose we want to test whether $OPT \geq V$ or not. A polynomial procedure answering that query can be extended to a polynomial algorithm for finding the exact value of OPT by simply performing a binary search. As our problem is $\mathcal{NP}$-hard we have to be satisfied with a weaker test.

Let ϵ be fixed, $0 < \epsilon < 1$. We now explain a polynomial time ϵ-approximation test with the following properties: If it outputs a positive answer then definitely $OPT \geq V$. If it outputs a negative answer, then all we know is that $OPT < V(1+\epsilon)$.

The test rounds the edge costs c_{ij}, replacing them by

$$\lfloor \frac{c_{ij}}{V\epsilon/(n-1)} \rfloor \frac{V\epsilon}{n-1}.$$

This decreases all edge costs by at most $V\epsilon/(n-1)$, and all path costs by at most $V\epsilon$. Now the problem can be solved by applying the previous algorithm to a network with the scaled edge costs $\lfloor c_{ij}/(V\epsilon/(n-1)) \rfloor$. The values $g_j(c), j = 2, \ldots, n$, are first computed for $c = 1$, then for $c = 2, 3, \ldots$ until either $g_n(c) \leq \lambda$ for some $c = \hat{c} < (n-1)/\epsilon$, or $c \geq (n-1)/\epsilon$.

In the first case, a λ-path of cost at most

$$\frac{V\epsilon}{n-1}\hat{c} + V\epsilon < V(1+\epsilon)$$

has been found, *i.e.*, $OPT < V(1+\epsilon)$. In the second case, every λ-path has $c' \geq (n-1)/\epsilon$ or $c \geq V$, so that $OPT \geq V$. So the test performs as required.

The polynomial running time for fixed ϵ is explained as follows: Taking the integer part of a nonnegative number in the interval $\{0, \ldots, U\}$ can be done in $O(\log U)$ time using binary search. Thus, rounding the edge costs takes $O(|E| \log(n/\epsilon))$ time since we only scale edge costs of edges with cost less than V, *i.e.*, $(n-1)/\epsilon$. Next we perform $O(n\epsilon)$ iterations of the algorithm above which again gives a $O(|E| \log(n/\epsilon))$ running time. The latter is therefore also the complexity of the whole test procedure.

Now we use this test procedure to derive a PTAS based on rounding and scaling: To approximate OPT we first determine easily computable upper and lower bounds. An upper bound UB can be set to the sum of the $n-1$ largest edge costs or, alternatively, to the cost of the minimum resource path. A lower bound LB can be set to 0 or to the cost of the minimum cost path.

If $UB \leq (1+\epsilon)LB$ then UB is an ϵ-approximation to OPT. Suppose now that $UB > (1+\epsilon)LB$. Let V be a given value $LB < V < UB/(1+\epsilon)$. The test procedure can now be applied to improve the bounds on OPT. Specifically, either LB is increased to V or UB is decreased to $V(1+\epsilon)$. By performing a sequence of tests, the ratio UB/LB can be reduced. Once this ratio reduces to some predefined constant, say 2, then an ϵ-approximation can be obtained by applying the dynamic programming algorithm to the scaled edge costs $\lfloor c_{ij}/(LB\epsilon/(n-1)) \rfloor$. The error introduced is at most $\epsilon LB < \epsilon OPT$.

The running time for the last step is $O(|E|n/\epsilon)$. Reduction of the ratio UB/LB is best achieved by performing binary search on the interval (LB, UB) in a logarithmic scale. After each test we modify the bounds. To guarantee fast reduction of the ratio we execute the test with the value x so that $UB/x = x/LB$, that is $x = (UB \cdot LB)^{1/2}$. The number of tests required to reduce the ratio below 2 is therefore $O(\log\log(UB/LB))$ and each test takes $O(|E|n/\epsilon)$ time. Hassin (1992) shows how to compute an integer value near $(UB \cdot LB)^{1/2}$ in $O(\log\log(UB/LB))$ time. This gives us a total running time of $O(\log\log(UB/LB)(|E|(n/\epsilon) + \log\log(UB/LB)))$ for this ϵ-approximation algorithm and hence a PTAS for the constrained shortest path problem.

Hassin (1992) also gives an alternative PTAS whose complexity only depends on the number of input variables and $1/\epsilon$ and obtains a running time of $O(|E|(n^2/\epsilon)\log(n/\epsilon))$. The best PTAS was obtained by Phillips (1993) who reaches a running time of $O(|E|n/\epsilon + (n^2/\epsilon)\log(n/\epsilon))$.

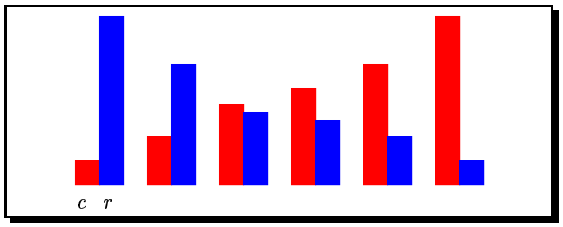

Figure 3.2: The ordered label list storing non-dominated promising subpaths.

3.2.4 Labeling Approaches

Labeling approaches can be seen as an improvement of the dynamic programming methods in the sense that they follow the line of *Pareto-optima* approaches and do not consider dominated paths. An v-w-path p is *dominated* by an v-w-path q if $c_p \geq c_q$ and $r_p \geq r_q$, *i.e.*, q is more efficient than p with respect to cost and resource consumption[2].

The standard labeling approaches as described in Aneja, Aggarwal, and Nair (1983), Desrochers and Soumis (1988), Desrosiers, Dumas, Solomon, and Soumis (1995), Lauther (2001), and Stroetmann (1997) use a set of labels for each node. Each label represents a path q from s to this node and consists of a tuple of numbers (c_q, r_q) representing cost and resource consumption of this path. Only non-dominated labels are stored in the list of each node in increasing cost order which implies decreasing resource order in the single resource case (see Figure 3.2). As Joksch (1966) has already observed, this list of non-dominated labels are the break points of a step-function and only these have to be considered to find the optimal solution.

A labeling algorithm now finds all non-dominated labels on every node. Starting with no labels at every node, except for label $(0, 0)$ at node s, the algorithm extends the label lists by extending non-dominated s-v-subpaths along their outgoing arcs. A path does not need to be extended and is called *non-promising* if its minimal resource completion exceeds the resource limit or if its minimal cost completion exceeds an upper bound on the solution.

Apart from this core algorithm, there are again label setting and label correcting variants. We will discuss these variants in more detail in Section 3.4.2.

No matter what strategy is used, the worst case complexity still does not improve on the $O(mR)$ bound of dynamic programming. We will see in Section 3.5 how the different variants perform in experiments.

[2] A multiple resource definition is analogous. Therefore, we concentrate on the single resource case.

Since CSP labeling methods follow the ideas of the standard shortest path labeling algorithms and since they allow easy incorporation of other constraints (*e.g.*, time windows on the nodes) they are often used in practice (see Jahn, Möhring, and Schulz (1999), Desrosiers, Dumas, Solomon, and Soumis (1995)).

3.2.5 Path Ranking

A straightforward but usually nonefficient strategy in combinatorial optimization problems is the enumeration of solutions.
Hence, CSP can be solved by ranking paths in non-decreasing cost order until we have found a path obeying the resource limit(s). This first feasible path then constitutes the optimal solution. The path ranking problem, often called k-shortest path problem, has been widely studied in the literature (Yen 1971; Katoh, Ibaraki, and Mine 1982; Skicism and Golden 1989; Miaou and Chin 1991; Azevedo, Madeira, and Martins 1993; Martins and Santos 1996; Eppstein 1999; Jimenez and Marzal 1999) (refer to Eppstein (1999) and Jimenez and Marzal (1999) for a more detailed discussion). The asymptotically best algorithm is due to Eppstein (1999) and runs in time $O(m + n \log n + kn)$. Jimenez and Marzal (1999) proposed a recursive enumeration algorithm that runs in $O(m + kn \log(m/n))$ but turned out to be more efficient than Eppstein's algorithm in their experimental comparison. We will briefly sketch their method now:

They use the following idea for the computation of the k-shortest paths[3] from node s to v: Let $P^k(v)$ denote the set of the k-shortest paths from s to v and $\pi^k(v)$ denote the k-shortest path from s to v. Each path in $P^k(v)$ reaches v from some node u incident to v. In order to compute $\pi^k(v)$, for every node u incident to v, we only need to consider the shortest path from s to u that does not lead to a path in $P^{k-1}(v)$. Thus, we can associate with v a set of candidate paths $C(v)$ among which $\pi^k(v)$ can be chosen, that contains at most one path from each node u incident to v.

Their recursive enumeration algorithm implements this idea:

1. Compute $\pi^1(v)$ for all $v \in V$ by any shortest path algorithm and set $k = 1$.

2. Repeat until $\pi^k(t)$ does not exist or no more paths are needed:

   (a) Set $k = k + 1$ and compute $\pi^k(t)$ by calling *NextPath*(t, k).

For $k > 1$, and once $\pi^1(v), \pi^2(v), \ldots, \pi^{k-1}(v)$ are available, *NextPath*(v, k) computes $\pi^k(v)$ as follows

[3]Not necessarily loopless paths.

1. If $k = 2$ then initialize a set of candidates to the next shortest path from s to v by setting $C(v) = \{\pi^1(u) \cdot v : (u, v) \in E \text{ and } \pi^1(v) \neq \pi^1(u) \cdot v\}$.

2. If $v = s$ and $k = 2$ go to step 6.

3. Let u and k' be the node and index, respectively, so that $\pi^{k-1}(v) = \pi^{k'}(u) \cdot v$.

4. If $\pi^{k'+1}(u)$ has not yet been computed, compute it by calling $NextPath(u, k'+1)$.

5. If $\pi^{k'+1}(u)$ exists, then insert $\pi^{k'+1}(u) \cdot v$ in $C(v)$.

6. If $C(v) \neq \emptyset$, then select and delete a path with minimum length from $C(v)$ and assign it to $\pi^k(v)$, else $\pi^k(v)$ does not exist.

Jimenez and Marzal (1999) store the k-shortest paths to a node v dynamically in a linked list, whereas each path $\pi^k(v) = \pi^j(u) \cdot v$ is compactly represented by its length and a back pointer to path $\pi^j(u)$. The set of candidates $C^k(v)$ is represented as a heap. Jimenez and Marzal (1999) show that their recursive enumeration algorithm runs in time $O(m + kn \log(m/n))$.

Since the number of paths to be ranked before finding the optimum may be exponential in the size of the graph, the path ranking method is known to perform badly in an experimental setting (Handler and Zang 1980; Skicism and Golden 1989) which we will confirm in our experimental section.

3.2.6 ILP Formulation

CSP can be formulated as a zero-one integer linear program as follows: We define 0-1 variables x_{ij} for edges $(i, j) \in E$:

$$x_{ij} = \begin{cases} 1 & \text{if edge } (i,j) \text{ belongs to an optimal path,} \\ 0 & \text{otherwise.} \end{cases}$$

The ILP for CSP is then given by

$$\min \quad \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij} \tag{3.1}$$

$$\text{s.t.} \quad \sum_{i \in V} \sum_{j \in V} r_{ij}^{(d)} x_{ij} \leq \lambda^{(d)} \qquad \forall d = 1, \ldots, k \tag{3.2}$$

$$\sum_{i \in V} x_{ij} = \sum_{i \in V} x_{ji} \quad \forall j \in V \setminus \{s, t\} \tag{3.3}$$

$$\sum_{j \in V} x_{sj} = 1 \tag{3.4}$$

$$\sum_{i \in V} x_{it} = 1 \tag{3.5}$$

$$x_{ij} \in \{0, 1\} \qquad \forall i, j \in V \tag{3.6}$$

Equations (3.2) ensure that the total resource consumptions of the path satisfy the resource limits. Equations (3.3), (3.4), and (3.5) are the standard shortest path constraints: Equation (3.3) is the degree constraint for each vertex of the graph (other than source and target) while equations (3.4) and (3.5) ensure that one edge leaves the source and one edge enters the target vertex.

Hence the ILP for CSP is the usual shortest path formulation with additional resource constraints. The ILP has $|E|$ variables and $|V| + k$ constraints.

3.2.7 Relaxation Methods

Since the resource constraint turns the well studied, efficiently solvable shortest path problem into an $\mathcal{NP}$-hard one, there has been a variety of work (for the single resource case) relaxing this resource constraint by turning it into the objective function and solving scaled cost shortest path problems to obtain bounds for the original problem.

The previously proposed methods essentially all follow this idea and only differ by a varying degree of theoretical underpinning or intuition. We will briefly review the different methods and present a unified, both theoretically sound and intuitive description of this idea together with a new running time analysis in the next section.

The Method of Aneja and Nair

Aneja and Nair (1978) were the first to propose a parametric approach for CSP. They examined the relation of CSP to bicriteria shortest paths (BCSP) and observed the importance of nondominated extreme paths. By turning the resource constraint in the objective function and introducing scaling parameters for costs and resources obtained

by the slope of the line through two extreme points, they essentially give the hull approach that we will introduce in the next section. They show that their algorithm needs at most n iterations of scaled costs shortest path computations if the nondominated extreme points are randomly distributed over the entire nondominated region. We will improve on this result in Section 3.3.3. Moreover, they erroneously claimed to have found the optimal solution of CSP with this procedure, thus ignoring the possible duality gap that was first pointed out by Pujari, Agarwal, and Gulati (1984).

The Method of Minoux and Ribeiro

Minoux and Ribeiro (1985, 1986) consider the *hard* constrained shortest path problem where we have upper and lower resource limits. As an application they show how to transform equality constrained knapsack problems to doubly constrained shortest path problems (Minoux and Ribero 1984).

Their approach relies on generating partial solutions (from s to v) by solving parametric shortest path computations and combining these with final solutions (from v to t) to obtain feasible solutions. They design a pseudopolynomial algorithm for solving the general parametric shortest path problem and extend a previous result by Karp and Orlin (1981) that was only valid for a special case.

Their approach gives approximate solutions for the doubly constrained shortest path problem with rather good quality as shown in their experiments with problems arising from equality constrained knapsack problems.

The Method of Handler and Zang

Handler and Zang (1980) were the first to propose a 2-step approach for single resource CSP. They first solve the Lagrangean Dual of the problem and then close the (potential) duality gap with a k-shortest path algorithm.

To derive a lower bound on the problem they relax the resource constraint in Lagrangean fashion, *i.e.*, they scale it with a Lagrange multiplier μ and turn it into the

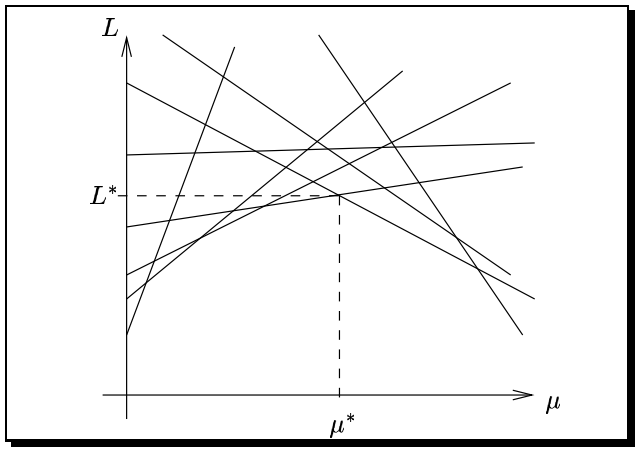

Figure 3.3: Illustration of Lagrangean Relaxation. Paths correspond to lines.

objective function.

$$L(\mu) = \min \quad \sum_{i \in V} \sum_{j \in V} (c_{ij} + \mu r_{ij}) x_{ij} - \mu\lambda \tag{3.7}$$

$$\sum_{i \in V} x_{ij} = \sum_{i \in V} x_{ji} \quad \forall j \in V \setminus \{s,t\} \tag{3.8}$$

$$\sum_{j \in V} x_{sj} = 1 \tag{3.9}$$

$$\sum_{i \in V} x_{it} = 1 \tag{3.10}$$

$$x_{ij} \in \{0,1\} \qquad \forall i,j \in V \tag{3.11}$$

The function $L(\mu)$ is just a scaled cost shortest path computation and thus can be efficiently solved for fixed nonnegative values of μ. Handler and Zang are interested in maximizing the lower bound, *i.e.*, in solving the Lagrangean dual

$$L^* = L(\mu^*) = \max_{\mu \geq 0} L(\mu).$$

They propose a combinatorial method to solve this Lagrangean Dual problem that is based on computing a successive approximation of the piecewise linear concave function $L(\mu)$ over $\mu \geq 0$. Since we can write $L(\mu) = L(\mu, x)$ as $L(\mu, x) = f(x) + ug(x)$ with $f(x) = \sum c_{ij}x_{ij}$ and $g(x) = \sum r_{ij}x_{ij} - \lambda$, each path can be interpreted as a line in L-μ-space (see Figure 3.3). The lower envelope of the lines corresponding to the so-far computed paths is interpreted as current approximation. In general they have two paths x^+ and x^- so that $g(x^+) > 0$ and $g(x^-) \leq 0$, and $f(x^-) \leq f(x^+)$. Then they set $\mu' = (f(x^-) - f(x^+))/(g(x^+) - g(x^-))$ as the new value for the multiplier and compute $L(\mu', x')$. If $L(\mu') = L$ or $g(x') = 0$ then $L(\mu')$ is the optimum of the Lagrangean dual.

Otherwise we set $x^+ = x'$ if $g(x') > 0$ or $x^- = x'$ if $g(x') < 0$. The procedure is iterated until we find the optimum.

Having solved the Lagrangean Dual, we have upper and lower bounds for CSP. If there is a gap, Handler and Zang propose closing this gap by path ranking with a k-shortest path algorithm. To make use of the dual relaxation solution they rank the paths with respect to the optimal reduced cost function and are thus able to continue "naturally" from the dual solution. In their experiments they show that this 2-phase method reduces the number of paths to be ranked considerably compared to path ranking with respect to the original cost function. We will support this claim in our experiments (see Section 3.5) and give an intuitive geometric explanation.

The relaxation method that we will propose in the next section follows this idea, in fact, in the single resource case our combinatorial approach is identical to their relaxation method except that we take a more intuitive primal viewpoint in the derivation of our method. Contrary to Handler and Zang, we are able to prove a tight polynomial bound on the number of iterations of our method and are also able to extend it to optimally solve the multiple resource relaxation.

The Method of Beasley and Christofides

Beasley and Christofides (1989) propose an algorithm for the multiple resource constrained shortest path problem. They first use a subgradient procedure to approximately solve the Lagrangean Dual and then close the possible duality gap with a tree search procedure after applying problem reduction techniques[4].

We have seen an ILP formulation for our problem in Section 3.2.6. To derive a lower bound on the problem they relax the resource constraints in Lagrangean fashion, *i.e.*, they scale them with Lagrangean multipliers μ_i $i = 1, \ldots, k$ and turn them into the

[4]Actually, they also consider costs and resources on nodes and lower bounds on the resource consumption but since the node consideration is a straightforward generalization and since their treatment of the lower bounds seems rather heuristical, we only describe the standard multiple resource formulation.

objective function.

$$\min \quad \sum_{i\in V}\sum_{j\in V}(c_{ij}+\sum_{l=1}^{k}\mu_l r_{ij}^{(l)})x_{ij}-\sum_{l=1}^{k}\mu_l\lambda^{(l)} \tag{3.12}$$

$$\sum_{i\in V}x_{ij}=\sum_{i\in V}x_{ji} \quad \forall j\in V\setminus\{s,t\} \tag{3.13}$$

$$\sum_{j\in V}x_{sj}=1 \tag{3.14}$$

$$\sum_{i\in V}x_{it}=1 \tag{3.15}$$

$$x_{ij}\in\{0,1\} \qquad \forall i,j\in V \tag{3.16}$$

For fixed nonnegative values of the multipliers μ_i this is just a usual shortest path problem with scaled costs and can be solved efficiently.

Beasley and Christofides now use a subgradient procedure to maximize the lower bound since k Lagrange multipliers make it complicated to obtain the exact solution[5].

We now briefly review their update of the Lagrangean multipliers μ_i. Let $\mu_i^{(g)}$ be the set of multipliers in step g of the subgradient procedure leading to a solution path $\bar{p}$ with cost $c_{\bar{p}}$ and resource consumption $r_{\bar{p}}^{(i)}$. Let UB and LB be the current upper bound and lower bound value given by path $\bar{p}$. The new Lagrangean multipliers for step $g+1$ are given as

$$\mu_i^{(g+1)}=\max\{0,\mu_i^{(g)}+T\cdot H_i\}$$

where $H_i=-\lambda^{(i)}+r_{\bar{p}}^{(i)}$ and the step size T is defined as

$$T=\frac{f(UB-LB)}{\sum_{i=1}^{k}((-r_{\bar{p}}^{(i)})^2+H_i^2)}.$$

This is an approximate procedure that depends on the parameters f and the initial upper bound. Beasley and Christofides use $f=0.25$, an initial upper bound $UB^{(0)}=50LB^{(0)}$, where $LB^{(0)}$ is the minimum cost of a path. The subgradient procedure is not guaranteed to converge against the optimum of the relaxation, hence it is sensible to stop the iteration when the change of the bounds (and the stepsize) falls beyond a certain limit. Beasley and Christofides abort the computation after 10 iterations. We will compare the solution quality and running time of their method with our new approach in our experimental Section 3.5.

After the subgradient procedure they perform problem reductions with the bounds obtained (see Section 3.4.1).

[5]We will show in Section 3.3.4 how this can be done.

If a duality gap exists, Beasley and Christofides close the gap using a binary depth first tree search procedure constructing a path from the source node and computing, at each tree node, a lower bound for the optimal completion of this path (backtracking when the upper bound is exceeded and performing further problem reductions when the upper bound is improved). This can be interpreted as a branch and bound method.

The Method of Xue

Xue (2000) proposes a primal-dual method for finding the optimal scaling parameter μ in the single resource case. He uses a mixture of binary search and hull approach and determines the running time to be $O(\log p)$ shortest path iterations where p is the number of nondominated[6] s-t-paths which is $O(n)$ since we might have $p = 2^n$ in the worst case.

Additionally, he observes that this method is also applicable for the constrained minimum spanning tree problem.

3.3 Solving the Relaxation

In this section we propose a simple combinatorial method for solving the relaxation of single resource CSP. We give a geometric and a simplex interpretation of our method and we will show that it only needs a logarithmic number of shortest path computations which is the first tight bound for this method. We then extend our approach to the multiple resource case to obtain the first combinatorial method computing the relaxation for multiple resource CSP exactly.

3.3.1 A Different ILP Formulation

We start with a (somewhat unusual) integer linear programming formulation. For any path p from s to t we introduce a 0-1 variable x_p and we use c_p and $r_p^{(i)}$ $(i = 1, \ldots, k)$

[6] minus equivalent ones

to denote the cost and resource consumption of p, respectively. Consider

$$\min \quad \sum_p c_p x_p \tag{3.17}$$

$$\text{s.t.} \quad \sum_p x_p = 1 \tag{3.18}$$

$$\sum_p r_p^{(i)} x_p \leq \lambda^{(i)} \quad i = 1, \ldots, k \tag{3.19}$$

$$x_p \in \{0, 1\} \tag{3.20}$$

This ILP models CSP. Observe that integrality together with (3.18) guarantees that x_p=1 for exactly one path, (3.19) ensures that this path is feasible, *i.e.* obeys the resource constraints, and (3.17) ensures that the cheapest feasible path is chosen.

We obtain an LP by dropping the integrality constraint. The objective value LB of the LP relaxation (P) is a lower bound for the objective value of CSP. The linear program has $k+1$ constraints and an exponential number of variables and hence is non-trivial to solve.

The dual of the LP relaxation (P) has an unconstrained variable u for equation (3.18) and k variables $v_i \leq 0$, $i = 1, \ldots, k$ for the inequalities (3.19):

The dual problem (D) can now be stated as follows:

$$\begin{aligned} \max \quad & u + \lambda^{(1)} v_1 + \cdots + \lambda^{(k)} v_k \\ \text{s.t.} \quad & u + r_p^{(1)} v_1 + \cdots r_p^{(k)} v_k \leq c_p \quad \forall p \\ & v_i \leq 0 \qquad i = 1, \ldots, k \end{aligned}$$

The dual relaxation has only $k+1$ variables albeit an exponential number of constraints.

Lemma 3.3.1: The dual relaxation can be solved in polynomial time by the Ellipsoid Method.

Proof. We show that the separation problem can be solved in polynomial time: Let $u^*, v_1^*, \ldots, v_k^*$ be the current values of the dual variables. The separation problem amounts to the question whether there is a path q so that

$$u^* + v_1^* r_q^{(1)} + \cdots + v_k^* r_q^{(k)} > c_q$$

or equivalently

$$c_q - v_1^* r_q^{(1)} - \cdots - v_k^* r_q^{(k)} < u^*.$$

This question can be solved with a shortest path computation with the scaled edge costs $\tilde{c}_e = c_e - v_1^* r_e^{(1)} - \cdots - v_k^* r_e^{(k)}$. Since costs and resources were assumed to be nonnegative

and since $v_i^* \leq 0$, it is guaranteed that the scaled edge costs are also nonnegative so that we may always compute a shortest path.

Hence we can identify violated constraints in polynomial time which completes the proof. □

Let us verify that the dual relaxation of our reformulation is indeed equivalent to the previously mentioned Lagrangean relaxation.

Lemma 3.3.2: The Lagrangean Relaxation and the dual relaxation (D) of the constrained shortest path problem have the same optimal value.

Proof. Strong duality (see Theorem 2.3.1) tells us that the optimal solutions of (D) and (P) coincide. It is also known (see Theorem 2.3.6) that the optimum of (P) is equal to the optimum of the Lagrangean relaxation of CSP. □

Now we turn to the simpler case of only one resource.

3.3.2 The Single Resource Case

Most of the previous work considered the single resource case which is also what we will do first. Only full understanding of the single resource case enables a generalization to the multiple resource case.

So let us consider the single resource relaxation. The dual now has two variables, u and $v \leq 0$:

$$\begin{aligned} \max \quad & u + \lambda v \\ \text{s.t.} \quad & u + r_p v \leq c_p \quad \forall p \\ & v \leq 0 \end{aligned}$$

The dual can be interpreted geometrically which gives us more insight into the LP and suggests a combinatorial method for solving it:

Standard Interpretation: A pair (u, v) is interpreted as a point in the u-v-plane. Each constraint is viewed as a halfspace in the u-v-plane and we search for the maximal feasible point in direction $(1, \lambda)$ (see Figure 3.4).

We propose a different geometric interpretation which gives even more intuition.

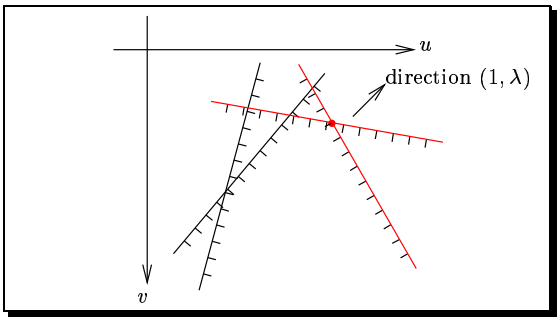

Figure 3.4: Find maximal point in direction $(1, \lambda)$ in the intersection of halfspaces.

(Geometric) Dual Interpretation: A pair (u, v) is interpreted as a line in r-c-space (the line $c = vr + u$). A constraint $u + r_p v \leq c_p$ is interpreted as a point (r_p, c_p) in r-c-space. Hence every path corresponds to a point in r-c-space. We are searching for a line with non-positive slope that maximizes $u + \lambda v$ while obeying the constraints, *i.e.* which has maximal c-value at $r = \lambda$, and which has all points (r_p, c_p) above or on it (see Figure 3.5), *i.e.* we are searching for the segment of the lower hull[7] which intersects the line $r = \lambda$.

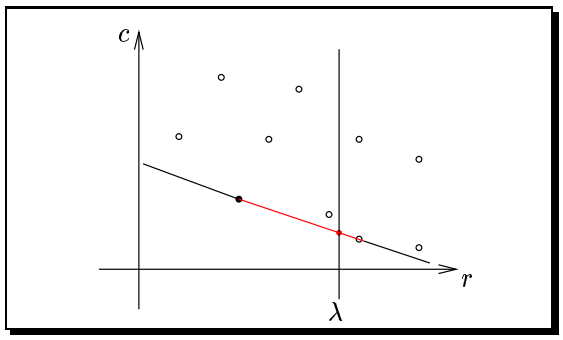

Figure 3.5: Find line with maximal c-value at $r = \lambda$ which has all points above or on it.

We now use the new interpretation to derive a combinatorial algorithm for solving the dual. Although equivalent to Handler and Zang's method, our formulation is simpler and more intuitive.

The Hull Approach: We are searching for the segment of the lower hull which intersects the limit line $r = \lambda$. We may compute points on the lower hull with shortest

[7]We use a sloppy definition of lower hull here: there is a horizontal segment to the right incident to the lowest point.

path computations, *e.g.* the extreme point in c-direction is the point corresponding to the minimal cost path and the extreme point in r-direction is the point corresponding to the minimal resource path.

We start with the computation of the minimum resource path. If this path is nonfeasible, *i.e.* all points lie to the right of the limit line, the dual LP is unbounded and the CSP problem is nonfeasible. Otherwise we compute a lower bound, the minimum cost path. If this path is feasible, we have found the optimum of CSP, and the constraint λ is redundant in that case.

Otherwise we have a first line through the two points that corresponds to a (u, v)-value pair where v is the slope and u is the c-intercept of the line.

Now we test whether all constraints are fulfilled, *i.e.* whether all points lie above or on the line. This separation problem is solved with a shortest path computation with the scaled costs $\tilde{c}_e = c_e - vr_e$ (see Lemma 3.3.1) and can be viewed as moving the line in its normal direction until we find the extreme point in that direction (see Figure 3.6). If no point below the line is found, we conclude that no constraint is violated and have

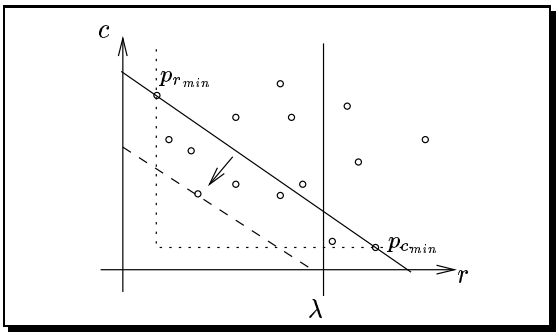

Figure 3.6: Finding a new point on the lower hull.

found the optimal line. The optimal value of the dual relaxation is its intersection with the limit line $r = \lambda$.

Otherwise we update the line: The new line consists of the new point and one of the two old points so that we again have a line that has all previously seen hull points above it and that intersects the limit line with maximal c-value.
We iterate this procedure until no further violating constraint is found.

Note that the hull approach is equivalent to Handler and Zang's method of Section 3.2.7. We operate in the more intuitive geometric dual (paths correspond to points) and they operate in the Lagrangean plane (paths correspond to lines). The absolute values for our variable v are therefore identical to the values of the Lagrangean multiplier μ (we have the identity $-v = \mu$).

3.3.3 Running Time of the Hull Approach

Now we show that the number of iterations for solving the relaxation with the hull approach for integral costs and resources is logarithmic in the input. This is the first result that gives nontrivial bounds for the number of iterations.

We assume that edge costs and resources are integers in the range $[0..C]$ and $[0..R]$, respectively. This implies that the cost and resource of a path lie in $[0..nC]$ and $[0..nR]$, respectively.

Theorem 3.3.1: The number of iterations of the hull approach is $O(\log(nRC))$. Hence the (Lagrangean) relaxation of CSP can be solved in $O(\log(nRC)(n \log n + m))$ time which is polynomial in the input.

Proof. We examine the triangle defining the unexplored region, *i.e.* the region where we may find hull points. The maximum area of such a triangle is $A_{max} = 1/2n^2RC$, the minimum area is $A_{min} = 1/2$. We will show in the following Lemma that the area of the triangle defining the unexplored region is at least divided by 4 in each iteration step of the hull 0. Since we have integral resources we may conclude that $O(\log(nRC))$ iterations suffice. □

Lemma 3.3.3: Let A_i and A_{i+1} be the area of the unexplored region after step i and $i+1$ of the hull approach, respectively. Then we have $A_{i+1} \leq 1/4A_i$.

Proof. Let A and B be the current feasible and nonfeasible hull point in step i and let sl_A and sl_B be the slopes which lead to the discovery of A and B, respectively. The line g_A through A with slope sl_A and the line g_B through B with slope sl_B intersect in the point C. The triangle T_{ABC} defines the unexplored region after step i (see Figure 3.7). Hence $A_i = A(T_{ABC})$.

Without loss of generality, consider an update of the current feasible point A[8]. Let A' be the new hull point that was discovered with slope sl_{AB} of the line through A and B.

We assume that A' lies on the segment $\overline{AC}$. The new unexplored region after the update step is defined by A', B, and the intersection point C' of line g_B and the line $g_{A'}$ through A' with slope sl_{AB} (see Figure 3.7). We have $A_{i+1} = A(T_{A'BC'})$.

Now the proportionality principle tells us something about the proportions of the side lengths of the two triangles. We have $|CA'|/|CA| = |CC'|/|CB|$ and $|CA'|/|CA| = |A'C'|/|AB|$. If we set $|CA'|/|CA| = x < 1$, we get $|C'B| = (1-x)|CB|$ and $|A'C'| = x|AB|$.

[8]The nonfeasible update is analogous.

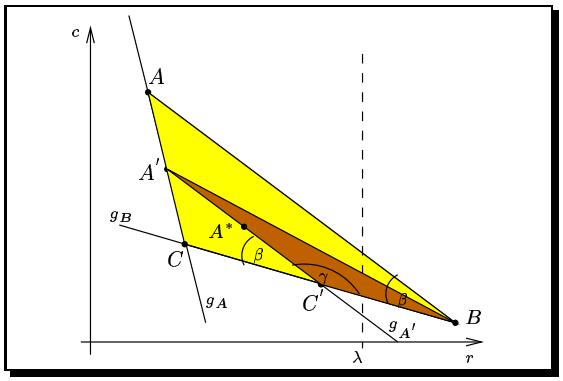

Figure 3.7: Area change of unexplored region in an update step.

The angle β between the segments AB and CB also appears between the segments $A'C'$ and CC', hence the angle γ between the segments $A'C'$ and $C'B$ is $\gamma = \pi - \beta$ (see Figure 3.7). Thus, $|\sin\gamma| = |\sin\beta|$.

The area of triangle T_{ABC} is $A_i = 1/2|AB||CB|\sin\beta$ and the area of the new triangle $T_{A'BC'}$ is $A_{i+1} = 1/2|A'C'||C'B|\sin\gamma = 1/2x(1-x)|AB||CB|\sin\beta = x(1-x)A_i$.

The product of the two side lengths of the new triangle is maximized for $x = 1/2$, *i.e.* when choosing A' as midpoint of the segment CA. Hence we have $A_{i+1} \leq 1/4A_i$.

It is easy to see that when we chose the new hull point A^* so that it lies on the line $g_{A'}$ but in the interior of the triangle T_{ABC} and form the new triangle $T_{A^*BC'}$, its area is smaller than the area of a triangle $T_{A'BC'}$. Hence $A_{i+1} \leq 1/4A_i$ for an arbitrary update step. □

The same asymptotic running time for solving the relaxation can be obtained by using binary search on the slopes. We will now elaborate this method that has already been outlined by Ahuja, Magnanti, and Orlin (1993) and was used in a modified way by Xue (2000), and compare it to the hull approach.

Binary Search: We again assume nonnegative integral costs and resources. The cost and the resource value of a path lie in $[0..nC]$ and $[0..nR]$, respectively. The slope of an edge of the lower hull is determined by two paths, say p and q, with $r_p \leq r_q$. The slope is therefore a rational number of the form $(c_q - c_p)/(r_q - r_p)$.

Lemma 3.3.4: A slope is either $-\infty$ or a rational number between $-nC$ and 0. Any two (finite) slopes differ by at least $1/(n^2R^2)$.

Proof. The numerator is an non-positive integer whose absolute value is bounded by

nC and the denominator is a non-negative integer bounded by nR. □

Now we can do the binary search on the slopes:

$s_1 = 0;$
$s_2 = nC;$

while $(s_2 - s_1 > 1/(n^2R^2))$ {

$s = \lfloor (s_2 - s_1)/2 \rfloor;$
compute shortest path p with respect to edge costs $c_e + sr_e$;
if $(r_p = \lambda)$ **break**;
if $(r_p < \lambda)$ $s_2 = s$; **else** $s_1 = s$;

}

When the program terminates we either have an optimal path whose resource consumption is exactly λ or two slopes s_1 and s_2 so that the line with slope s_1 touches the lower hull to the right of the vertical line $r = \lambda$, the line with slope s_2 touches the lower hull to the left of the vertical line, and so that $s_1 - s_2 \leq 1/(n^2R^2)$. Let p and q be the two paths corresponding to the two extreme points. The segment $\overline{pq}$ must be a hull segment since p and q lie on the hull and since the slopes of the tangents in p and q differ by at most $1/(n^2R^2)$.

Theorem 3.3.2: The binary search algorithm solves the relaxation of CSP in $O(\log(nRC))$ iterations which is polynomial in the input.

Proof. The difference of the slopes of the current left and right hull point is halfed in each iteration until it is smaller than $1/(n^2R^2)$ (starting from nC). Hence the number of iterations is bounded by $O(\log(nRC))$. □

This means that the number of iterations performed by binary search to compute the relaxation is asymptotically the same as the number of iterations performed by the hull approach. As binary search is conceptually simpler than the derivation of the hull approach, we might ask about the advantages of the hull approach.

We now give both a theoretical and a qualitative argument for prefering the hull approach in practice:

Although asymptotically equivalent, the running time of the hull approach has better constants. Here we have the stop criterion $1/2n^2RC/4^i < 1/2$ which means that we need at most $i \leq \log(nRC)$ iterations, *i.e.* the constant hidden in the Big-Oh notation is 1.

The stop criterion in the binary search approach is $nC/2^i < 1/(n^2R^2)$ which means that we need at most $i \leq 6\log(nRC)$ iterations. Hence the running time of the hull approach is better by a factor of 6.

This can also be explained more informally. The hull approach is more goal driven and detects optimality in a single step, whereas binary search is not as adaptive and has to continue the iteration until the two slopes differ by less than $1/(n^2R^2)$ to be able to guarantee optimality.

Worst Case Scenario

We proved that the hull approach solves the relaxation in $O(\log(nRC))$ iterations. Now, the question is whether this worst case bound can actually be reached with a problem instance. Our proof suggests a geometric scenario where the hull approach runs into this worst-case bound. We simply place the points so that in each step the area where we may still find points is exactly divided by 4.

Lemma 3.3.5: Given the points $p_{-1} = (nR, 0), p_i = ((1 - (i+1)/2^i)nR, nC/2^i)$ for $i = 0, 1, \ldots, \log(nC)$ and $\lambda = nR - 1$, the hull approach has to visit all the points, hence performing $O(\log(nRC))$ iterations, until it terminates.

Proof. The proof is by induction on the number of steps i following the ideas of the iteration proof. □

Figure 3.8 represents the geometric worst case scenario. It remains to find a real network

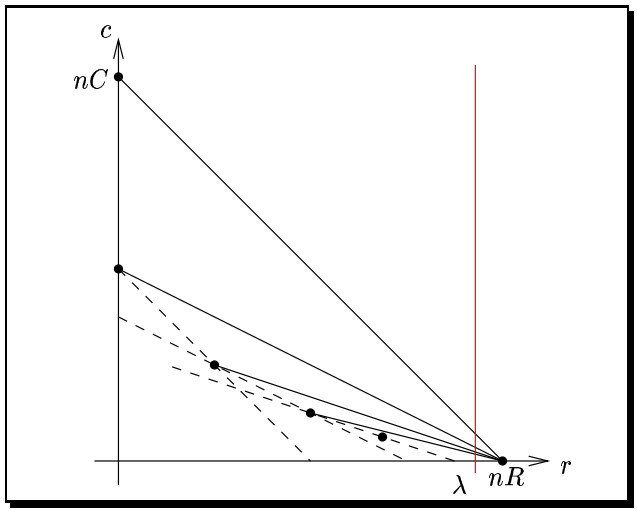

Figure 3.8: Geometric worst case for the hull approach.

corresponding to this geometric situation. The rules for logarithm-computation tell us that $O(\log(nRC) = O(\log \max\{n, R, C\})$. We first assume that $C \geq n$ and without loss of generality that $R = C$. Figure 3.9 shows a network that forces the hull approach into $\log R$ iterations if λ is set to $R - 1$.

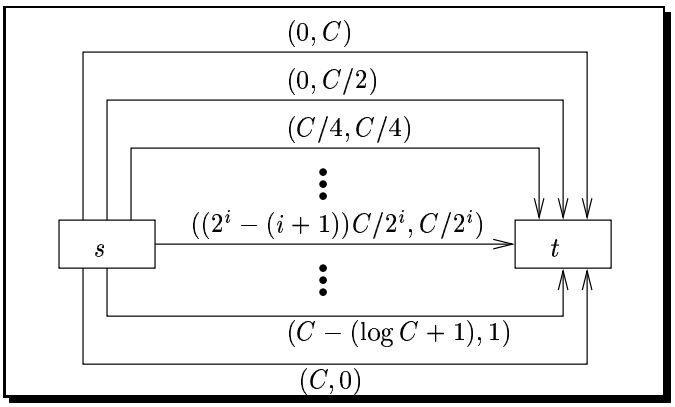

Figure 3.9: Network giving the worst case bound for $R = C \geq n$.

Now we consider the case $n \geq R, C$ and assume without loss of generality that $R = C = 1$ and that n is a power of 2. Figure 3.10 shows a network with $2n + 2$ nodes and $2n + \log n + 2$ edges. The network realizes $\log n + 2$ paths leading to points $p_{-1} = (n, 0), p_i = ((1 - (i + 1)/2^i)n, n/2^i)$ for $i = 0, 1, \ldots, \log n$. This forces the hull approach

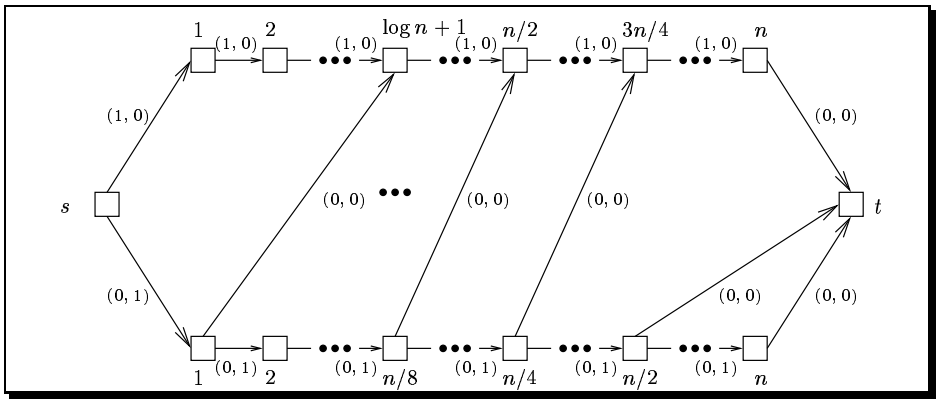

Figure 3.10: Network giving the worst case bound for $n \geq R, C$.

into $\log n + 1$ iterations when we have a resource limit $\lambda = n - 1$. Hence we can conclude that our worst case bound is indeed tight.

The bound for the binary search method is $\Theta(\log(nRC))$ in the given version since we always have to continue the iteration until the slopes differ by less than $1/(n^2R^2)$

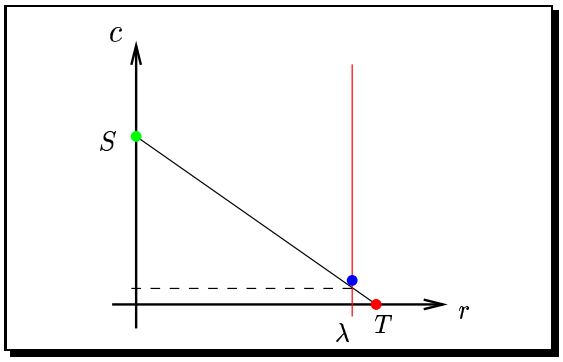

Figure 3.11: Bad quality of the bounds.

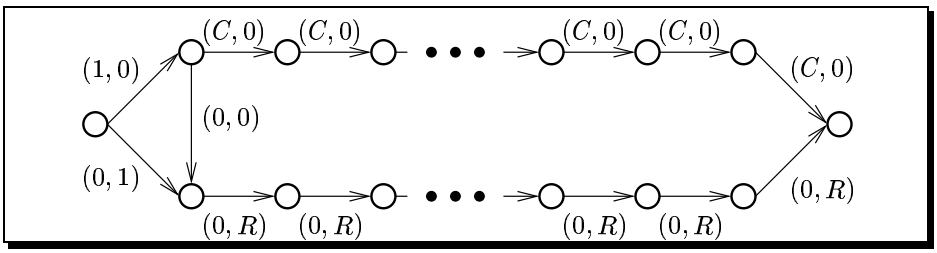

Figure 3.12: Network with n nodes ($n/2-1$ nodes in the upper and lower chain). Assuming a resource limit $\lambda = nR/2$ and $R \geq C$ we realize the geometric situation of Figure 3.11 and get a $(nC/2+1)$-approximation of the optimum.

except when we hit the true optimal solution, *i.e.* an extreme point p with $r_p = \lambda$.

Quality of the Bounds

The hull approach gives us a feasible path as an upper bound whereas the optimum of the relaxation is a lower bound for CSP. The question is: how good are the bounds?

We will give a simple example showing that the bounds come with no constant approximation guarantee. Again, it is easy to construct a geometric example (see Figure 3.11). We have only three points p_1, p_2, and p_3. Two of them, $p_1 = (0, S)$ and $p_2 = (T, 0)$, define the lower hull. The resource limit λ is set to $T-1$, so p_1 is feasible and p_2 is nonfeasible. The third point denoting the optimum is $p_3 = (\lambda, \lceil S/T \rceil)$. We have $UB = S$, $LB = S/T$, and $OPT = \lceil S/T \rceil$. Without loss of generality, we assume that $T \geq S$ and receive a S-approximation in the example under consideration. Figure 3.12 shows us that S can be $O(nC)$, hence the approximation can be made arbitrarily bad. Slight modifications also show that the obtained lower bound does not yield a constant approximation of the optimum.

Despite these negative theoretical results for pathological examples we will see in our experiments that the bounds are usually very good considering networks arising in practical situations.

A Different Interpretation: The Simplex Approach

In the following we want to interpret the hull approach as a dual simplex algorithm with cutting plane generation. This will lead to the generalization for multiple resources.

Cutting plane generation is a commonly used technique when the number of constraints is very large compared to the number of variables. This is precisely our situation, we only have two variables but an exponential number of constraints in the worst case, moreover these are only implicitly given.

So let p be the minimal resource path. If $r_p > \lambda$, the dual LP is unbounded and CSP is nonfeasible. So assume that $r_p \leq \lambda$ and let $P = \{p\}$. Then $(u^*, v^*) = (c_p, 0)$ is an optimal solution for the linear program $\max u + \lambda v$ subject to $v \leq 0$ and $u + v r_p \leq c_p$ for all $p \in P$.

More generally, assume that P is a set of s–t paths and that (u^*, v^*) is an optimal solution to the LP, $\max u + \lambda v$ subject to $v \leq 0$ and $u + v r_p \leq c_p$ for all $p \in P$. It follows from complementary slackness that if $v^* < 0$ then there are two paths p_1 and p_2 with $u^* + v^* r_{p_i} = c_{p_i}$ $(i = 1, 2)$ and if $v^* = 0$ then there is a path p_1 with $u^* + v^* r_{p_1} = c_{p_1}$. Moreover, $r_{p_1} \leq \lambda$ and $r_{p_2} > \lambda$, *i.e.* the paths define the segment of the lower hull of the points $\{(r_p, c_p) : p \in P\}$ that is intersected by the line $r = \lambda$.

A shortest path computation with edge costs $\tilde{c}_e = c_e - v r_e$ checks whether the constraint $u^* + v^* r_p \leq c_p$ is satisfied by all s–t paths. If yes, we have the optimal solution of the dual. If not, the shortest path computation gives a path q most violating the constraint.

Next we use the dual simplex algorithm to find the optimum for the constraint set $P \cup \{q\}$. In the single resource case, this is possible with a single pivot step, moreover, the new optimum is defined by q and one of the paths p_1, p_2 defining the old optimum. This follows from the construction history since it defines a region where the point (r_q, c_q) corresponding to the new path q may lie (see Figure 3.13). The only points of the current lower hull that are seen by the new point are the two points corresponding to p_1 and p_2. Hence, the new optimum is easily obtained. We terminate when no more violated constraints can be found.

This is the simplex interpretation of the hull approach. Hence, the same running time as for the hull approach applies for this cutting plane generation approach.

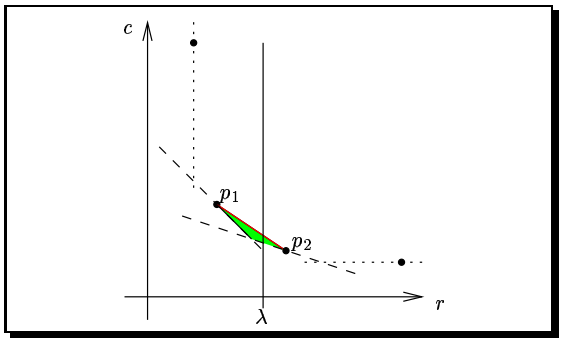

Figure 3.13: The "unexplored" region.

3.3.4 The Multiple Resource Case

We return to the general case of k resources. Beasley and Christofides' method also approximates the optimum for the dual relaxation in the multiple resource case whereas Handler and Zang leave this open for future work. We restate the relaxation problem:

$$\begin{aligned} \max \quad & u + \lambda^{(1)} v_1 + \cdots + \lambda^{(k)} v_k \\ \text{s.t.} \quad & u + r_p^{(1)} v_1 + \cdots r_p^{(k)} v_k \leq c_p \quad \forall p \\ & v_i \leq 0 \qquad i = 1, \ldots, k \end{aligned}$$

We again give two interpretations of the dual relaxation:

Standard Interpretation: A tuple $(u, v_1, \ldots, v_k)$ is interpreted as a point in u-v_1-$\cdots$-v_k-space. We view each constraint as a halfspace in u-v_1-$\cdots$-v_k-space and search for the maximal feasible point in direction $(1, \lambda^{(1)}, \ldots, \lambda^{(k)})$. The optimum is defined by $k + 1 - l$ constraints that are satisfied with equality and is parallel to l coordinate axes[9].

(Geometric) Dual Interpretation: A tuple $(u, v_1, \ldots, v_k)$ is interpreted as a hyperplane in $r^{(1)}$-$\cdots$-$r^{(k)}$-c-space (the hyperplane $c = v_1 r^{(1)} + \cdots + v_k r^{(k)} + u$). A constraint $u + v_1 r_p^{(1)} + \cdots + v_k r_p^{(k)} \leq c_p$ is interpreted as a point $(r_p^{(1)}, \ldots, r_p^{(k)}, c_p)$ in $r^{(1)}$-$\cdots$-$r^{(k)}$-c-space. We are searching for a hyperplane $c = v_1 r^{(1)} + \cdots + v_k r^{(k)} + u$ with nonpositive v_i's, that has all points $(r_p^{(1)}, \ldots, r_p^{(k)}, c_p)$ above or on it, and has maximal c-value at the limit line $(r^{(1)}, \ldots, r^{(k)}) = (\lambda^{(1)}, \ldots, \lambda^{(k)})$ (see also Figure 3.14).

We again describe two approaches to solve the dual relaxation based on the different interpretations:

[9] Where l is the number of v_i's that are zero.

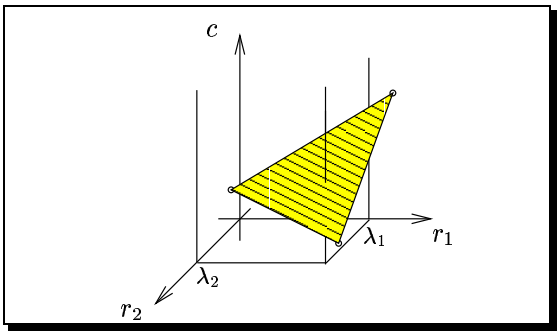

Figure 3.14: Two resources.

Simplex Approach: We extend the dual simplex method with cutting plane generation to multiple resources. Contrary to the single resource case, it is not clear how to efficiently find a feasible path that bounds the optimal value. We add an artificial path p from s to t which consists of a single edge of very large cost[10] and uses no resources, and set $P = \{p\}$. The constraint for p guarantees finiteness: Let u and v_i be a feasible solution of the dual relaxation. Then $v_i r_p^{(i)} \geq v_i \lambda^{(i)}$ for all i by the feasibility of p and the non-positivity of the v_i's and hence

$$u + \sum_i v_i \lambda^{(i)} \leq u + \sum_i v_i r_p^{(i)} \leq c_p.$$

The optimal solution for the LP, $\max u + \sum_i v_i \lambda^{(i)}$ subject to $v_i \leq 0$ and $u + \sum_i v_i r_p^{(i)} \leq c_p$ for all $p \in P$ is given by $u = c_p$ and $v_i = 0$ for all $i = 1, \ldots, k$.

More generally, assume that P is a set of s–t paths and that $(u^*, v_1^*, \ldots, v_k^*)$ is an optimal solution to the LP, $\max u + \sum_i \lambda^{(i)} v_i$ subject to $v_i \leq 0$ and $u + \sum_i v_i r_p^{(i)} \leq c_p$ for all $p \in P$. It follows from complementary slackness that if $v_i^* = 0$ for l of the v_i's then there are $k + 1 - l$ paths $p_1, \ldots p_{k+1-l} \in P$ with $u^* + \sum_i v_i^* r_{p_j}^{(i)} = c_{p_j}$ for $j = 1, \ldots, k + 1 - l$.

Now we check whether there are violated constraints, *i.e.* we ask whether there is a path q with $\hat{u} + \sum_i \hat{v}_i r_q^{(i)} > c_q$ where $(\hat{u}, \hat{v}_1, \ldots, \hat{v}_k)$ is the current optimal solution. This separation problem is again solved by a shortest path computation with scaled costs $\tilde{c}_e = c_e - \sum_i \hat{v}_i r_e^{(i)}$.

In the first iteration, we have $\hat{v}_i = 0$ for all $i = 1, \ldots, k$ and compute the minimum cost path. If this is feasible for all constraints, we have found the optimum for CSP.

Otherwise, we use the dual simplex algorithm to find the optimum for the set $P \cup \{q\}$. Contrary to the single resource case, optimizing may need more than one pivot step

[10] *e.g.* nC since the cost of every simple path is bounded by $(n-1)C$

and the new optimum may also consist of other points in P that have not been part of the previous optimum. This is due to the following fact: The area where new points can be found is a simplex determined by the halfspaces that correspond to the points that define the current optimum, and the hyperplane defining the current optimum (see Figure 3.15). In contrast to the single resource case, the current optimal hull facet does not "block" previous points from that region, the new point may also see previous hull points. Hence, they might also be part of a new optimal solution.

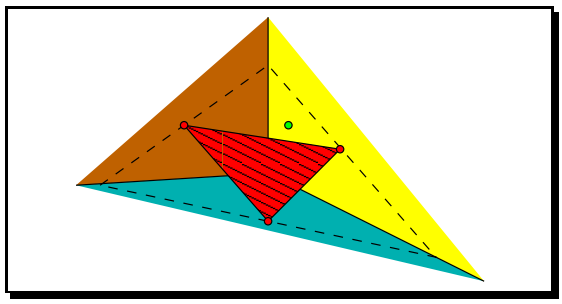

Figure 3.15: The unexplored region in the multiple resource case.

We iterate this procedure and terminate when no more violated constraints can be found.

Geometric Approach: We start with the artificial feasible point and then compute the minimum cost point. Then, in each iteration, we have to determine the optimal face[11] of the lower hull[12] seen so far. Since a newly computed hull point q might also see previous hull points not belonging to the old optimal face (as explained above), we have more candidates for the new optimal solution and hence, the update step is not as easy as in the single resource case. We have to determine the q-visible facets of the old hull to update the new lower hull as in an incremental $k+1$-dimensional convex hull algorithm (Clarkson, Mehlhorn, and Seidel 1993). The new optimal face is the face incident to q that intersects the limit line[13].
Now we check whether there is a violated constraint. This is done by solving a shortest path problem with scaled costs which again can be seen as moving the facet defining hyperplane in its normal direction until we reach the extreme point in this direction.

We stop the iteration when no more violating constraints are found. If the artificial

[11] *i.e.* the face intersecting the limit line

[12] A point is again treated as rays in $r_1, \ldots, r_k$-directions to ensure that all faces of the lower hull have a normal vector with nonpositive entries.

[13] It may be parallel to some coordinate axis $r^{(i)}$ indicating that the dual variable v_i is zero.

feasible point is part of the optimal solution then the problem is nonfeasible[14].

Bounding the number of iterations is difficult for the multiple resource case but Lemma 3.3.1 tells us that the dual relaxation can be solved in polynomial time using the Ellipsoid method.

Due to the more complicated update step, we could not extend the geometric arguments in the proof for the number of iterations of the single resource case. However, if we assume that the volume where we may still find points corresponding to violating constraints decreases by a factor of $Z > 1$ per iteration of the multiple resource hull approach, we get the following conjecture.

Conjecture 1: Let C be the maximum cost and $R_1, \ldots, R_k$ the maximum resource consumption of an edge. Then the CSP relaxation for k resources can be solved with $O(k \log(nCR_1 \cdots R_k))$ shortest path computations.

The biggest initial volume is bounded by $\frac{1}{n!}n^{k+1}CR_1 \cdots R_k$, the smallest volume by $1/n!$. Then, we have the break criterion

$$\frac{\frac{1}{n!}n^{k+1}CR_1 \cdots R_k}{Z^i} < \frac{1}{n!}$$

which implies that

$$i < \frac{k+1}{\log Z} \log(nCR_1 \cdots R_k).$$

The conjecture is supported by our experiments in the next section.

3.3.5 The Parametric Shortest Path Problem

Before moving on to the central problem of closing the gap, we discuss some extensions of our problem focussing on the single resource case. Taking a look at the relaxation we see that we are left with a parametric shortest path problem and our aim is to find the optimal parameter for a given resource limit. Two generalizations come to mind:

1. If we don't have a resource constraint, we are interested in the bicriteria problem and would like to know the whole hull.

2. At the moment we always consider paths from s to t. How about a bicriteria (parametric) single source shortest path problem (SSSP) ?

[14] Only if the artificial path had infinite cost.

Bicriteria Shortest Paths - Computing the Whole Hull

In the original CSP formulation we are given a resource limit and the hull approach computes the hull segment intersecting the limit line giving the optimum of the Lagrangean relaxation. In the *bicriteria* shortest path problem we do not have a resource limit and are interested in *Pareto-optimal* paths, *i.e.* paths that dominate others. If we compute the whole lower hull we get all the extreme Pareto-optimal paths and, given an arbitrary resource limit, would be able to report the optimum of the corresponding CSP relaxation immediately.

We can compute the whole lower hull with an easy modification of the hull approach. In the hull approach, starting with the minimum cost and minimum resource point, we computed a new hull point (scaling with the slope of the segment defined by the two points) and always had one recursive call with the new tentatively optimal basis segment that was determined by the limit λ. In the absence of the resource limit we now have two recursive calls. We abort a recursive call if we have found no violating constraint.

How many recursive calls do we need to compute the whole lower hull ? In each recursive call we either find a new hull point and continue with two recursive calls or we see that we can stop the recursion. Hence, the number of shortest path computations is at most twice the number of points on the lower hull. If we again consider integral costs and resources in $[0..C]$ and $[0..R]$ we obtain the following result.

Lemma 3.3.6: The whole lower hull can be computed in $O((n \max\{C, R\})^{2/3})$ iterations using the modified hull approach.

Proof. Har-Peled (1998) has shown that the number of points on an integer hull with diameter D is bounded by $O(D^{2/3})$. The lemma follows since the modified hull approach has no problems with degeneracies (they at most double the number of iterations). □

An example with $2\sqrt{nC}$ points on the hull can be easily obtained when choosing the points

$$p_i = (i, nC - i\sqrt{nC} + \sum_{j=1}^{i-1} j) \qquad i = 0, 1, \ldots, \sqrt{nC}$$

and their reflections on the main diagonal $q_i = (p_i^{(y)}, p_i^{(x)})$ (here we assume that $\sqrt{nC}$ is integral). It is easy to verify that all points lie on the hull (see Figure 3.16), the points $p_{\sqrt{nC}}$ and $q_{\sqrt{nC}}$ are collinear with the points $p_{\sqrt{nC}-1}$ and $q_{\sqrt{nC}-1}$ and will not be counted. A corresponding network realizing this geometric scenario can be easily constructed following the lines of Figure 3.10.

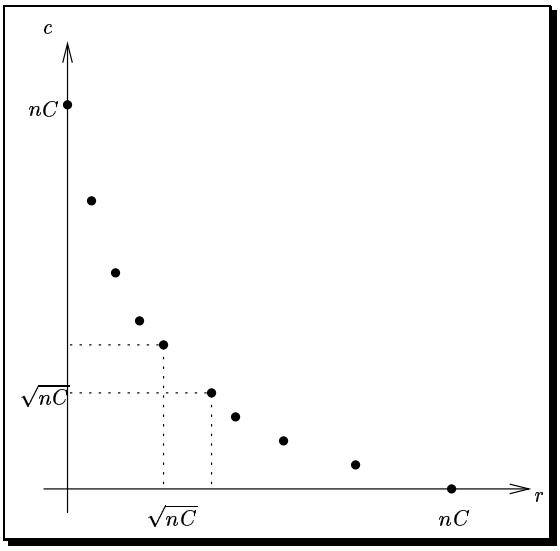

Figure 3.16: Case of $\sqrt{nC}$ points on the lower left hull.

Parametric SSSP - Can we deal with many targets?

In the parametric single source shortest path problem the cost function of each edge is a linear function in μ. We have $\tilde{c}_e = c_e + \mu r_e$ and we want to obtain a tree T^μ of shortest paths from a source node s to all other nodes for all values of μ from 0 to ∞.

The question now is how many different trees may exist and how can we compute them. Ahuja, Magnanti, and Orlin (1993) outline the following algorithm.

Let $d^c(j), d^r(j)$, and $d^{\tilde{c}}(j)$ denote the shortest distance between nodes s and j with respect to the cost functions c, r, and $\tilde{c}$. Consider T^μ for some μ. If $d^c(j)$ and $d^r(j)$ are the distances in T^μ with respect to cost functions c and r, then, by the shortest path optimality conditions, $d^c(j) + \mu d^r(j)$ is the distance with respect to cost function $\tilde{c}$ in T^μ. We now define μ_{kl} for a nontree edge $e = (k, l)$ by

$$\mu_{kl} = \begin{cases} -c^{d^c}_{kl}/c^{d^r}_{kl} & \text{if } c^{d^r}_{kl} < 0, \\ \infty & \text{otherwise.} \end{cases} \tag{3.21}$$

with $c^{d^r}_{kl} = r_{kl} + d^r(k) - d^r(l)$. Let $\bar{\mu} = \min\{\mu_{kl} | (k, l) \notin T^\mu\}$. Then the tree T^μ remains a shortest path tree as long as $\mu \leq \bar{\mu}$. Let edge (p, q) be a nontree edge for which $\mu_{pq} = \bar{\mu}$. Then, adding edge (p, q) to T^μ and dropping the unique tree edge entering node q gives us an alternate shortest path tree at $\mu = \bar{\mu}$.

Hence, starting with T^0 which is simply the shortest path tree with respect to cost function c, we repeatedly use the method described above to obtain successive shortest

path trees for increasing values of the parameter μ, until $\mu = \infty$. It follows from our definition (3.21) that this will only happen when $c_{kl}^{d^r} \geq 0$ for each nontree edge (k,l). The last tree will be the same as the shortest path tree computed with respect to the cost function r.

What can we say about the number of iterations of the previous algorithm, *i.e.* the number of different shortest path trees ? We have a look at the following potential function $\phi = \sum_{v \in V} d^r(v)$ as suggested by Ahuja, Magnanti, and Orlin (1993). If R is the maximal resource value of an edge, then we can conclude that $\phi \leq n^2 R$. Now we take a look at the effect of a change in the shortest path tree T^μ on the potential function ϕ: A nontree edge $e = (k,l)$ can only change T^μ if $c_{kl}^{d^r} < 0$, or alternatively $d^r(l) > d^r(k) + r_{kl}$. After the change we have $d^r(l) = d^r(k) + r_{kl}$. This means we have a strict decrease in the potential function ϕ. Therefore, we can conclude that we need at most $O(n^2 R)$ iterations of the previous algorithm that runs in $O(m)$ time after the initialization. Hence, we can solve the parametric shortest path problem in $O(mn^2 R)$ time.

If we compute the whole hull for the CSP relaxation given a target node t, the slopes of the hull segments are a subset of the μ-values giving rise to different shortest path trees[15]. Thus, computing the whole hull for all nodes in the graph, we may also get the different values of μ in $O(n(nR)^{2/3})$ iterations assuming $R \geq C$ without loss of generality. This would give us a running time of $O(n(nR)^{2/3}(n \log n + m))$ for the parametric shortest path problem which is a better bound for large or dense graphs.

Multiple target CSP is a different story, however. Given a resource limit λ, the parameter leading to the optimum of the relaxation for one target node does not have to do anything with the one for another target node.

3.4 Closing the Gap

We have seen in the previous section how we can solve the Lagrangean relaxation of CSP efficiently. However, integer programming duality tells us that a duality gap may exist. In this section we will present several methods for closing this gap to obtain the optimal solution of CSP. Before we come to the discussion of gap-closing we will review pruning methods that reduce the problem size using the resource limit(s) and upper and lower bounds.

[15] Two points defining a segment denote two alternative shortest paths from s to t using the slope of the segment as parameter. Using all parameters between the slope of the preceding hull segment and this hull segment gives us the left endpoint. All parameters between the slope of this segment and the slope of the subsequent hull segment lead to the right endpoint.

3.4.1 Problem Reductions

We review possibilities for reducing the problem size by pruning nodes and edges that cannot be part of an optimal solution. The hope is that this will speed up the gap-closing step.

Resource-based reductions

Let $R_{vw}^{(i)}$ be the least amount of resource i that we can use when going from node v to node w $(i = 1, \ldots, k)$. Then, any node v for which a resource i $(i = 1, \ldots, k)$ exists so that

$$R_{sv}^{(i)} + R_{vt}^{(i)} > \lambda^{(i)}$$

can be eliminated from the problem as it cannot lie on the optimal path. A similar formula exists for edges. The resource-based reductions were first given by Aneja, Aggarwal, and Nair (1983). The time needed for the resource reductions is $2k$ SSSP computations[16] to compute the values $R_{vw}^{(d)}$, followed by an iteration over all nodes and edges.

The resource reductions are possible right at the beginning, before computing the relaxation. We will see in the experimental section whether the time spent in the problem reductions will actually pay off in the subsequent computation.

Cost-based reductions

Given a feasible upper bound UB on the optimal solution of CSP we can extend the idea of the resource reductions to cost reductions. If C_{vw} denotes the minimal cost from node v to node w, then any node v with

$$C_{sv} + C_{vt} > UB$$

can be pruned. A similar condition exists for the edges. The effectivity of the cost reductions depends on the quality of the upper bound.

Dumitrescu and Boland (2000) propose a repeated reduction method. Their idea is simple: They eliminate nodes and arcs that cannot be part of an optimal solution using the pruning techniques described above and update upper and lower bounds on the fly. If the network was reduced in such a step, this process is reiterated as the

[16] The first k SSSP computations have s as starting node then we reverse all edges and use t as starting node for the last k SSSP computations before restoring the original graph.

bounds might have changed. This method performs very well for hard constrained problems but fails otherwise since detecting a better upper bound during the iteration is mere luck.

Beasley and Christofides (1989) have shown how to use upper *and* lower bounds from the relaxation for pruning.

Consider any feasible path p with coordinates $(r_p^{(1)}, \ldots, r_p^{(k)}, c_p)$, and any $\vec{v} = (v_1, \ldots, v_k)$ with $v_i \leq 0$ for $i = 1, \ldots, k$. Then $r_p^{(i)} \leq \lambda^{(i)}$ and $-v_i r_p^{(i)} \leq -v_i \lambda^{(i)}$ for all i and hence

$$c_p - \sum_{i=1}^{k} v_i r_p^{(i)} + \sum_{i=1}^{k} v_i \lambda^{(i)} \leq UB$$

for any given upper bound $UB \geq c_p$.

Let $\tilde{C}_{vw}$ be the shortest path from node v to node w with respect to cost function $\tilde{c}_e = c_e - \sum_i v_i r_e^{(i)}$. If

$$\tilde{C}_{sv} + \tilde{C}_{vt} + \sum_{i=1}^{k} v_i \lambda^{(i)} > UB$$

then there can be no feasible path through node v with costs smaller than UB, and node v can be deleted. A similar formula for edges is also easily derived.

Let $\hat{u}, \hat{v}_1, \ldots, \hat{v}_k$ be the optimal solution of the dual relaxation. Since $\hat{u}$ is the cost of the shortest path with respect to cost function $\tilde{c}_e = c_e - \sum_i \hat{v}_i r_e^{(i)}$ and the objective function $u + \sum_i v_i \lambda^{(i)}$ is maximized for $\hat{u}, \hat{v}_1, \ldots, \hat{v}_k$, this choice of the v_i seems to be good for obtaining a large problem reduction. We will support this claim in our experimental section.

A geometric interpretation of the problem reductions can be obtained from Figure 3.17. Paths to the right of the limit line and above the upper bound line cannot be optimal, so the hope is to eliminate a large number of these with the resource and cost reductions. We are also not interested in paths that lie above the line parallel to the optimal hull segment that intersects the limit line λ with cost value UB. This motivates the lower bound cost reductions. However, it should be noted that we cannot guarantee the elimination of all paths in the uninteresting areas since they might pass through nodes and edges being part of the optimal solution.

3.4.2 Closing the Duality Gap

Now we come to the central gap-closing step. We review the gap-closing methods proposed by Handler and Zang (1980) and Beasley and Christofides (1989). Then we show how we can adapt labeling methods to do efficient gap-closing.

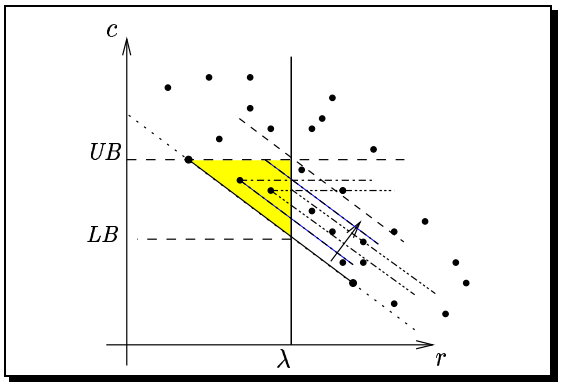

Figure 3.17: Closing the gap between upper and lower bound.

Let us first give an intuitive geometric interpretation of the gap-closing step[17]. Figure 3.17 shows the situation after solving the relaxation. The relaxation optimum is the c-value of the intersection of the optimal hull segment found by the hull approach with the limit line λ, providing a lower bound on the optimal solution of CSP. The left endpoint of the optimal hull segment corresponds to a feasible path and its cost provides an upper bound. Now, the true optimum may lie in the triangle defined by the optimal hull segment, the limit line, and the upper bound line (see shaded area in Figure 3.17). A similar interpretation exists for the multiple resource case.

Path Ranking

Handler and Zang (1980) use a k-shortest path algorithm to close the duality gap. They observed that path ranking with respect to the scaled costs leading to the relaxation optimum provides a way to continue "naturally" from the dual solution. They update upper and lower bounds along the way and stop when the duality gap is closed, *i.e.*, $LB \geq UB$ which means that $OPT = UB$.

We now give a simple geometric intuition. Closing the duality gap by path ranking with respect to the Lagrangean costs means moving the line through the optimal hull segment in its normal direction and thus "sweeping" the area where the true optimum may lie (see Figure 3.17). The path ranking will start with the two segment defining hull points since they both have minimal Lagrangean cost[18]. The intersection of the "sweepline" with the limit line λ depicts the lower bound progression. When we encounter a feasible path improving the previous upper bound, the "sweep area" gets smaller. We can stop sweeping, *i.e.*, ranking once the possible area is completely swept, *i.e.*, $LB \geq UB$.

[17] for the single resource case

[18] Of course there also may be other paths with identical minimal Lagrangean cost.

We may also see from the geometric interpretation that although the area where the optimal solution may lie can be small, we might have to sweep a much larger area to detect optimality. So we end up enumerating a lot of paths that are known to be nonfeasible or worse than the current upper bound. Below, we attempt a different approach that only considers feasible paths that improve the upper bound, pruning others away once we discover that they are not important.

A Tree Search Procedure

Beasley and Christofides (1989) suggested using a depth-first tree-search procedure to close the duality gap. Their method can be seen as *branch and bound* technique.

Let $\tilde{c}$ be the cost function that is scaled with parameters $\hat{v}_i \leq 0$, *i.e.*, $\tilde{c}_e = c_e - \sum_i \hat{v}_i r_e^{(i)}$. Typically, the $\hat{v}_i$ are the parameters leading to the optimum of the Lagrangean relaxation. Starting from the source node s, we do a depth-first tree-search procedure. If v is our current branching node we have a subpath from s to v that can potentially be extended to an optimal path. Then, we chose the edge $e = (v, w)$ for which $x_e = 1$ when computing shortest paths from node v with cost function $\tilde{c}$.
Now we do reductions, eliminating nodes and edges that would force the subpath into nonfeasibility or its cost over the best upper bound.
We can backtrack in the tree when the current lower bound exceeds the best known upper bound or when all outgoing edges of the current branching node have been eliminated by problem reductions. Backtracking involves restoring pruned nodes and edges. Backtracking from node s means that there is no feasible path in the network.
This is basically a branch and bound scheme. We fix values of some variables x_e and compute lower bounds for the corresponding subproblems, branching from them if they might be extended to an optimal solution, and discarding them when we detect nonfeasibility or no improvement on the upper bound.

This method is not efficient in practice. We will see in the following how to implement the basic idea much more efficiently.

Labeling Methods

We have already discussed labeling methods in Section 3.2.4. Now we go into more detail keeping our focus on the single resource case although the methods also work for the multiple resource case. In the standard shortest path problem, labeling methods keep the tentative distance of the shortest path from s to v as a label in each node v. In the CSP case, a label representing an s-v-path has a cost *and* a resource value. We need to store a list of labels per node since there is no longer a total order among

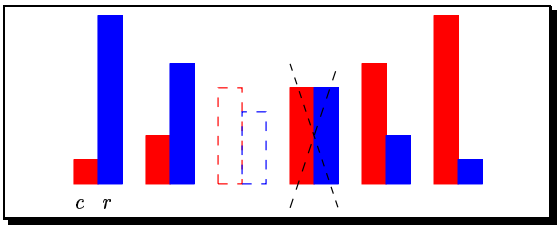

Figure 3.18: Inserting a new label in the label list.

labels. Only the notion of *domination* exists. A label (c_p, r_p) dominates another label (c_q, r_q) if $c_p \leq c_q$ and $r_p \leq r_q$. Hence we keep a list of non-dominated labels in each node and keep it sorted in increasing cost order to facilitate update operations.

Now we distinguish two methods, *label setting* like Dijkstra's algorithm and *label correcting* like Pape/Moore's algorithm. The CSP counterparts of label setting algorithms have been introduced by Aneja, Aggarwal, and Nair (1983) and Desrosiers, Dumas, Solomon, and Soumis (1995), label correcting approaches were proposed by Desrosiers, Dumas, Solomon, and Soumis (1995), and Lauther (2001), Stroetmann (1997). We now discuss these two methods in detail and show how we may make use of the relaxation bounds.

Label Setting: As in Dijkstra's algorithm we use a priority queue PQ to drive the consideration of labels from which to propagate. The priority queue is ordered by minimal completion cost to aim for optimal cost completion[19]. While the priority queue is non-empty we extract the minimum cost label. Let v be the end node of the corresponding path. Since we consider the labels in increasing cost order we can prune all labels with smaller cost from the label list at node v since they must have all been considered already. This is done to keep the label lists small and thus to decrease update time. Then we check whether the extracted label is still non-dominated[20]. This is the case if we find the label at the head of the label list[21]. Now we propagate from node v, and consider all outgoing edges $e = (v, w)$. If the minimal cost completion of the enlarged path is beyond the current upper bound and if the minimal resource completion is still feasible we update the label list of node w. This is done as follows (see also Figure 3.18). Again, we first prune labels with cost less than that of label l. Then, we go through the list to locate the cost position of the enlarged label l'. If we encounter a label that dominates l' we discard label l', otherwise we insert it

[19] In the single resource case we could also order it by minimal resource completion to aim for feasibility. This should be better for hard constrained problems.

[20] Note that we might have found something better in the meantime.

[21] In the multiple resource case we would have to check all labels with the current cost.

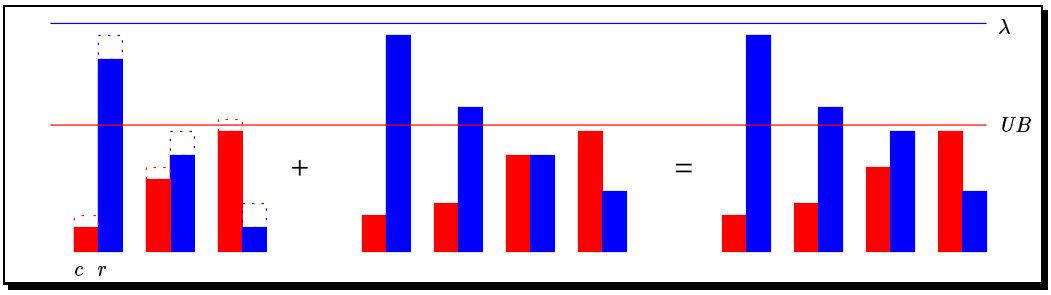

Figure 3.19: Propagating an updated label list.

in the list. After insertion we check whether we can prune subsequent labels since l' might dominate them. We can stop pruning after the first non-dominated label is encountered[22]. If the new label l' was non-dominated, we also insert it into the priority queue and continue. We stop with the optimum once a path has reached the target since this will be a path of minimal cost and will be feasible.

Using the upper bound from the relaxation we expect to consider fewer labels and hence expect a better practical performance. This only takes the upper bound into account, but we would like to continue from the lower bound as in the case of path ranking. Hence we use the lower bound (completion) costs in the priority queue. Now we can stop the iteration when the lower bound completion cost value at the limit line λ has reached the current upper bound. This also means that we might reach several feasible paths improving the upper bound. The problem now is that the pruning and the update of the label list is more difficult. If we keep the label list as it is, we cannot do the initial pruning since we consider labels in lower bound completion cost order. If we store the labels in lower bound cost order in the label list, we loose the simple increasing/decreasing structure in the single resource case and checking for domination is more difficult. We will see in the experiments that the best strategy is to skip the pruning in the single resource case and to do pruning with the labels stored in lower bound order in the multiple resource case since here there are fewer dominating paths.

Label Correcting: In the label correcting case we do not use a priority queue having labels as entries but a list of nodes instead. While this list is non-empty, we remove the first node and propagate from this node to all adjacent nodes. Propagation now means to "merge" the label list of the current node enlarged with the new edge with the label list of the adjacent node while removing labels that cannot possibly be extended to

[22] This is trickier for the multiple resource case, though.

optimal paths (see Figure 3.19)[23]. We keep track of the nodes in the list and only push or append a node that is not in the list when the label list was modified. We push a node if the label list of the adjacent node was previously empty and append otherwise. The idea is that once a label list has been modified we have to update the adjacent nodes at some point and this should rather be done immediately instead of using the wrong labels again before the update.

Again, using the upper bound from the relaxation we expect a reduction in the number of considered labels. But now, the consideration of nodes is independent of the cost and resource functions so we cannot use lower bound costs here. However, we may use them in the propagation step and therefore expect further pruning.

In the following experimental section we compare the practical performance of the different gap-closing methods.

3.5 Experiments

We will perform a detailed experimental comparison of the different proposed methods for single and multiple resource CSP. Since the performance of the algorithms is expected to vary for different network types we will do experiments on random and real world data using the following graph types: grid graphs, road graphs, and curve approximation graphs. We will also vary the "hardness" of the constraint since some methods favour weakly constrained problems whereas they fail on problems with stronger constraints.

All our experiments measure CPU time in seconds on a Sun Enterprise 333MHz, 6Gb RAM running SunOS 5.7 using LEDA 4.3 and our CNOP package[24] compiled with GNUs `g++` version `2.95.2` with optimizing flag `-O`.

3.5.1 The Benchmarks

We use the following three network types as benchmarks:

DEM: Digital elevation models (DEM) are gridgraphs where each node has an associated height value. We use samples of European DEMs[25].

[23] Usually we will discover that most compared labels are identical but keeping track of the "hot" labels is very difficult.

[24] Refer to Chapter 5 for a detailed discussion.

[25] Available from USGS EROS Data Center (global digital elevation model (DEM) with a horizontal grid spacing of 30 arc seconds (approximately 1 kilometer)).

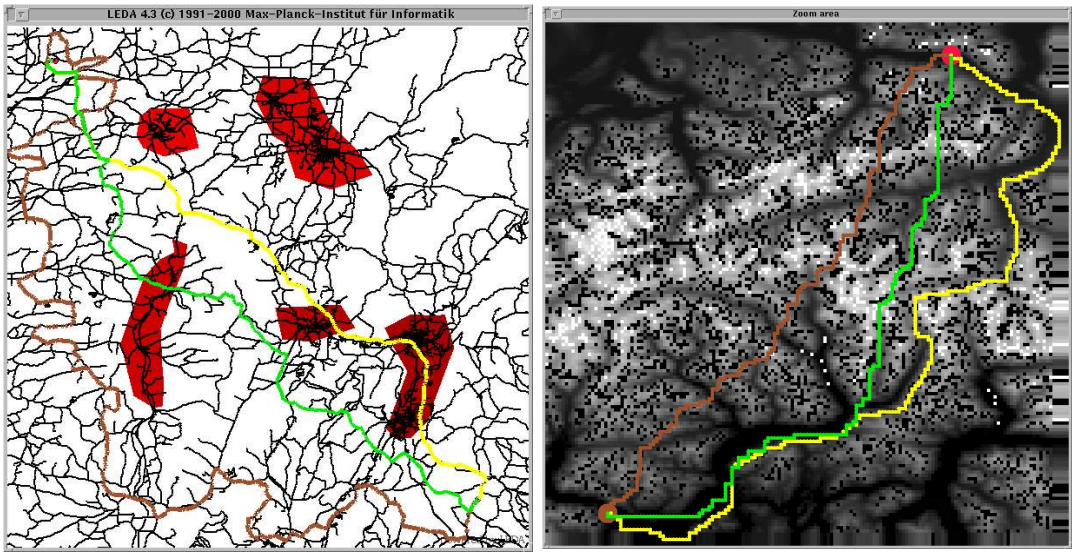

Figure 3.20: Minimum congestion satisfying length constraint (areas of congestion areas are shaded) (left). Minimum total height difference satisfying length constraint (right). The minimum cost paths are yellow, the minimum resource paths brown, and the constrained shortest paths green.

Our DEMs are bidirected, *i.e.*, $m \sim 4n$. We use the absolute value of height differences as the cost function for an edge and get integer edge costs in the interval [0..600]. We use random integers from the range [10..20] as resource of an edge modeling some random length function. We are interested in minimizing the accumulated height difference of a path with a constraint on its total length (see also Figure 3.20).

ROAD: We use US road graphs[26]. Edges modeling roads are again bidirected, and the structure gives us $m \sim 2.5n$. We use congestion as the cost function. We define congested areas with integers in the range [0..100]. We use the euclidean distance between the endpoints of an edge as the resource function. These distances are double precision floating point values in the range [0..7]. We are interested in minimizing congestion subject to a (euclidean) length constraint (see also Figure 3.20).

CURVE: In the curve approximation problem we want to approximate a piecewise linear curve by a new linear curve using fewer breakpoints. This is an important data compression problem in areas like cartography, computer graphics, and signal

[26]Obtained from TIGER/DLG3 data, provided by Jan Vahrenhold.

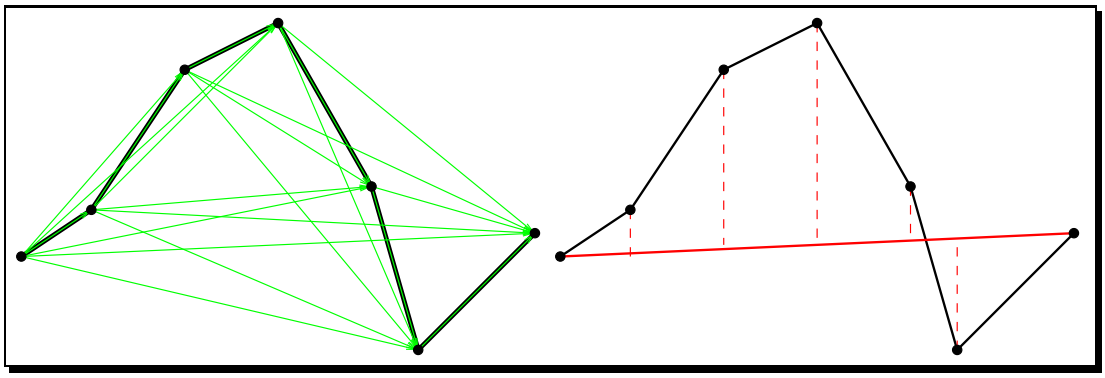

Figure 3.21: Curve approximation graph (left) Approximation error of a shortcut edge (right).

processing. Dahl and Realfsen (2000) and Nygaard (2000) have shown how to model this problem as a constrained shortest path problem:

Assuming that the breakpoints on the given curve occur in the order $v_1, v_2, \ldots, v_n$, we view the breakpoints as nodes and introduce an arc (v_i, v_j) for each $i < j$. Figure 3.21 shows the resulting graph. The cost of an edge is set to the approximation error that is introduced by taking this shortcut instead of the original curve (see Figure 3.21). Note that no further assumption is made concerning the metric generating these weights. For instance, any l_p-norm for $1 \leq p < \infty$ may be used. The resource consumption of an edge is set to 1. Given a limit K on the number of breakpoints, we now have a constrained shortest path problem and may compute the minimum error approximation of the linear curve using at most K breakpoints[27]

We use a random test signal as linear function. We assume that one can reach all 20 subsequent breakpoints from a given node which implies that $m \sim 20n$. We use the l_1-metric to compute approximation errors and get double precision floating point values in the range [0..300] as costs.

We have four examples per network class and use three different resource limits between minimum resource consumption r_{min} and resource consumption r_{max} of the minimum cost path: Strongly constrained (10% off r_{min}), normally constrained (in the middle between r_{min} and r_{max}), and weakly constrained (10% off r_{max}) problems.

[27]Note that this special constrained shortest path problem can be solved in polynomial time $O(n^3)$ with dynamic programming since $K \leq n$.

3.5.2 Single Resource Case

We first experimentally evaluate the performance of the different CSP methods for the single resource case.

Relaxation

We now want to compare different methods for solving the relaxation:

1. Hull approach,
2. Subgradient procedure,
3. Binary search.

We stop the subgradient method if two subsequent multipliers μ differ by less than 0.02 or after the hull approach has reached the optimum of the relaxation to allow for a better comparison. The subgradient parameters regulating the stepsize and the initial upper bound were set as suggested by Beasley and Christofides (1989).
We stop the binary search approach prematurely when it has reached the optimal hull segment[28].
We are interested in the performance of the different relaxation methods as well as in the quality of the obtained bounds.

We first take a look at the DEM benchmarks. Figure 3.22 shows a comparison of the number of iterations needed to solve the relaxation[29]. We see that the hull approach seems to take an almost constant number of iterations: it takes about eight iterations to solve the relaxation exactly. The binary search method uses two to three iterations more to reach the optimal hull segment[30]. The subgradient procedure is terminated after the same number of iterations as in the hull approach.

Let us take a look at the quality of the obtained bounds (see Figure 3.23). We see that both upper and lower bounds obtained by the hull approach are very close to the optimum. The lower bounds are within 1%, the upper bounds within 5% of the optimum. Binary search reaches these bounds with more iterations than the hull approach. The approximate bounds of the subgradient procedure are clearly worse, the lower bounds are within 10% of the optimum and the upper bounds can exceed the optimum by up to 100%.

[28] but not yet verified its optimality.
[29] approximately or exactly depending on the method.
[30] However, without premature break, we would need around 50 iterations until the break criterion is met.

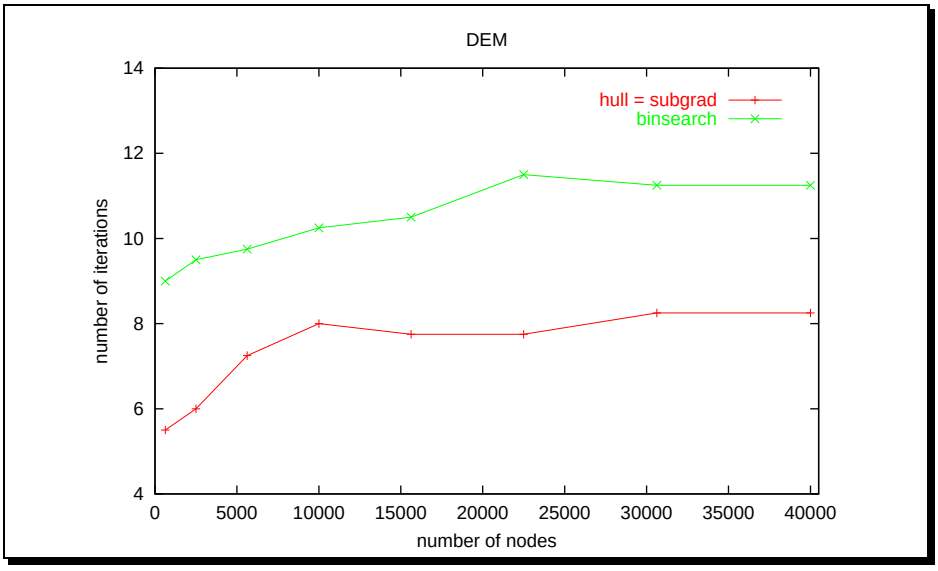

Figure 3.22: Number of iterations of the relaxation methods (DEM).

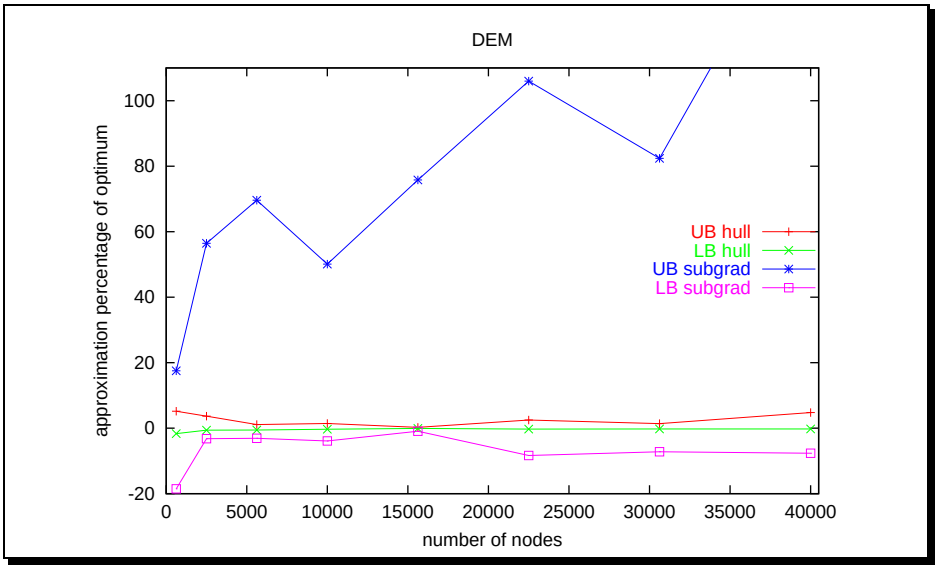

Figure 3.23: Quality of the relaxation bounds (DEM).

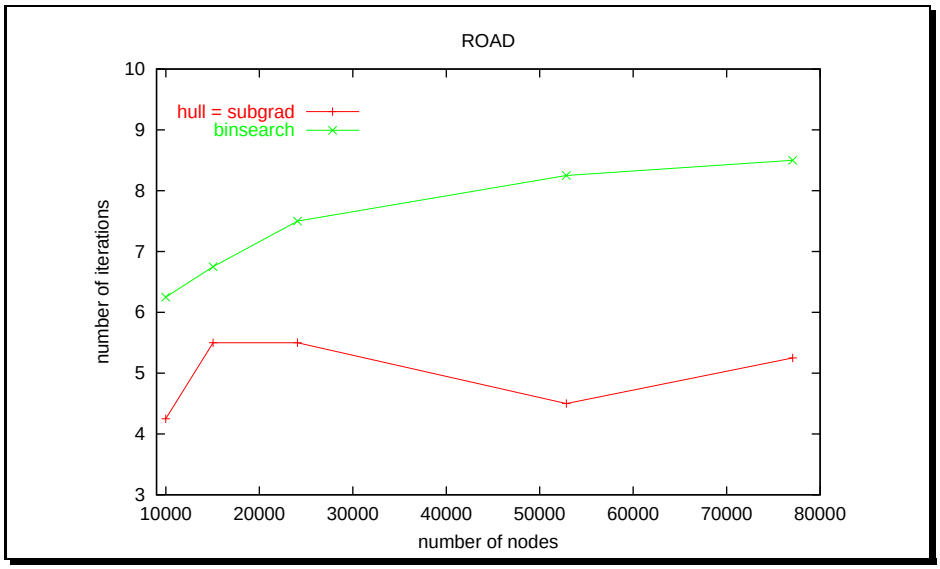

Figure 3.24: Number of iterations of the relaxation methods (ROAD).

Now we take a look at the ROAD benchmarks. Figure 3.24 shows a comparison of the number of iterations needed to solve the relaxation. Again, the number of iterations of the hull approach seems to be constant with never more than six iterations. Binary search reaches the optimal hull segment several iterations after the hull approach, whereas the subgradient procedure is again stopped after the same number of iterations.

Figure 3.25 shows the effect on the quality of the obtained bounds. The lower bounds of the hull approach are within 20% of the optimum, the upper bounds are better and lie within 10% of the optimum. The subgradient procedure fails completely. The bounds are often almost a factor of two away from the optimum which does not improve much on the trivial bounds in that case.

Finally, we take a look at the CURVE benchmarks. Figure 3.27 shows a comparison of the number of iterations needed to solve the relaxation. Here we see a logarithmic behaviour of the number of iterations of the hull approach (as theoretically guaranteed). The reason for this is the special structure of the hull (see Figure 3.26) that contains a larger number of hull points since there is a solution for any resource value between minimum and maximum resource consumption. The binary search approach again uses a few iterations more to reach the optimal hull segment, whereas the subgradient method often reaches the multiplier break in around 11 iterations which is slightly better than the hull approach.

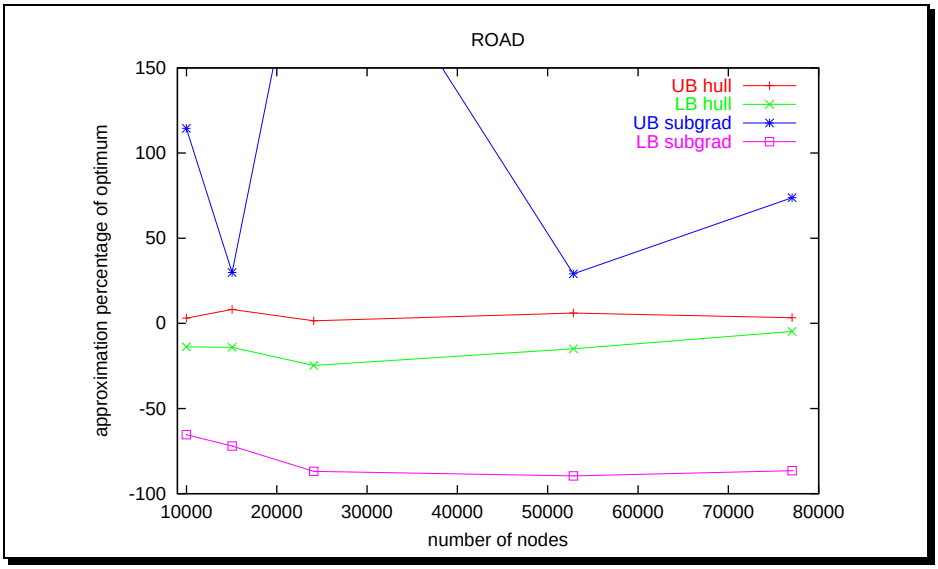

Figure 3.25: Quality of the relaxation bounds (ROAD).

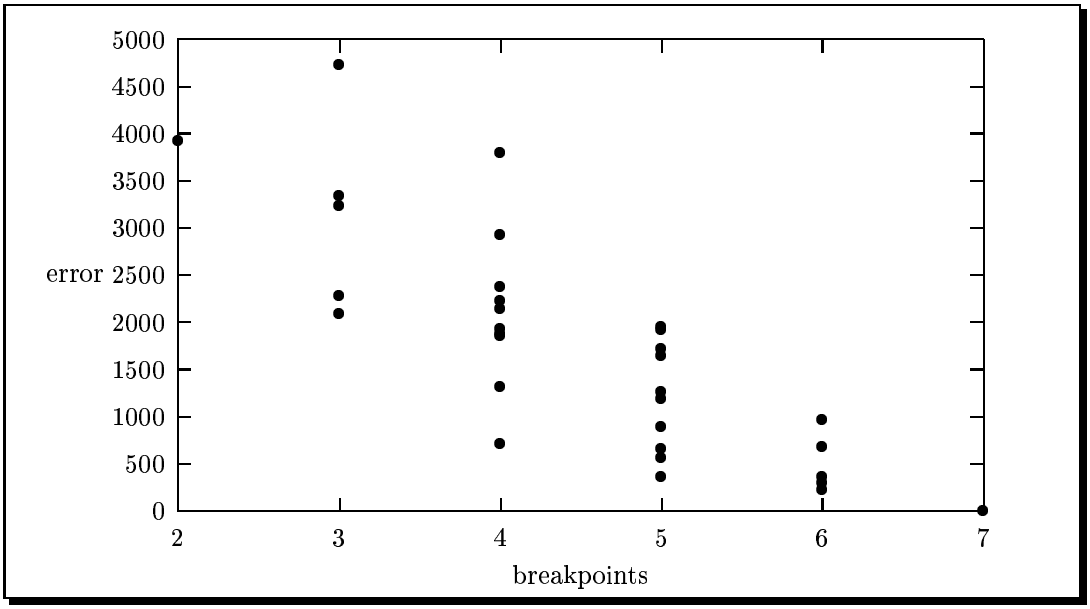

Figure 3.26: Structure of the curve approximation hull for the example in Figure 3.21.

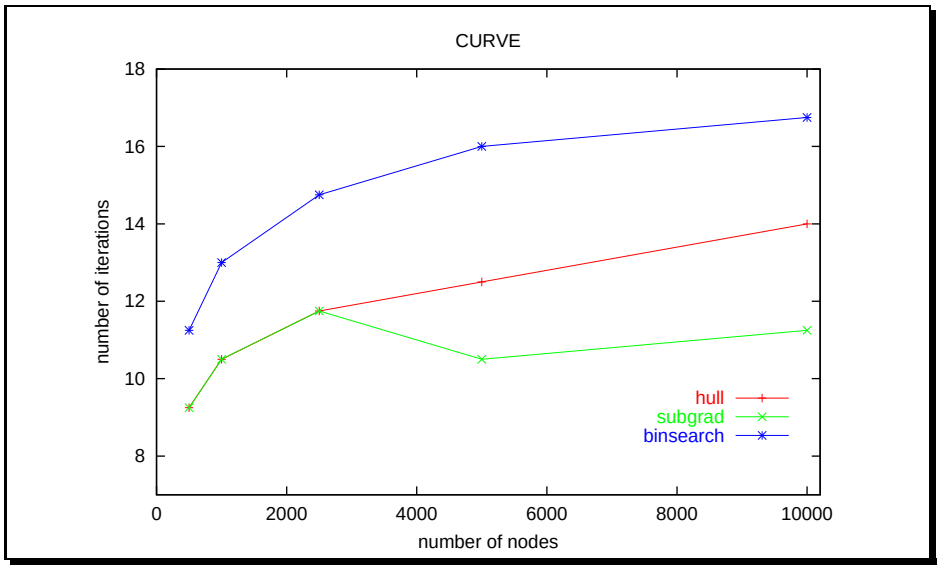

Figure 3.27: Number of iterations of the relaxation methods (CURVE).

Figure 3.28 shows the quality of the obtained bounds. We see that the exact bounds of the hull approach are extremely good. In most cases they are only 0.1% away from the optimum, indeed, we even reach the CSP optimum several times. The approximate lower bounds of the subgradient method are also very good reaching almost the same quality as the exact bounds, but the upper bounds are worse and exceed the optimum by around 10%.

The experiments presented used the normally constrained benchmarks. The benchmarks with strong and weak constraints exhibit similar results.

We conclude that the relaxation bounds are really good in experiments with random and real world data (the higher the number of iterations of the hull approach, the better the bounds), although pathological examples can make them arbitrarily bad. We also see that the number of iterations of the hull approach seems to rise very slowly for increasing graph sizes. The subgradient procedure sometimes requires fewer iterations than the hull approach but one has to pay regarding the bound quality. In the case of road graphs, however, it fails completely. An adapted setting of the stepsize parameters might improve this but tuning parameters for different network structures is usually not what we want. Binary search is never better than the hull approach. Hence, the hull approach should be the method of choice when we are interested in good bounds in efficient time.

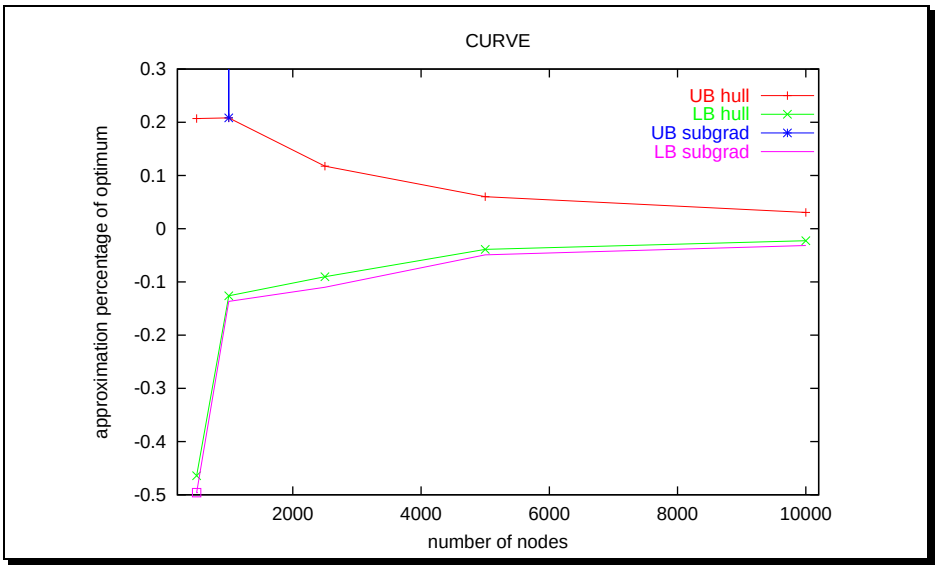

Figure 3.28: Quality of the relaxation bounds (CURVE).

Problem Reductions

Now we investigate the effectivity of the problem reductions using the obtained bounds. Here we examine all three constraint types.

Apparently, the reduction percentage using the resource reductions depends on the constraint type. Only for the CURVE benchmarks (see Table 3.3), there is no resource reduction possible due to the special structure of the underlying graph.

In the DEM case (see Table 3.1) we see that combined resource and cost reductions can reduce the networks to around 5% of their original size if we use the exact bounds

		strong constraints			normal constraints			weak constraints		
n	m	rr	crBC	crMZ	rr	crBC	crMZ	rr	crBC	crMZ
2500	9800	8.17	7.94	2.91	49.36	47.48	8.07	75.37	45.06	14.75
5625	22200	11.29	11.10	3.86	53.50	52.25	3.06	76.06	54.25	4.29
10000	39600	17.76	17.75	4.50	62.88	39.03	1.97	80.94	57.16	2.45
15625	62000	16.98	16.98	2.43	72.33	54.98	1.08	88.61	68.23	12.15
22500	89400	16.63	16.59	1.42	66.05	66.00	7.12	82.61	82.05	7.68
30625	121800	16.94	10.60	0.92	71.05	31.39	2.49	88.41	69.93	1.62
40000	159200	28.76	23.09	5.57	79.67	63.37	9.80	92.60	52.15	1.94

Table 3.1: Pruning reduction percentage after resource reductions (rr) and cost reduction using approximate bounds (crBC) or exact bounds (crMZ) (DEM).

		strong constraints			normal constrained			weak constraints		
n	m	rr	crBC	crMZ	rr	crBC	crMZ	rr	crBC	crMZ
9990	21342	6.01	0.29	0.22	14.85	1.04	0.92	22.52	14.48	10.83
15049	35438	1.94	0.50	0.49	5.32	1.67	0.64	9.50	2.66	0.65
24086	50826	5.72	2.13	2.06	20.33	17.40	4.32	28.05	15.5	12.73
52847	120332	2.51	1.48	0.68	14.04	8.18	6.40	18.05	10.22	5.98
77059	171536	3.58	1.83	0.93	15.46	7.98	3.69	23.70	13.51	12.40

Table 3.2: Pruning reduction percentage after resource reductions (rr) and cost reduction using approximate bounds (crBC) or exact bounds (crMZ) (ROAD).

		strong constraints			normal constraints			weak constraints		
n	m	rr	crBC	crMZ	rr	crBC	crMZ	rr	crBC	crMZ
500	9790	100	100	44.82	100	29.12	16.26	100	29.74	16.67
1000	19790	100	100	22.77	100	17.61	17.60	100	48.19	16.75
2500	49790	100	100	57.21	100	50.90	27.51	100	55.25	14.87
5000	99790	100	100	16.78	100	65.35	27.68	100	49.66	5.48
10000	199790	100	100	40.75	100	100	27.80	100	100	23.41

Table 3.3: Pruning reduction percentage after resource reductions (rr) and cost reduction using approximate bounds (crBC) or exact bounds (crMZ) (CURVE).

regardless of the hardness of the constraint. Using the approximate subgradient bounds in the cost reduction we get almost no additional network reduction.

In the ROAD case (see Table 3.2) we again get a large reduction to around 5% of the original network size. In this case, the reduction effectivity varies with the hardness of the constraint. The subgradient bounds allow a similar reduction (only slightly worse).

In the CURVE case (see Table 3.3) we do not get such impressive reductions as before. We only reach an average of around 25% using the exact bounds. The approximate bounds result in a much worse average reduction percentage of around 60%. Of course, we have to keep in mind that resource reductions were not possible in that case.

We conclude that the problem reductions are most effective using the exact relaxation bounds and that we are often able to reduce the networks considerably. We will see in the following how this improves the gap-closing step[31].

[31] Since our gap-closing methods use online reduction techniques we will not perform the reductions explicitly before the gap-closing step.

Closing the Gap

Now we turn to the gap-closing step. We compare different gap-closing strategies after the computation of the relaxation:

- Path ranking with the k-shortest path algorithm of Jimenez and Marzal (1999)[32] Their algorithm also enumerates paths with loops, which does not cause a problem for the gap-closing step provided that the network contains no zero length cycles with respect to the lower bound costs. If we modify the algorithm to only rank loopless paths we get an additional factor of n in the runtime bound. We switch between both methods to always get the best possible performance for the different network types,
- Label setting method driven by lower bound costs using upper and lower bounds of the relaxation,
- Label setting method driven by original costs using upper and lower bounds of the relaxation,
- Label correcting method using upper and lower bounds of the relaxation.

We compare the gap-closing performance starting from the exact and the approximate solution of the relaxation.

We also compare our 2-step method with other CSP methods:

- ILP solving with CPLEX,
- Dynamic programming,
- Path ranking,
- Label setting,
- Label correcting.

We first turn to the DEM case. Figure 3.29 shows a runtime comparison of different gap-closing methods. We observe that path ranking[33] (`ksp`) and the labeling approach starting from the lower bound costs (`lbc`) are the best methods. We also observe that gap-closing using these two methods is dominated by the time for solving the relaxation. We even get an almost linear behaviour in that case. Gap closing with labeling methods

[32] Although online pruning is also possible when closing the gap with a k-shortest path algorithm we omit it here since it is difficult to integrate in the recursive enumeration scheme.

[33] The original version that does not enforce loopfree paths is faster than the loopfree variant in this case.

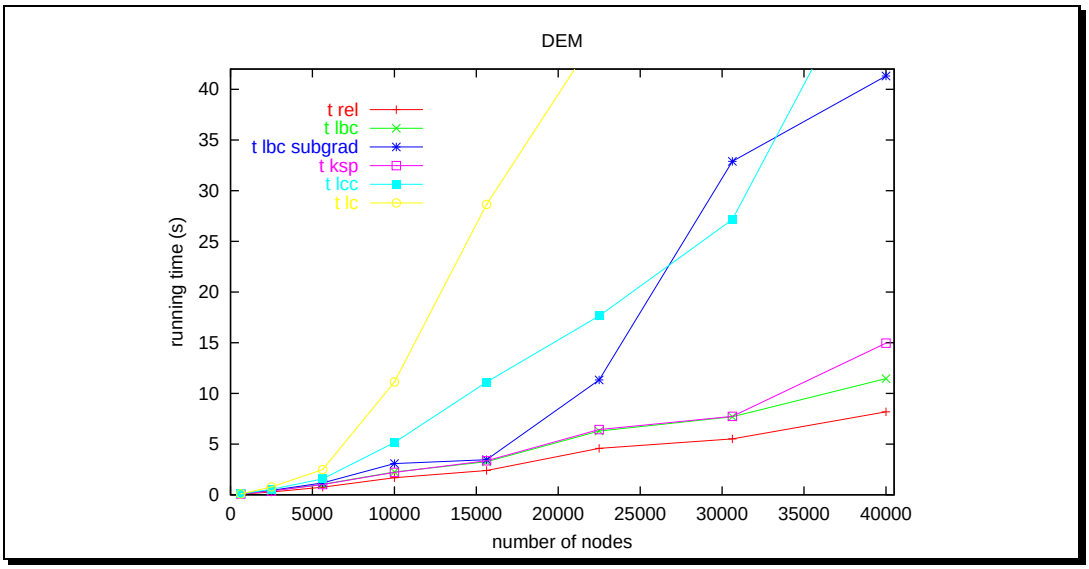

Figure 3.29: Runtime of different gap-closing methods starting from the relaxation optimum and the relaxation approximation (DEM).

using the original cost function (`lcc`, `lc`) is clearly worse (label correcting being superior to label setting).
If we use the approximate bounds from the subgradient procedure and use labeling starting from the corresponding lower bound costs (`lbc subgrad`), we see that the gap-closing step also takes much more time than using the exact bounds of the hull approach.

Figure 3.29 shows the results using the normally constrained benchmarks. The examples with other constraints lead to similar results.

Now we compare the best of the different CSP methods on the DEM benchmarks (see Table 3.4). We see that ILP solving performs worst, followed by dynamic programming which is also not competitive. Naive path ranking in increasing cost order even fails to solve the smallest problems. Labeling methods are competitive for small problems. We observed that the label correcting method is better than the label setting method[34]. The label correcting method even performs best for strongly constrained problems but becomes much slower for the other constraint types[35] where the 2-step method (hull approach then gap-closing with labeling starting from the lower bound costs) clearly performs best. Moreover, the 2-step method shows similar performance on all different constraint types.

[34] As expected, both methods are slower without using the relaxation bounds.
[35] The online pruning is not that effective in these cases.

	strong constraints				normal constraints				weak constraints			
N	2step	label	DP	ILP	2step	label	DP	ILP	2step	label	DP	ILP
625	0.08	0.03	3.15	13.7	0.09	0.04	3.16	13.2	0.09	0.06	3.22	13.9
2500	0.4	0.14	29.7	68.2	0.39	0.49	30.9	67.8	0.41	0.98	31.4	68.5
5625	0.99	0.46	-	-	1.03	2.16	-	-	1.06	4.35	-	-
10000	2.17	1.36	-	-	2.24	8.06	-	-	2.29	17.18	-	-
15625	3.29	2.23	-	-	3.25	17.86	-	-	3.41	41.27	-	-
22500	6.35	3.43	-	-	6.29	26.3	-	-	6.03	60.7	-	-
30625	7.02	4.49	-	-	7.70	42.53	-	-	7.79	104.2	-	-
40000	10.7	10.29	-	-	11.46	98.9	-	-	12.05	241.5	-	-

Table 3.4: Total time in seconds for different CSP methods (DEM). A '-' means that the computation was aborted after 5 minutes.

Now we turn to the ROAD benchmarks. We first have a look at different gap-closing methods starting from the exact bounds (see Figure 3.30). Again, the labeling method starting from the lower bound costs (`lbc`) is overall the best gap-closing method. This time however, the differences are not that pronounced. Also, we again have the situation that gap-closing takes less time than computing the relaxation. Due to the special cost/resource distribution we have to use the loopless path ranking variant (`ksp`), otherwise too many paths with loops are involved in the ranking process. Gap closing starting from the approximate relaxation of the subgradient method (`lbc subgrad`) again takes longer than starting from the exact bounds of the hull approach but the difference is not that significant, as in the DEM case.

Figure 3.30 shows the results using the normally constrained benchmarks. The examples with other constraints lead to similar results.

Now we compare the best of the different CSP methods on the ROAD benchmarks (see Table 3.5). We see that ILP solving and dynamic programming fail to compute the optimum within five minutes even on our smallest road graphs. In the ROAD case, we observe that the label correcting method performs best over all constraint types, although its running time is dependent on them. The running time of the 2-step method is not dependent on the constraint type but is worse by a factor of 1.5 to 3 than labeling. The reason for the excellent performance of labeling in the ROAD case is the special structure of the road networks that have a very small node degree. Moreover, even for weakly constrained road networks the resource reductions allow to reduce the size of the network to at least 30% of the original size, hence the online reductions in the labeling methods are very effective. Using the explicit resource reductions before performing the hull approach also reduces the time for the 2-step method to around the same level as the labeling methods.

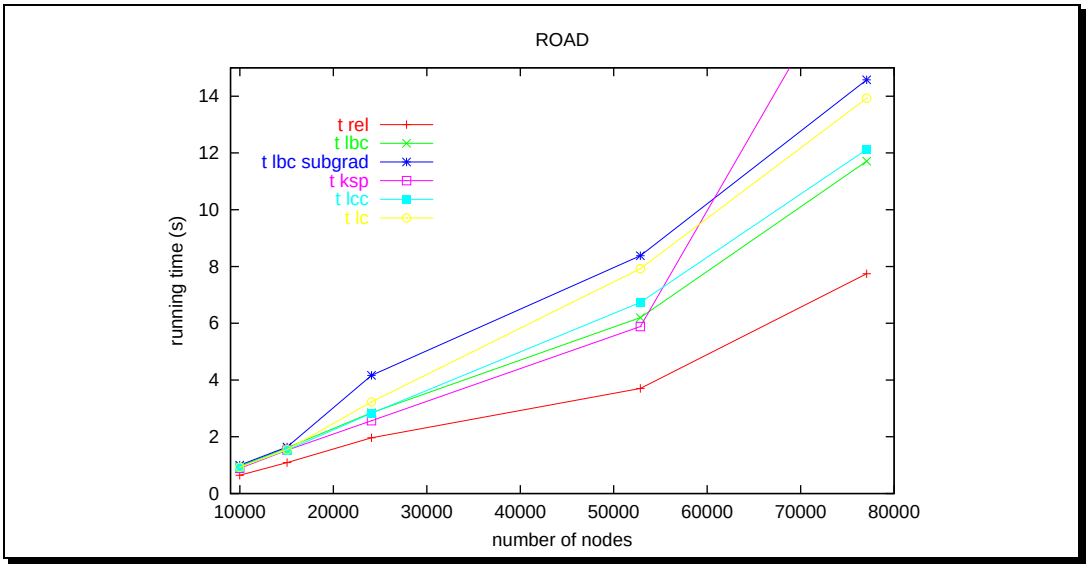

Figure 3.30: Runtime of different gap-closing methods starting from the relaxation optimum and the relaxation approximation (ROAD).

	strong constraints				normal constraints				weak constraints			
N	2step	label	DP	ILP	2step	label	DP	ILP	2step	label	DP	ILP
9990	0.99	0.36	-	-	0.99	0.39	-	-	0.99	0.43	-	-
15049	1.6	0.58	-	-	1.64	0.60	-	-	1.61	0.66	-	-
24086	2.93	1.01	-	-	2.92	1.3	-	-	3.05	1.54	-	-
52847	6.78	2.61	-	-	6.33	3.66	-	-	7.09	6.04	-	-
77059	12.03	4.34	-	-	12.02	6.58	-	-	13.19	10.77	-	-

Table 3.5: Total time in seconds for different CSP methods (ROAD). A '-' means that the computation was aborted after 5 minutes.

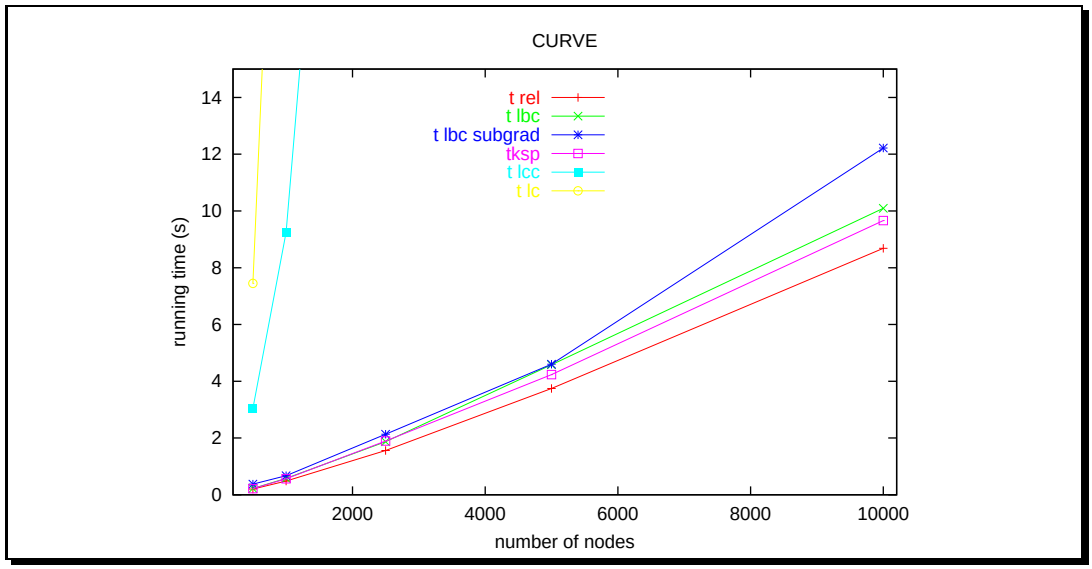

Figure 3.31: Runtime of different gap-closing methods starting from the relaxation optimum and the relaxation approximation (CURVE).

We finally turn to the CURVE benchmarks. Figure 3.31 shows the performance of different gap-closing methods starting from the exact relaxation bounds. Again path ranking[36] (`ksp`) and label setting starting from the lower bound costs (`lbc`) perform best and take only very little time compared to solving the relaxation. Label correcting (`lcc`) and normal label setting (`lc`) are extremely bad despite the good bounds which can be explained by the special structure of the graph that does not allow very effective online pruning. Starting from the approximate relaxation of the subgradient method (`lbc subgrad`) gap-closing again takes longer than starting from the exact bounds of the hull approach but is also dominated by the time for solving the relaxation.

Figure 3.31 shows the results using the normally constrained benchmarks. The examples with other constraints lead to similar results.

Now we compare the best of the different CSP methods on the CURVE benchmarks (see Table 3.6). It is interesting to observe that dynamic programming is competitive to labeling in the case of CURVE graphs. The reason for this effect is that we now have a possible solution for each resource value, so the "naive" method of considering each resource value for a better solution is promising here. However, the best method clearly is again the 2-step method[37].

[36] The original version that does not enforce loopfree paths is faster than the loopfree variant in this case.

[37] Something interesting happens when we choose a fractional resource limit. Due to the special structure of the problem (see Figure 3.26) it is possible that we have to rank a lot of paths until we

	strong constraints				normal constraints				weak constraints			
N	2step	label	DP	ILP	2step	label	DP	ILP	2step	label	DP	ILP
500	0.22	0.85	0.49	9.7	0.22	3.25	0.98	9.8	0.24	4.88	1.49	9.9
1000	0.30	2.97	1.95	26.7	0.57	10.31	4.16	26.5	0.65	17.44	6.17	26.8
2500	1.59	12.42	14.9	162	1.87	59.2	30.8	163	1.93	101.2	46.4	162
5000	2.67	44.7	65.3	-	4.51	225.1	136.0	-	4.09	-	209.8	-
10000	7.14	-	-	-	10.09	-	-	-	11.04	-	-	-

Table 3.6: Total time in seconds for different CSP methods (CURVE). A '-' means that the computation was aborted after 5 minutes.

Conclusion: We have tested different CSP methods on three benchmark network types using three different constraint types. We observed that the bounds obtained by solving the relaxation exactly with the hull approach are very good. The approximate bounds of the subgradient method are sometimes close to the exact bounds but often clearly worse.

The good bounds allow effective problem reductions that are implicit in all gap-closing methods. Label setting starting from the exact lower bound costs is the best overall gap-closing method in the 2-step approach followed by path ranking. For all benchmarks we observed that this gap-closing method is dominated by solving the relaxation. Moreover, the running time does not seem to depend on the hardness of the resource constraint. Using the approximate bounds for gap-closing always results in a higher running time.

Dynamic programming and ILP solving are never competitive with the 2-step method. Only labeling is competitive[38]. It is always a good choice in small and hard constrained problems but suffers on other types, except in the case of ROAD graphs. Hence the 2-step method that first performs the hull approach and then closes the gap with label setting from the lower bound costs is a safe choice for all settings.

3.5.3 Multiple Resource Case

Now we experimentally evaluate the performance of CSP methods for the multiple resource case. Due to the increasing number of parameters that may affect the running time, we only concentrate on random grid graphs with costs and resources in the interval $[10, 50]$.

have swept the necessary region to guarantee optimality. In the case of an integral resource limit, we almost immediately hit a path with this resource consumption that constitutes the optimum.

[38] We found out that label correcting usually outperforms label setting methods.

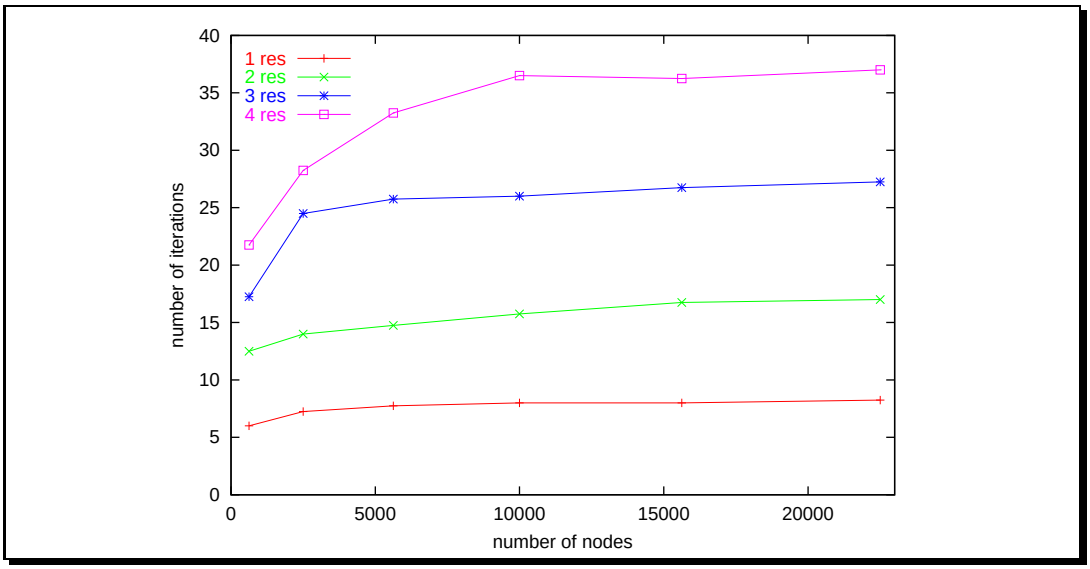

Figure 3.32: Iterations of the hull approach for multiple resources.

Relaxation

In the multiple resource case, we have only two relaxation methods, the hull approach and the subgradient procedure. We are again interested in the number of iterations and the quality of the bounds obtained from the relaxation.

Contrary to the single resource case, we could not prove a polynomial running time for our hull approach but Figure 3.32 shows that the number of iterations seems to be k times the number of iterations of the single resource case which supports our Conjecture 1 in Section 3.3.4.

Figure 3.33 shows the quality of the obtained bounds. Contrary to the single resource case, we have no guarantee of a feasible solution[39]. However, the lower bounds are again very close to the optimum with the subgradient bounds being slightly worse than the exact hull approach bounds.

Problem Reductions

Since we only get upper bounds in some cases, problem reductions are not as effective as in the single resource case.

[39] The observation is that the hull approach returns more feasible solutions of better quality than the subgradient procedure.

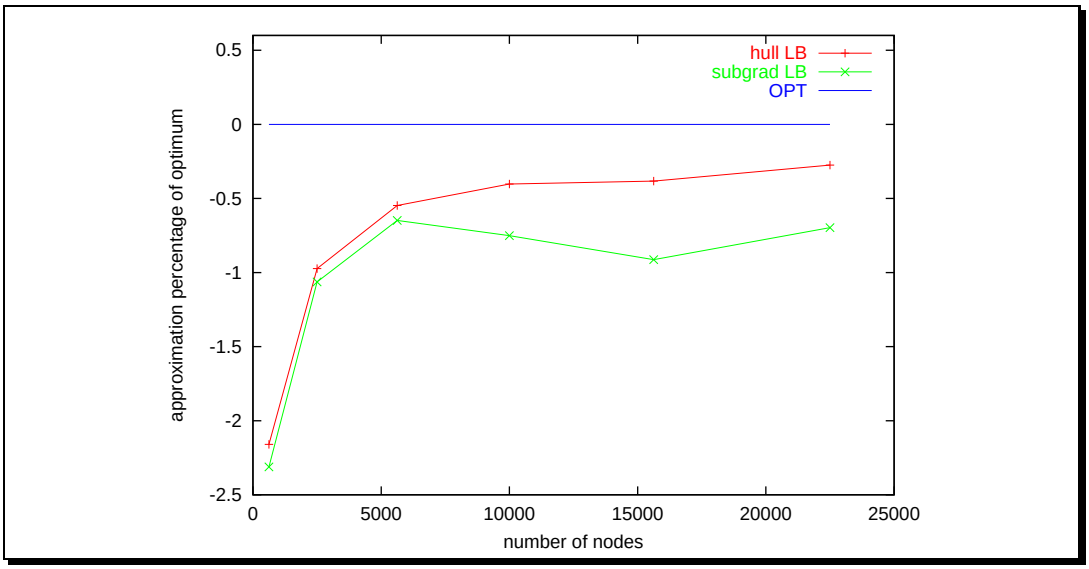

Figure 3.33: Quality of the relaxation bounds for two resources.

Closing the Gap

Now we want to compare the running times of the 2-step method with different gap-closing strategies starting from the exact and the approximate bounds (see Figure 3.34). The exact bounds lead to a faster gap-closing step, as observed before. The labeling method starting from the lower bounds (`lbc`) is the best overall gap-closing approach but we now observe that gap-closing may dominate the running time since pruning is no longer as effective.

This can also be observed in Figure 3.35 where we compare two labeling methods. The performance deteriorates and is now clearly worse than the 2-step method. The reason for this effect is that the nice structure of the label lists in the single resource case is no longer possible, there are fewer dominated labels, thus more labels to consider.

Other methods like multiple resource dynamic programming or ILP solving are also not competitive with the 2-step method.

Conclusion: We have seen that the multiple resource case is harder than the single resource case. The performance of labeling methods breaks down since the pruning is no longer effective due to fewer dominated paths. The 2-step method performs best, although the number of iterations for the relaxation increases and although the gap-closing step sometimes dominates the overall running time.

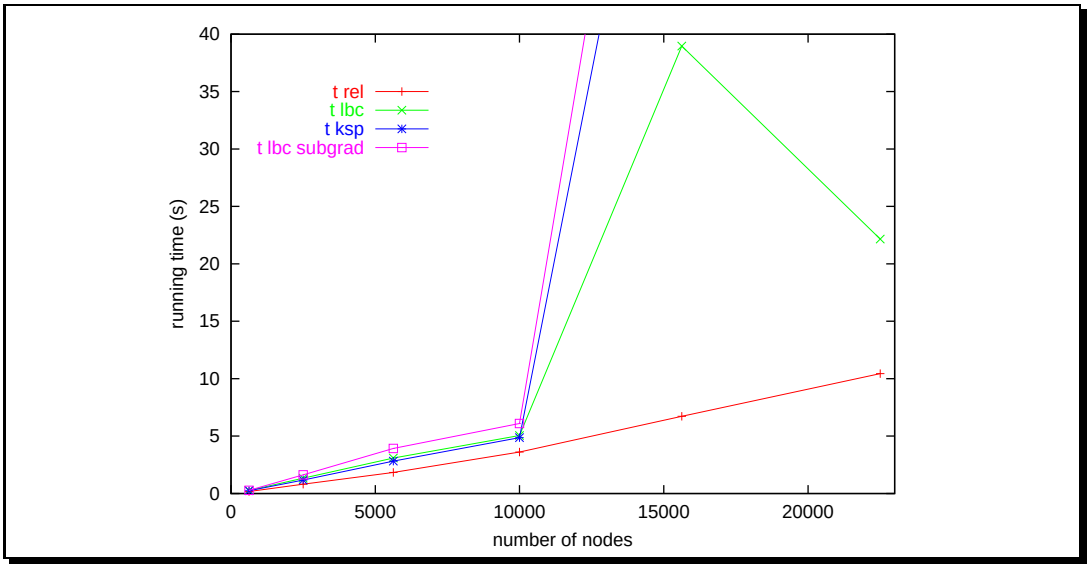

Figure 3.34: 2-step method for two resources.

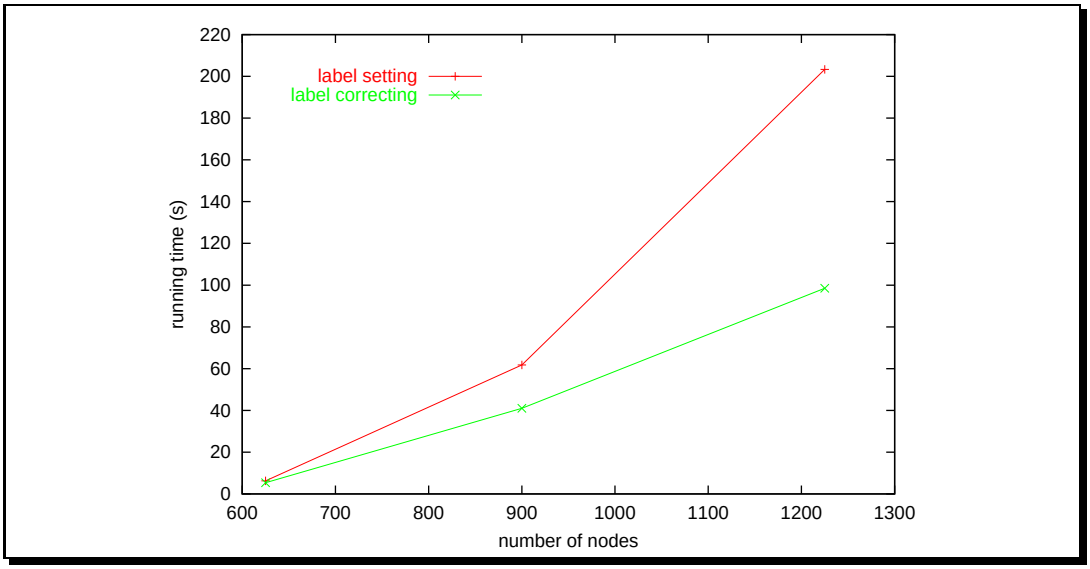

Figure 3.35: Labeling methods for two resources.

3.6 Problem Variants

There are many variants of the CSP problem:

Instead of having an upper limit on the resource consumption, we might also have lower bounds on the resource consumption (see Elimam and Kohler (1997) for an application). Saigal (1968) (see also Roessl (1968)) have considered the special case of finding a minimum cost path containing exactly q arcs. The 2-step method can be adapted to handle this situation by using a modified function testing for feasibility.

If we omit the resource constraint(s) and want to minimize both cost and resource(s), we are dealing with bi- or multicriteria shortest path problems. Here we are interested in the so-called Pareto-optimal paths, *i.e.*, paths that are not dominated by any other path. Work on the bicriteria shortest path problem was done by (Hansen 1980; Henig 1985; Brumbaugh-Smith and Shier 1989; Mote, Murthy, and Olson 1991; Müller-Hannemann and Weihe 2001), the multicriteria version was considered by (Martins 1984; Warburton 1987; Tung and Chew 1992; Modesti and Sciomachen 1998). Henig (1985) and Modesti and Sciomachen (1998) also consider special utility measures for Pareto-optima in bicriteria and multicriteria shortest path problems, respectively.

Hansen (1980) also discusses the interesting case in which the cost and/or resource function is not additive along paths but the minimum or maximum of the edge costs and resources. They examine all possible combinations. The 2-step method relies on the efficient computation of parametric shortest path problems. This is not so easy in these cases, as simple scaling usually does not do the trick.

Another well-known problem variant is the so-called shortest path problem with time windows (SPPTW) in which we are looking for a shortest path with the side condition that for each node visited by the path, the time required to reach it is contained in the time window of the specific node. Hence, CSP is a special case of SPPTW where there only is a time window on the target node. Labeling approaches can be easily adapted to solve SPPTW (see Desrosiers, Dumas, Solomon, and Soumis (1995) for an overview of existing methods). The SPPTW problem has many important applications: It appears as a subproblem in column generation or Lagrangean relaxation approaches used to solve vehicle routing problems, fleet planning problems and crew scheduling problems. It is not clear whether a 2-step method using an adapted labeling approach for the gap-closing step improves on the proposed labeling methods of Desrosiers, Dumas, Solomon, and Soumis (1995).

In the CSP problem we examined paths from s to t. The single source variant could also be of interest, in which we want to have the CSP from s to every other node v. The dynamic programming method solves this problem and the labeling methods can

also be adapted to deal with this variant sacrificing pruning possibilities. The 2-step method does not seem adaptable in a reasonable way to handle multiple targets at once.

3.7 Conclusion

In this chapter we have reviewed state-of-the-art methods and proposed new methods for the constrained shortest path problem. Starting from a new ILP formulation we derived a combinatorial method for solving the relaxation of CSP by combining simple geometric intuition with optimization theory. We obtained the first exact combinatorial approach for solving the multiple resource relaxation. In the single resource case our approach is equivalent to previously proposed methods, however, we were the first to prove a tight polynomial runtime bound of $O(\log(nRC)(n \log n + m))$ using only geometric arguments. Although we could show that the obtained upper and lower bounds do not come with a constant approximation guarantee we have seen in our experiments that the bounds are very good.

To obtain the optimal solution of CSP we have to apply a gap-closing step that we again visualized with a simple geometric interpretation. We reviewed existing gap-closing approaches and showed how to modify the label setting method to obtain an efficient new gap-closing method.

The experimental comparison of the new methods with existing state of the art methods showed that the best 2-step method (solving the relaxation with the hull approach and closing the gap with label setting starting from the lower bound costs) is a competitive method for all considered network classes and constraint types, in some cases it is even superior to all other methods. Only the label correcting approach is faster in the case of road graphs and sometimes for other hard constrained small networks.

Chapter 4

Constrained Network Optimization

In this chapter we consider the general form of resource constrained network optimization problems. CSP is a special case of this problem class but it turns out that the 2-step method for CSP also extends to the general case.

We will first give a definition of constrained network optimization problems and show how the 2-step method easily extends to this broader problem class.

Then we will discuss examples of constrained network optimization problems: constrained minimum spanning trees, table layout, and constrained geodesic shortest paths.

4.1 Constrained Network Optimization Problems

Suppose we are given a network G with n nodes and m edges and a cost function $c : E \to \mathbb{R}^+$ defined on the edges.

Many linear network optimization problems like shortest paths, minimum spanning trees or minimum cuts ask for the computation of a list of edges satisfying problem specific structural constraints (to form a path, a spanning tree, etc.) so that the sum of the edge costs is minimized.

Using 0-1-variables z_e for each edge in the graph, this can be written as an integer

linear program (ILP) as follows:

$$\begin{aligned} \min \quad & \sum_{e \in E} c_e z_e \\ \text{s.t.} \quad & z \in S \\ & z_e \in \{0,1\} \end{aligned}$$

where $z \in S$ abbreviates the specific structural constraints (path, spanning tree, etc.).

For many network optimization problems of this form efficient algorithms for solving the problem in polynomial time $O(p(n,m))$ exist (like in the case of shortest paths, minimum spanning trees, etc.).

If we also have k resource functions $r^{(i)} : E \to \mathbb{R}^+$ defined on the edges and impose the additional constraints that the resource consumptions of our optimal solution should not exceed given resource limits $\lambda^{(i)}$ for $i = 1, \ldots, k$, we get a (resource) constrained network optimization problem of the form:

$$\begin{aligned} \min \quad & \sum_{e \in E} c_e z_e \\ \text{s.t.} \quad & z \in S \\ & \sum_{e \in E} r_e^{(i)} z_e \leq \lambda^{(i)} \qquad \forall i = 1, \ldots, k \\ & z_e \in \{0,1\} \end{aligned}$$

Even one additional resource constraint usually makes the problem $\mathcal{NP}$-hard as it is for example the case for constrained shortest paths or constrained minimum spanning trees (see Garey and Johnson (1979)).

Constrained network optimization problems are of special interest in practical applications since there we often have to deal with bi- or multicriteria optimization problems in networks that are precised by budget- or resource constraints.

We have seen a 2-step method for CSP, a special constrained network optimization problem, in the previous chapter. It turns out that this method can easily be reformulated for the general case.

4.1.1 The General Hull Approach

As solving the ILP for constrained network optimization is too hard, we again first aim for the Lagrangean relaxation that is obtained by turning the complicating resource constraints into the objective function.

We proceed the same way as in the previous chapter, that is, we turn the cost/resource independent structural problem constraints $z \in S$ on the 0-1-variables z_e into newly defined variables: We introduce integer variables x_{sol} for each possible solution (*e.g.*, path, spanning tree, etc.) of our problem. So our problem can be stated as:

$$\begin{aligned} \min \quad & \sum_{sol} c_{sol} x_{sol} \\ \text{s.t.} \quad & \sum_{sol} x_{sol} = 1 \\ & \sum_{sol} r^{(i)}_{sol} x_{sol} \leq \lambda^{(i)} \qquad i = 1, \ldots, k \\ & x_{sol} \in \{0, 1\} \end{aligned}$$

Dropping the integrality constraint we can set up the corresponding dual problem (D):

$$\begin{aligned} \max \quad & u + \sum_{i=1}^{k} v_i \lambda^{(i)} \\ \text{s.t.} \quad & u + \sum_{i=1}^{k} v_i r^{(i)}_{sol} \leq c_{sol} \quad \forall sol \\ & v_i \leq 0 \qquad i = 1, \ldots, k \end{aligned}$$

The dual program (D) has $k + 1$ variables, an unconstrained variable u and k non-positive variables v_i for $i = 1, \ldots, k$, and a constraint for each solution satisfying the structural properties. Hence, the number of constraints can get exponentially large. The dual program is again equivalent to the Lagrangean Dual of the constrained network optimization problem.

The only difference from the previously studied CSP relaxation is that we now have constraints for each solution obeying certain structural properties instead of paths. However, the constraints can still be interpreted as halfspace equations, and a solution as a point in the cost-resource-space (see Figure 4.1 for the single resource interpretations).

The method for solving the CSP relaxation that we described in the previous chapter was based on the fact that we could solve the separation problem efficiently with scaled-cost shortest path computations. Now, in the general case, the separation problem is to ask whether there is a solution sol' so that

$$c_{sol'} - \sum_{i=1}^{k} v_i r^{(i)}_{sol'} < u$$

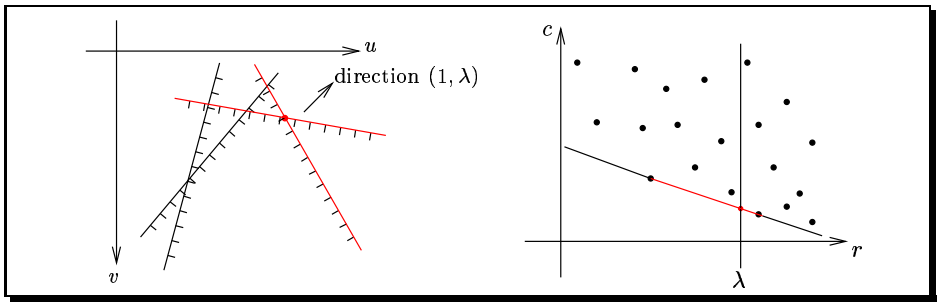

Figure 4.1: (Left) Find the maximal point in direction $(1, \lambda)$ in the halfspace intersection. (Right) Find line with maximal c-value at $r = \lambda$ which has all points above or on it.

for given values of u and $v_i \leq 0$. Such a solution can be found by solving the unconstrained network optimization problem with a scaled cost function $w_e = c_e - \sum_i v_i r_e^{(i)}$ on the edges.

If we assume that the underlying (unconstrained) network optimization problem for a given cost function can be solved in time $O(p(n, m))$, where p is a polynomial in the input, then we get the following lemma.

Lemma 4.1.1: The Lagrangean relaxation of the constrained network optimization problem can be solved in polynomial time.

Proof. The problem (D) is equivalent to the Lagrangean relaxation. We assumed that the unconstrained network optimization problem can be solved in polynomial time, hence also the separation problem is polynomially solvable. Thus, we can solve the Lagrangean relaxation in polynomial time with the Ellipsoid method. □

The preceding lemma guarantees that the multiple resource relaxation can be solved polynomially with the Ellipsoid method. Of course, we are interested in a special combinatorial method as in the CSP case. Since solving the separation problem was the only problem-dependent step in the proposed dual simplex method with cutting plane generation and its geometric interpretation, the hull approach, we may also use this method in the general case of constrained network optimization.

The iteration bound of the single resource hull approach also extends since the proof involved only problem independent geometric arguments. Hence, assuming that a desired network solution consists of at most $O(m)$ edges and assuming $m = O(n^2)$, we have the following general theorem.

Theorem 4.1.1: The general hull approach computes the Lagrangean relaxation of a network optimization problem with a single resource constraint in $O(\log(nRC)p(n, m))$

time if the corresponding unconstrained problem can be solved in time $O(p(n,m))$ and the maximal cost and resource of an edge is C and R, respectively.

Proof. Assuming that a desired network solution consists of at most $O(m)$ edges and assuming $m = O(n^2)$ we conclude that a solution of maximal cost and resource is bounded by $O(mC)$ and $O(mR)$, respectively. The rest of the proof uses only geometric arguments, hence the iteration bound follows. □

We obtain the same asymptotic bound for the general binary search approach:

Lemma 4.1.2: The general binary search approach computes the Lagrangean relaxation of a network optimization problem with a single resource constraint in $O(\log(nRC)p(n,m))$ time.

The other geometric result concerned with the computation of the whole lower hull containing the extreme Pareto-optimal paths also extends:

Lemma 4.1.3: The whole lower hull containing the extreme Pareto-optimal solutions can be computed in $O((m \max\{C,R\})^{2/3})$ iterations using the modified hull approach.

4.1.2 Closing the Gap with Solution Ranking

Solving the relaxation with the hull approach, we obtain a lower bound on the optimal solution. We also record the value of the best feasible solution that provides an upper bound[1]. A duality gap may exist since the optimal hyperplane found by the hull approach together with the resource limit planes and the upper bound plane define an area where the true optimal solution may lie (see Figure 4.2 for an illustration of the single resource case). In the CSP case, we have closed this gap using a path-ranking procedure that enumerates paths in nondecreasing order with respect to the optimal reduced costs. Sweeping the area where the true optimal solution may lie with this procedure and updating bounds along the way, we may determine the optimum.

We can again adopt the same geometric interpretation in our general constrained network optimization case. Now, we want to rank general network solutions with respect to the optimal reduced costs[2].

Lawler (1972) gave a general method for ranking discrete optimization problems that we will explain in the following.

[1] A feasible solution (provided the problem is feasible) is only guaranteed in the single resource case.

[2] The hope is again that the area (or volume) that has to be swept until finding the optimum (being an indicator for the number of solutions to be ranked) does not get too large.

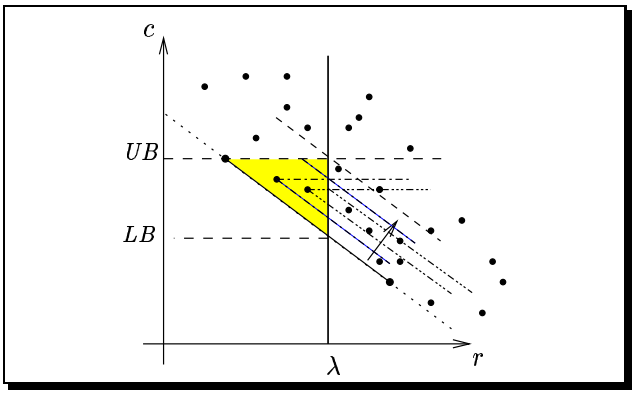

Figure 4.2: Closing the gap between upper and lower bound.

The Method of Lawler: We assume that the problem is one of minimization in a network and that the solutions are expressed in terms of 0-1-variables $x_1, x_2, \ldots, x_m$ on the edges. We also assume that an efficient procedure determining optimal solutions exists, subject to the condition that certain of these variables are assigned fixed values. If no feasible solution exists for certain fixed values of the variables, the value of an optimal solution is taken to be $+\infty$.

The following simple procedure ranks solutions from the first to the K-th, for predetermined K:

1. (*Start*) Compute an optimal solution without fixing the values of any variables, and place this solution in LIST as the only entry. Set $k = 1$.

2. (*Output k-th solution*) Remove the least costly solution from LIST and output this solution, denoted $x^{(k)} = (x_1^{(k)}, \ldots, x_m^{(k)})$, as the k-th solution.

3. (*Test k*) If $k = K$, stop.

4. (*Augmentation of LIST*) Suppose, without loss of generality, that $x^{(k)}$ was obtained by fixing the values of $x_1, x_2, \ldots, x_s$. Leaving these variables fixed as they are, create $m - s$ new problems by fixing the remaining variables as follows:

$$
\begin{array}{lrcl}
(1) & x_{s+1} & = & 1 - x_{s+1}^{(k)}, \\
(2) & x_{s+1} & = & x_{s+1}^{(k)}, x_{s+2} = 1 - x_{s+2}^{(k)}, \\
(3) & x_{s+1} & = & x_{s+1}^{(k)}, x_{s+2} = x_{s+2}^{(k)}, x_{s+3} = 1 - x_{s+3}^{(k)} \\
\vdots & & & \vdots \\
(m-s) & x_{s+1} & = & x_{s+1}^{(k)}, x_{s+2} = x_{s+2}^{(k)}, \ldots, x_{m-1} = x_{m-1}^{(k)}, x_m = 1 - x_m^{(k)}.
\end{array}
$$

Compute optimal solutions to each of these $m - s$ problems and place them in

LIST, together with a record of the variables which were fixed for each of them. Set $k = k + 1$. Return to Step 2.

The key to this procedure is the "branching" operation performed in Step 4. Let X denote the set of feasible solutions for the problem for which $x^{(k)}$ is optimal, and let $X^{(1)}, X^{(2)}, \ldots, X^{(m-s)}$ denote the sets of feasible solutions for problems $(1), (2), \ldots, (m-s)$ created in Step 4. It is easy to verify that $X^{(1)} \cup X^{(2)} \cup \cdots \cup X^{(m-s)} = X - \{x^{(k)}\}$. That is, the branching operation excludes only $x^{(k)}$ from further consideration.

If we again assume that the time to compute a single optimization problem is $O(p(n,m))$, then we can conclude that we may rank the K best solutions in time $O(Kmp(n,m))$. The space is bounded by $O(Km)$ but this bound may be reduced to $O(K+m)$ while only doubling the computation time (see Lawler (1972)).

We can use this general method to close the duality gap in our constrained network optimization problems. Let LB be the optimum of the Lagrangean relaxation, w^* the reduced costs leading to the relaxation optimum (*i.e.*, $w_e^* = c_e - \sum_i v_i^* r_e^{(i)}$), and UB the cost of the best-known feasible solution. We use the optimal reduced costs w^* as cost function and start solution ranking updating upper and lower bounds along the way until $w^*(sol) + \sum_i v_i^* \lambda^{(i)} > UB$. The following "online" pruning is possible: We may discard a new problem in Step 4 if the resource consumption of the fixed chosen edges[3] exceeds a resource limit or if the cost of the fixed chosen edges[4] exceeds UB. We also only insert a new solution in LIST if $w^*(sol) + \sum_i v_i^* \lambda^{(i)} \leq UB$.

It should be noted however, that despite the simplicity and genericity of Lawler's method, it is often possible to find more efficient ranking algorithms for specific problems, as it is for example the case in the k-shortest path or the k-minimum spanning tree problem.

Now we have a generic 2-step method for solving constrained network optimization problems for single or multiple resources:

1. Compute the Lagrangean relaxation with the hull approach solving unconstrained network optimization problems in each iteration.

2. Close the duality gap with Lawler's solution ranking method starting from the lower bound.

[3] together with the minimal resource completion of the solution
[4] together with the minimal cost completion of the solution

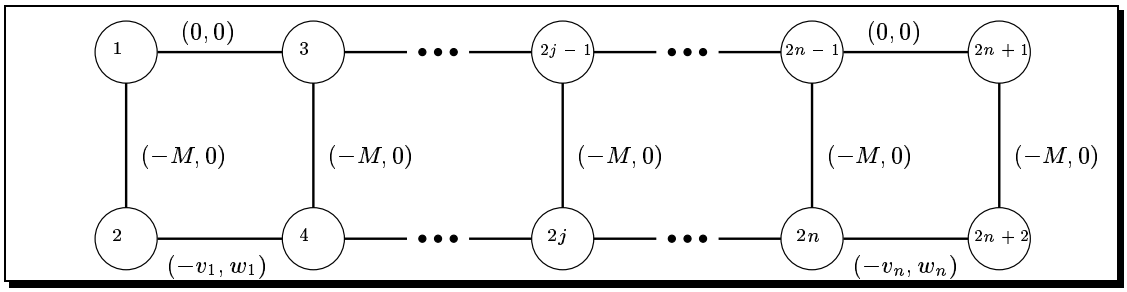

Figure 4.3: Network for the knapsack problem.

4.2 Constrained Minimum Spanning Trees

We now consider another constrained network optimization problem, the *constrained minimum spanning tree problem (CMST)*. Again, each edge has a cost value and k resource values and we are given limits on the resource consumption of a spanning tree. The goal is to find a spanning tree of minimal cost that satisfies the resource limits.

Previous work on this problem concentrating on the single resource case has been done by Aggarwal, Aneja, and Nair (1982) and Hamacher and Ruhe (1994) who solved the relaxation with a hull approach variant and closed the duality gap by branch and discard, and local search, respectively. Ravi and Goemans (1996) presented a PTAS for the problem and Xue (2000) solved the relaxation with a combination of binary search and hull approach.

Using the general 2-step approach of the previous section we get a simple algorithm for multiple resource CMST.

4.2.1 Complexity

The unconstrained minimum spanning tree problem is efficiently solvable in polynomial time, for example with Kruskal's or Prim's algorithm. Again the introduction of even a single additional resource constraint changes the complexity of the problem.

Theorem 4.2.1 (Aggarwal, Aneja, and Nair (1982)): CMST is $\mathcal{NP}$-complete.

Proof. We polynomially transform the knapsack problem to CMST. Consider the fol-

lowing knapsack problem:

$$\begin{aligned} \max \quad & \sum_{j=1}^{n} v_j x_j \\ \text{s.t.} \quad & \sum_{j=1}^{n} w_j x_j \leq W \\ & x_j \in \{0,1\} \quad j = 1, \ldots, n \end{aligned}$$

Now we construct a network with $2n+2$ nodes as shown in Figure 4.3, where each edge e in the network carries two numbers corresponding to c_e and r_e, M is a large positive real number, and W corresponds to λ.

Clearly, the knapsack problem may be solved by finding a minimum cost spanning tree in the constructed network with resource consumption $\sum_e r_e \leq \lambda$. The lower horizontal edges which are in this tree would reveal the variables at level 1 in the optimal solution to the knapsack problem. Since the knapsack problem is $\mathcal{NP}$-complete, the theorem follows. □

4.2.2 Solving the Relaxation

It is easy to see that CMST has the desired properties of a constrained network optimization problem as defined in Section 4.1. Hence we can use a 2-step method to solve CMST. The hull approach now has scaled cost minimum spanning tree computations as a subproblem.

Let us now consider the Lagrangean relaxation of single resource CMST. The bound on the number of iterations of the hull approach extends if we again assume integral costs and resources in $[0..C]$ and $[0..R]$, respectively.

Corollary 4.2.1: The Lagrangean relaxation of single resource CMST can be solved in $O(\log(nRC))$ iterations with the hull approach. This gives a total running time of $O(\log(nRC)(m + n \log n))$.

The bound $O((n \max\{R, C\})^{2/3})$ on the maximum number of points on the lower hull (corresponding to extreme Pareto-optimal spanning trees) also extends.

However, the bicriteria minimum spanning tree problem also allows a strongly polynomial bound, hence the relaxation can also be solved in strongly polynomial time. The reason is that a minimum spanning tree is defined by the order of the edges of the network.

Lemma 4.2.1: In the parametric minimum spanning tree problem with cost function $\tilde{c}_e = c_e + \mu r_e$, as we vary μ from $-\infty$ to ∞, we get at most m^2 minimum spanning trees and every two consecutive minimum spanning trees are adjacent.

Proof. We call an ordered pair of edges $[(i,j),(k,l)]$ with $(i,j) \neq (k,l)$ a *qualified pair* if $r_{ij} > r_{kl}$. Clearly, there are at most $m(m-1)/2 = O(m^2)$ qualified pairs. Now suppose that T' and T'' are two adjacent minimum spanning trees at $\mu = \bar{\mu}$, and T'' is obtained from T' by replacing a tree edge (i,j) with a non-tree edge (k,l). The path optimality conditions tell us that for any nontree edge $f = (u,v)$ we have $c_e + \mu r_e \leq c_f + \mu r_f$ for all tree edges e in the tree path connecting u and v. Since both T' and T'' are minimum spanning trees at μ we have equality if we choose $e = (i,j)$ and $f = (k,l)$. Since we are increasing μ from $-\infty$ to ∞ we can conclude that $r_{ij} > r_{kl}$, *i.e.*, the pair $[(i,j),(k,l)]$ is a qualified pair. Hence, each change in the spanning tree uses up a qualified pair and clearly no qualified pair can be repeated. Thus, we obtain at most $O(m^2)$ spanning trees during the parametric analysis. □

This also means that the lower hull consists of at most $O(m^2)$ points, *i.e.*, the hull approach cannot take more than $O(m^2)$ iterations of MST computations which is strongly polynomial in the input.

Ravi and Goemans (1996) have shown how to use the elegant *parametric search* technique of Megiddo (1983) to obtain a bound of $O(\log^2 n)$ for the number of iterations in solving the Lagrangean relaxation.

Solving the Lagrangean Relaxation with Parametric Search: It is well known that the minimum spanning tree solution only depends on the linear order which is induced on the set of edges by their weights. Our edge weights are of the form $\tilde{c}_e = c_e + \mu r_e$ and we want to find the parameter μ^* that leads to the optimum of the Lagrangean relaxation.

For any given value of μ we can determine whether $\mu < \mu^*$, $\mu = \mu^*$, or $\mu > \mu^*$. For this purpose, among all optimum trees with respect to $\tilde{c}$, we can find the two trees T_{min} and T_{max} which have smallest and largest resource consumption. This can be done by using a lexicographic ordering of the edges instead of the ordering induced by $\tilde{c}$. For example, to compute T_{min}, we use the ordering $(r_e, c_e) < (r_f, c_f)$ if $c_e < c_f$ or if $c_e = c_f$ and $r_e < r_f$. Then $\mu < \mu^*$ if $r(T_{min}) > \lambda$, $\mu > \mu^*$ if $r(T_{min}) < \lambda$, and μ is (a possible value for) μ^* otherwise.

If we now try to find an optimum tree for the value μ^*, without knowing μ^*, we would like to sort the edges with respect to their costs at μ^*. Given two edges e and f, we can determine if $c_e^* < c_f^*$, $c_e^* = c_f^*$, or $c_e^* > c_f^*$ without actually knowing μ^*. We only

need to determine the breakpoint, say μ_{ef}, of the two linear functions $\tilde{c}_e$ and $\tilde{c}_f$ as a function of μ and determine if $\mu_{ef} < \mu^*$, $\mu_{ef} = \mu^*$, or $\mu_{ef} > \mu^*$. This can be done by two minimum spanning tree computations (to determine T_{min} and T_{max} as above) at the value μ_{ef}.

Therefore, if we use an algorithm which makes $O(m \log m)$ comparisons to determine the ordering of the weights at μ^*, we will be able to determine μ^* and the corresponding tree by $O(m \log m)$ minimum spanning tree computations.

Megiddo (1983) proposed doing that much more efficiently. Instead of using a serial sorting algorithm, suppose now that we use a parallel sorting algorithm, repeatedly restricting the interval containing μ^* until the edges are sorted with respect to c^*. Specifically, we apply the parallel sorting algorithm of Preparata (1978) with $O(m \log n)$ processors to the set of c_e^*'s. The process works in $O(\log n)$ phases. During a single phase, $O(m \log n)$ critical values are produced by sequentially simulating the multiprocessor machine. Then we perform a binary search consisting of $O(\log n)$ minimum spanning tree operations to obtain a restricted interval. This implies a time bound of $O(m \log n + (m + n \log n) \log n)$ per round, which results in a total running time of $O(m \log^2 n + n \log^3 n)$ and thus only $O(\log^2 n)$ minimum spanning tree computations. Given the correct permutation of the edges, we can find μ^* as well as the corresponding tree(s). Hence using Megiddo's technique, we have the following theorem giving a strongly polynomial bound on the number of minimum spanning tree computations.

Theorem 4.2.2 (Ravi and Goemans (1996)): The Lagrangean relaxation of CMST can be computed using $O(\log^2 n)$ minimum spanning tree computations implying a bound of $O(m \log^2 n + n \log^3 n)$.

However, in practice we usually have $C, R = O(n)$, so the hull approach should be preferred since it is much easier to implement.

4.2.3 Performance Guarantee

Ravi and Goemans (1996) also showed that in solving the relaxation we actually get a performance guarantee. Define an (α, β)-approximation for CMST as a polynomial-time algorithm that always outputs a spanning tree with total cost at most αOPT and total resource at most $\beta\lambda$[5].

It can be shown that (a slight modification of) the hull approach computing the Lagrangean relaxation is a $(1, 2)$-approximation and, provided that $OPT \geq C$, also a $(2, 1)$-approximation algorithm. The proof of the performance guarantee exploits the

[5] Observe that the definition is not completely symmetrical in the two cost functions as the limit λ is given.

fact that two adjacent spanning trees on the spanning tree polytope[6] differ by exactly two edges (one in each tree).

Theorem 4.2.3 (Ravi and Goemans (1996)): Let c^* denote the cost function leading to the optimum LB of the Lagrangean relaxation, and let C and R denote the maximal cost and resource of an edge, respectively. There exists a minimum spanning tree T_1 with respect to c^* of cost at most $LB \leq OPT$ and of resource at most $\lambda + R$. Additionally, there exists a minimum spanning tree T_2 with respect to c^* of cost at most $LB + C \leq OPT + C$ and of resource at most λ.

Proof. Computing the optimum of the Lagrangean relaxation using either the hull approach or parametric search we get two spanning trees $T^<$ and $T^>$ that are minimal with respect to the optimal lower bound costs c^*. We know that $c(T^<) \geq OPT \geq LB$ and $r(T^<) \leq \lambda$. On the other hand we know that $c(T^>) \leq LB \leq OPT$ and $r(T^>) \geq \lambda$.

To derive the existence of a minimum spanning tree with respect to c^* of resource between λ and $\lambda + R$ we use the adjacency relationship on the spanning tree polytope. This adjacency relationship follows from the fact that forests of a graph define a matroid, the graphic matroid.

Lemma 4.2.2: The spanning trees T and T' are adjacent on the spanning tree polytope if and only if they differ by a single edge swap, *i.e.*, there exist $e \in T$ and $e' \in T'$ such that $T - e = T' - e'$.

By considering the optimum face of the spanning tree polytope induced by the spanning trees minimal with respect to c^*, this lemma implies that if we have two optimum trees $T^<$ and $T^>$ then there must exist a sequence $T^< = T_0, T_1, \ldots, T_k = T^>$ of optimum spanning trees so that T_i and T_{i+1} are adjacent for $i = 0, \ldots, k-1$. But T_i and T_{i+1} only differ in one edge swap. Thus, there must exist two adjacent spanning trees T_i and T_{i+1} with $r(T_{i+1}) \leq r(T_i) + R \leq \lambda + R$, moreover, we have $c(T_{i+1}) \leq OPT$. Similarly, it can be seen that T_i obeys the desired cost and resource bounds.

To compute the sequence $T_0, T_1, \ldots, T_k$ starting from T_0, we repeatedly swap an edge e in T_k but not in the current tree with a maximum (lower bound) cost edge not in T_k but on the cycle closed by e. We can stop when we reached a tree with resource at least λ. This sequence will therefore end in $k \leq n-1$ steps. Naively implemented, this would take $O(n)$ time per swap, for a total running time of $O(n^2)$. However, using dynamic trees (Sleator and Tarjan 1983) we can do the series of swaps in $O(n \log n)$ time. □

[6]The convex hull of incidence vectors of spanning trees.

Corollary 4.2.2: A $(1,2)$-approximation of CMST can be computed in $O(\log^2 n(m + n\log n))$ time or, alternatively, in $O(\log(nRC)(m + n\log n))$ time.

Proof. Compute the optimum of the relaxation with parametric search or the hull approach. Compute the sequence of optimum spanning trees in $O(n\log n)$ time until we find a tree T with $c(T) \leq OPT$ and $r(T) \leq \lambda + R$. Hence, $r(T) \leq 2\lambda$ if we assume to have pruned all edges with resource greater than λ. □

Unfortunately, we don't get a $(2,1)$-approximation with the same arguments since we do not know OPT and thus cannot prune edges with cost exceeding OPT. So we only get this weaker result.

Corollary 4.2.3: If $OPT \geq C$ then a $(2,1)$-approximation of CMST can be computed in $O(\log^2 n(m + n\log n))$ time or, alternatively, in $O(\log(nRC)(m + n\log n))$ time.

4.2.4 A PTAS for CMST

Ravi and Goemans (1996) show how the $(1,2)$-approximation algorithm can be turned into a PTAS by modifying the initial edge pruning rule. Earlier, we pruned all edges with resource greater than λ since no such edge edge would be used in any feasible solution. The approximation guarantee of 2λ on the resource consumption of the solution followed from Corollary 4.2.2. To reduce this ratio, we could prune all edges with resource greater than $\epsilon\lambda$ for some fixed $\epsilon > 0$. Then R would be at most $\epsilon\lambda$, resulting in a final tree of resource consumption at most $(1+\epsilon)\lambda$. However, we may discard edges that could possibly be used in an optimal solution. The key observation is that at most $1/\epsilon$ of the pruned edges can be used in any optimal solution, and there are only $O(n^{O(1/\epsilon)})$ choices of subsets of pruned edges that may occur in any optimal solution. For each one of these polynomially many choices, we include the chosen edges in the tree, shrink the connected components , and run our algorithm on the resulting graph with a resource limit of λ minus the resource of the chosen edges. The solution output is the tree with minimum cost among all trees over all the choices[7]. The proof that the cost of the returned tree is at most the optimum value OPT is completed by considering the running of the algorithm for the same choice of the chosen edges as in some optimal solution of CMST.

This PTAS gives us a solution with cost of at most OPT and resource consumption of at most $(1+\epsilon)\lambda$ for any fixed $\epsilon > 0$. However, we would be more interested in a "traditional" PTAS giving us a feasible solution with cost of at most $(1+\epsilon)OPT$.

[7]Note that all trees have resource consumption of at most $(1+\epsilon)\lambda$.

The idea now is to use costs instead of resources as the budgeted objective in the available algorithm. Consider running the given algorithm for all possible integral limit values on the cost of the tree to find a tree of approximately minimum resource. Over all these runs, find the smallest value C' of the limit so that the resource of the tree output is at most λ. Since no smaller value of the cost limit of the tree gives a tree of resource at most λ, it must be the case that C' is a lower bound on the cost of any spanning tree of resource at most λ. But the tree obtained by running the given algorithm with a cost limit of C' must have cost at most $(1+\epsilon)C'$, and therefore has the desired properties. Binary search is used to speed up the determination of C' using $O(\log(nC))$ invocations of the given approximation algorithm. This is still polynomial in the input but no longer strongly polynomial.

4.2.5 Problem Reductions

As in the constrained shortest path case it is also possible here to use the resource limit(s) and upper and lower bounds to prune nodes and edges that cannot be part of an optimal solution. Aggarwal, Aneja, and Nair (1982) were the first to propose a pruning step. They suggested finding for each edge e a minimum resource spanning tree T containing e. If $r(T) > \lambda$ then it is safe to delete e from the network. Similarly, we can use a given upper bound UB on the cost of the optimal solution of CMST. Find for each edge e a minimum cost spanning tree T containing e. Edge e can be pruned if $c(T) > UB$. Aggarwal, Aneja, and Nair (1982) suggested applying the above steps repeatedly until no change results in the network (as Dumitrescu and Boland (2000) proposed for CSP).

We will extend their pruning ideas to also considering the lower bound costs and to shrink edges. This results in a more effective pruning step as we will see in the experimental section. We follow the ideas of Eppstein (1990) who used them to obtain a more efficient spanning tree ranking algorithm.

Let T be a spanning tree that is minimal with respect to the weight function w. Every non-tree edge has a single tree replacement edge. We define $R_G(e)$ for a non-tree edge e connecting nodes x and y to be that edge on the tree path between x and y having the highest weight.

Lemma 4.2.3: Let $R_G(e)$ denote the replacement edges of a graph G with respect to weight function w, and let UB and LB denote an upper and a lower bound on the weight of an optimal tree, respectively, then any edge e with $w(e) - w(R_G(e)) > UB - LB$ can be pruned from the graph. Moreover, the pruning can be implemented in time $O(m\alpha(m,n))$.

Proof. It follows from the definition that $w(e) - w(R_G(e))$ is the weight added by including e in a spanning tree. If this exceeds the gap between upper and lower bound, this edge cannot be part of an optimal solution. The replacement edges $R_G(e)$ can be computed efficiently in $O(m\alpha(n,m))$ time using a result of Tarjan (1979) as shown in (Eppstein 1990). The time bound follows. □

This implies problem reductions for CMST regarding the resource function(s) where we have $\lambda^{(i)}$ as an upper bound and $\min_{r_i}$ as lower bound for $i = 1, \ldots, k$. Given upper and lower bounds on the cost (as we have after the hull approach) we can also do problem reductions with respect to costs and lower bound costs.

It is also possible to reduce the number of nodes in the graph by contracting edges that are guaranteed to be part of an optimal solution. Just as each non-tree edge has a single tree replacement edge, it turns out that each tree edge has a single non-tree replacement edge $r_G(e)$. For any edge e in the spanning tree, disconnecting the tree into two components T_1 and T_2, we define the replacement edge $r_G(e)$ to be the least weight edge in G, other than e, between a vertex in T_1 and one in T_2.

Lemma 4.2.4: Let $r_G(e)$ denote the replacement edges of a graph G with respect to cost function c, and let UB and LB denote an upper and a lower bound on the cost of an optimal tree, respectively, then any edge e with $c(r_G(e)) - c(e) > UB - LB$ can be contracted as it will be contained in an optimal solution. Moreover, the contractions can be implemented in time $O(m\alpha(m,n))$.

Proof. For each edge, $c(r_G(e)) - c(e)$ is the extra cost that we would have to add to the tree if we were to remove e. If this extra cost exceeds the gap between upper and lower bound, we can conclude that e has to be part of an optimal solution, so we can contract edge e. The replacement edges $r_G(e)$ can be computed efficiently in $O(m\alpha(n,m))$ time using a result of Tarjan (1979) as shown in (Eppstein 1990). The time bound follows. □

We will see in the experiments that this pruning strategy improves on the pruning of Aggarwal, Aneja, and Nair (1982) and allows a more efficient gap closing step.

4.2.6 Gap Closing

Solving the Lagrangean relaxation we get an upper and a lower bound on the optimal solution of CMST. As in the CSP case we are interested in closing this duality gap.

Aggarwal, Aneja, and Nair (1982) use a branch and discard strategy that essentially corresponds to a minimum spanning tree ranking algorithm with pruning, whereas

Hamacher and Ruhe (1994) apply a local search heuristic to close the gap in their algorithm for the single resource case.

We will use our general solution ranking idea and close the gap with a k-minimum spanning tree ranking algorithm:
Katoh, Ibaraki, and Mine (1981) proposed a k-minimum spanning tree algorithm that runs in time $O(Km + \min\{n^2, m \log\log n\})$ and space $O(K + m)$. Their algorithm computes a spanning tree T_j when $T_1, T_2, \ldots, T_{j-1}$ are given using the branch-and-bound technique of Lawler (1972) that we presented in Section 4.1.2. Using the special properties of spanning trees they improve upon Lawler's general time bound.
Eppstein (1990) used reduction techniques as described in Section 4.2.5 to obtain a $O(m \log \beta(m,n) + k^2)$ time bound where $\beta(m,n) = \min\{i : \log^{(i)} n \leq m/n\}$ that improves on the previous bound if $k = O(m)$.

We use a spanning tree ranking algorithm to close the duality gap as described in Section 4.1: We use the optimal reduced costs as cost function and rank spanning trees until we have swept the possible area. The mentioned online pruning again applies.

Despite the strongly polynomial size of the hull in the CMST case, we again cannot give a polynomial worst case bound on the number of paths to be ranked in the gap closing step.

4.2.7 Experiments

Our CNOP package[8] offers a 2-step implementation for CMST. LEDA's implementation of Kruskal's algorithm for minimum spanning trees is used as the core function in the hull approach. The gap closing step is done with an implementation of the spanning tree ranking algorithm of Katoh, Ibaraki, and Mine (1981). The problem reductions of Section 4.2.5 are also provided.

We use benchmarks of the minimum-cost reliability-constrained spanning tree problem that arises in communication networks: We are given a set of n stations in the plane that can communicate with each other. We now want to connect the stations, the cost of a connection might be modeled by the distance of the stations and the reliability of a connection by its fault probability. We now want to compute a minimum cost connection (spanning tree) so that its total fault probability is beyond a given limit (see Figure 4.4). In our benchmarks, we place the stations randomly in a 1000×1000 square. Then we connect nodes lying in a certain range of each other to obtain a clustered graph with $m = 4n$. The cost of an edge is its euclidean distance. We choose random fault probabilities from the range [10..20] as edge resources and impose

[8] Refer to Chapter 5 for a detailed discussion.

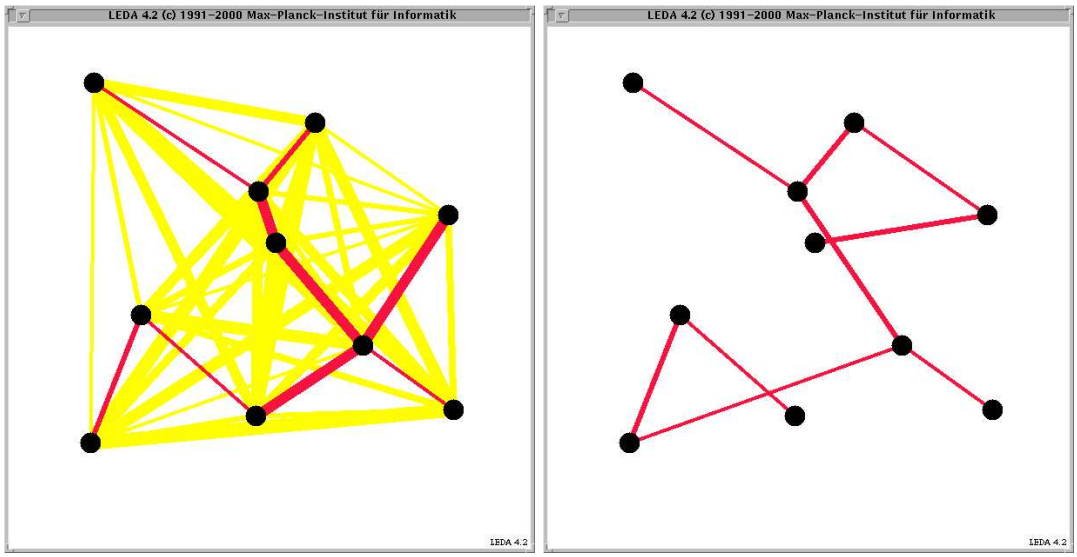

Figure 4.4: Minimum cost spanning tree and Minimum cost reliability constrained spanning tree. Width of edges corresponds to fault probability.

a "normal" constraint (as in the CSP experiments in Section 3.5) on the resource consumption, *i.e.*, fault probability.

We want to experimentally analyze the number of iterations needed to solve the relaxation, the quality of the obtained upper and lower bounds, the effectivity of the problem reductions and the number of spanning trees to be ranked in the gap closing step.

Table 4.1 shows the results. We observe that the number of iterations of the hull approach needed to solve the relaxation rises very slowly for increasing network sizes. The gap between the upper and lower bounds is extremely small, we sometimes even obtain the optimal solution with the hull approach. This confirms the experimental observations in the constrained shortest path case.

Now we consider the problem reductions. The reductions proposed by Aggarwal, Aneja, and Nair (1982) are not effective and leave our benchmark networks unreduced. Our resource reductions also do not reduce the problem size. This is due to the resource distribution on the edges and to the fact that the gap between the resource constraint and the value of the minimum resource spanning tree is too big. However, our cost reductions using upper and lower bounds are very effective (see columns 6 and 7 in Table 4.1) since the bounds are so good.

Finally, we turn to the gap-closing step. We observe that only a very small number of spanning trees has to be ranked until the optimum is found. This can be explained

size		hull approach			reduced size		gap-closing	
n	m	UB	LB	it	n	m	# trees	OPT
250	1250	14504.7	14474.6	9	143	343	5	14475
250	1250	15639.8	15606.8	10	155	382	50	15607.7
250	1250	16089.7	16040.6	11	194	538	18	16040.9
250	1250	15572.6	15537.2	10	156	368	5	15537.2
500	2500	28297	28265.7	12	300	709	6	28265.8
500	2500	28561.1	28561.1	11	0	0	0	28561.1
500	2500	29014.3	28982.9	10	301	744	39	28983.9
500	2500	29202.1	29170.7	11	298	710	13	29170.9
750	3750	39788.2	39788.2	10	0	0	0	39788.2
750	3750	40078.2	40046.9	11	444	1084	13	40047.1
750	3750	40381	40351.6	12	471	3404	4	40351.6
750	3750	40722.5	40675.9	11	600	1587	58	40676.2
1000	5000	49420.1	49405.5	12	363	738	3	49405.6
1000	5000	50916.5	50901.7	13	343	801	30	50901.9
1000	5000	50575.2	50575.2	11	0	0	0	50575.2
1000	5000	51442.8	51427	13	331	757	8	51427.2

Table 4.1: Performance of the 2-step method on the minimum-cost reliability-constrained spanning tree benchmarks.

by the clustered network structure and the resource distribution. A large number of spanning trees exists for most given resource consumptions which results in a large number of hull points. Thus, the lower bound almost constitutes the optimum which means that only very few spanning trees have to be ranked until we hit the optimal solution.

Since the number of trees to be ranked in the gap closing step is so small, it is more efficient to omit the problem reduction step as it takes more time than is gained in the gap closing step on the reduced graph[9].

The experiments show that the 2-step method for CMST is very efficient on our benchmarks. However, there are certainly other cost/resource distributions and graph types, where the bounds from the relaxation are not as excellent and lead to a much larger number of spanning trees to be ranked to determine the optimum.

[9]The number of spanning trees to be ranked is usually identical or changes only minimally.

Sam I am	I don't like them
Green eggs and ham	at all

Sam I am	I don't like them
Green eggs and ham	at all

Table 4.2: Layout options for a table.

Sam I am

Table 4.3: Non-nesting configurations for the content "Sam I am".

4.3 The Table Layout Problem

In this section we study a geometric problem arising in typography: the problem of laying out a two-dimensional table. Each cell has content associated with it and we have choices of the geometry of cells.

Anderson and Sobti (1999) formulated a combinatorial version of this problem that fits into our constrained network optimization problems. We review their formulation and illustrate the application of our 2-step method.

4.3.1 Problem Formulation

An $M \times N$ table is a two-dimensional array with M rows and N columns, which has content associated with each of its MN cells (see Table 4.2). For each cell, there is a specified set of configurations in which the associated item can be displayed. A configuration is, basically, a rectangle specified by giving its height and its width (see also Table 4.3). In other words, a configuration is a tuple of the form $(\mathit{height}, \mathit{width})$, and the configuration set C_{ij} is of the form $\{(h_{ij}^k, w_{ij}^k) | 1 \leq k \leq K_{ij}\}$ for $1 \leq i \leq M$ and $1 \leq j \leq N$. A table can be displayed in a variety of different layouts (see Table 4.2). Suppose $\mathcal{T}$ is an $M \times N$ table. A layout $\mathcal{L}$ of $\mathcal{T}$ is a selection of a configuration for each cell. More formally, for each i, j we have an integer l_{ij} $(1 \leq l_{ij} \leq K_{ij})$ giving the index of the configuration used from the set C_{ij}. When we have fixed the configuration for each cell, we can determine the dimensions of the table. Let h_{ij} and w_{ij} be the height and the width necessary for the (i,j)-th cell, *i.e.*, let $h_{ij} = h_{ij}^{l_{ij}}$ and $w_{ij} = w_{ij}^{l_{ij}}$. The height h_i of row i is the maximum height of any cell in the row, and the width w_j

of column j is the maximum width of a cell in the column. The height of the layout, $height(\mathcal{L})$ is $\sum_i h_i$, and the width of the layout, $width(\mathcal{L})$ is $\sum_j w_j$.

In the table layout problem, given a width limit W for a table $\mathcal{T}$, we want to find the layout $\mathcal{L}$ of $\mathcal{T}$ with minimum height $height(\mathcal{L})$ so that $width(\mathcal{L}) \leq W$.
The ILP formulation uses 0-1-variables x_{ij}^k to indicate whether or not cell i, j is in configuration k. Then we obtain this formal ILP:

$$\begin{aligned} \min \quad & \sum_i \max_{j,k} \{h_{ij}^k x_{ij}^k\} \\ \text{s.t.} \quad & \sum_j \max_{i,k} \{w_{ij}^k x_{ij}^k\} \leq W \\ & \sum_k x_{ij}^k = 1 \\ & x_{ij}^k \in \{0, 1\} \end{aligned}$$

In the table layout problem, we assume that the configurations do not nest. Given two configurations, one must have greater height and the other greater width. This means we only consider the Pareto-optimal configurations. We will assume that the configurations are ordered by increasing height and decreasing width.

4.3.2 Complexity

The recognition version of the table layout problem, the question of whether there is a layout $\mathcal{L}$ with $height(\mathcal{L}) \leq H$ and $width(\mathcal{L}) \leq W$ for the given table, is $\mathcal{NP}$-complete, even when the tables are restricted to be *simple*, *i.e.*, the heights and widths of configurations are restricted to being 1 or 2.

For simple tables, a cell is said to have a *choice* item if there are two distinct configurations to choose from, and a *dummy* item if there is only a single configuration. In the layout problem, a dummy item has the configuration set $\{(1, 1)\}$, and a choice item has the configuration set $\{(1, 2), (2, 1)\}$.

Theorem 4.3.1 (Anderson and Sobti (1999)): The layout problem for simple tables is $\mathcal{NP}$-complete.

Proof. We reduce the clique problem to the layout problem. Suppose we want to determine if there is a clique of size F in an N vertex, M edge graph G. We construct a *simple* $N \times M$ table $\mathcal{T}$ as follows: For any i, j, cell (i, j) of $\mathcal{T}$ has a choice item if the i-th vertex of G is incident on the j-th edge of G, and otherwise has a dummy item.

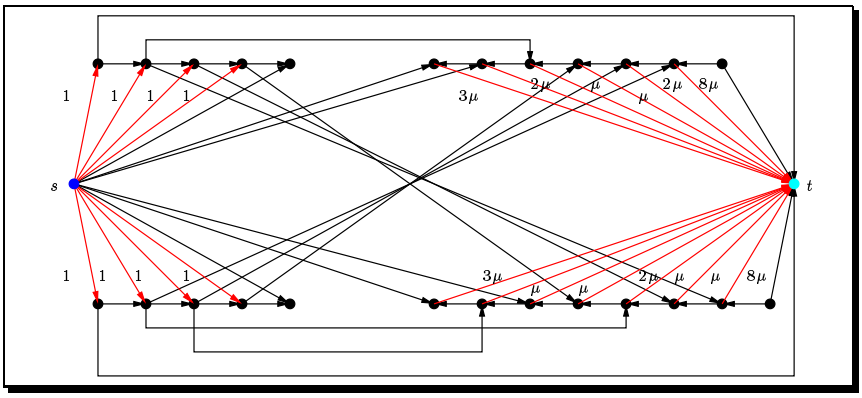

Figure 4.5: Flow graph corresponding to the table layout problem of Table 4.2. Black edges are infinite capacity edges.

(Essentially, $\mathcal{T}$ is just the incidence matrix of G with the 0-entries replaced by choice items.) Let $H = N + F$ and $W = 2M - \binom{F}{2}$. Then, G has a clique of size F if and only if table $\mathcal{T}$ has a layout $\mathcal{L}$ with $height(\mathcal{L}) \leq H$ and $width(\mathcal{L}) \leq W$. □

4.3.3 Reduction to Minimum Cut

Anderson and Sobti (1999) found an interesting connection between table layout and network flow. They showed how to solve the weighted sum minimization problem by constructing a flow graph where the minimum cut corresponds to a layout $\mathcal{L}$ which minimizes $height(\mathcal{L}) + \mu \cdot width(\mathcal{L})$. The flow graph has the following bipartite graph structure: We have a source, a sink, and row vertices and column vertices. The source is connected to row vertices, column vertices are connected to the sink, and there are infinite capacity edges between some row and column vertices. The vertices for a particular row correspond to each possible height for the row (determined by the configurations of cells in the row). These vertices are chained together (through infinite capacity edges) in increasing order of height, and there is an edge from the source to a row vertex with capacity equal to the difference between its height and its predecessor's height. A similar construction is used for each column, there is an edge from a column vertex to the sink with capacity equal to μ times the difference between its width and its predecessor's width. The edge for a particular configuration of cell (i, j) is placed between a vertex of row i and column j.

We now give the formal definition of Anderson and Sobti (1999) for the flow network G_μ for a given $M \times N$-table $\mathcal{T}$:

We define the sets of "row-heights" and "column-widths" as $RH(i) = \bigcup_{j=1}^{N} \{h_{ij}^k : 1 \leq k \leq K_{ij}\}$ and $CW(j) = \bigcup_{i=1}^{M} \{w_{ij}^k : 1 \leq k \leq K_{ij}\}$.

It is easy to observe that if layout $\mathcal{L}$ minimizes μ-weight, then for each i, j we have $h_i \in RH(i)$ and $w_j \in CW(j)$. For notational convenience, we rename elements of $RH(i)$ and $CW(j)$ so that $RH(i) = \{a_{i,k} : 1 \leq k \leq d_i\}$ and $CW(j) = \{b_{j,k} : 1 \leq k \leq e_j\}$, with $a_{i,1} < a_{i,2} < \cdots < a_{i,d_i}$, and $b_{j,1} < b_{j,2} < \cdots < b_{j,e_j}$. For every i, j, k $(1 \leq k \leq K_{ij})$, let p_{ij}^k denote the rank of h_{ij}^k in $RH(i)$, and let q_{ij}^k denote the rank of w_{ij}^k in $CW(j)$. Thus, $h_{ij}^k = a_{i,p_{ij}^k}$ and $w_{ij}^k = b_{j,q_{ij}^k}$.
The vertex set of G_μ is $U \cup V \cup \{s, t\}$, where $U = \bigcup_{i=1}^{M} U_i$ and $V = \bigcup_{j=1}^{N} V_i$ with $U_i = \{u_{i,k} : 1 \leq k \leq d_i + 1\}$ and $V_j = \{V_j = \{v_{j,k} : 1 \leq k \leq e_j + 1\}$. Vertices in U_i constitute the gadget for row i, vertices in V_j for column j.
The edge set of G_μ is $E_U \cup E_V \cup X \cup Y \cup Z$, where
$E_U = \bigcup_{i=1}^{M}\{(u_{i,k}, u_{i,k+1}) : 1 \leq k \leq d_i\}$, $E_V = \bigcup_{j=1}^{N}\{(v_{j,k+1}, v_{j,k}) : 1 \leq k \leq e_j\}$,
$X = \{(s, u)) : u \in U\} \cup \{(s, v_{j,1}) : 1 \leq j \leq N\}$,
$Y = \{(v, t)) : v \in V\} \cup \{(u_{i,1}, t) : 1 \leq i \leq M\}$, and
$Z = \bigcup_{i=1}^{M} \bigcup_{j=1}^{N}(Z_{ij} \cup \{(u_{i,p_{ij}^1}, t), (s, v_{j,q_{ij}^{K_{ij}}})\})$ with $Z_{ij} = \{(u_{i,p_{ij}^k}, v_{j,q_{ij}^{k-1}}) : 1 < k \leq K_{ij}\}$.
Edge capacities in G_μ are as follows: All edges in E_U, E_V and Z have infinite capacity. Arcs in X of the form: $(s, v_{j,1})$ have infinite capacity, $(s, u_{i,1})$ have capacity $a_{i,1}$, $(s, u_{i,k})$ (for $1 \leq k \leq d_i$) have capacity $a_{i,k} - a_{i,k-1}$, and (s, u_{i,d_i+1}) have infinite capacity.
Arcs in Y of the form: $(u_{i,1}, t)$ have infinite capacity, $(v_{j,1}, t)$ have capacity $\mu b_{j,1}$, $(v_{j,k}, t)$ (for $1 \leq k \leq e_j$) have capacity $\mu(b_{j,k} - b_{j,k-1})$, and (v_{j,e_j+1}, t) have infinite capacity. This completes the construction. Figure 4.5 gives an example.

What can we say about the number of nodes and edges in this graph? Clearly, the number of nodes n is bounded by $O((N + M)K)$ where $K = \max\{K_{ij}\}$ is the maximal number of configurations in a cell. Let L be the maximal number of words in a cell and let C be the maximal number of characters of a word in the content. A simple way to compute all configurations in a cell, hence bounding the maximal number of configurations, is to compute, for each h, the minimum width w that gives a cell of height h. To find the minimum width cell for a fixed height, we use binary search on the line width that is bounded by $O(LC)$. Given the width of a cell, we can compute its height in $O(L)$ time with a simple greedy algorithm[10]. This gives us a total running time of $O(L^2 \log(LC))$. Hence, the number of nodes n in the flow graph is bounded by $O((N + M)L^2 \log(LC))$. The number of edges m can be as large as $O(n^2)$.

A minimum s-t cut of this graph is found. Since there is a finite capacity cut, the infinite capacity edges do not cross the cut. This makes it possible to identify row heights and column widths which correspond to the cut. There is a direct correspondence between the cut size and the weighted sum of the height and width, so the minimum cut gives the optimal table. The key point to verify is that for each cell, there is a configuration which can fit in the cell. This holds because of the placement of infinite capacity edges

[10] Anderson and Sobti (1999) give an improved but more complicated algorithm.

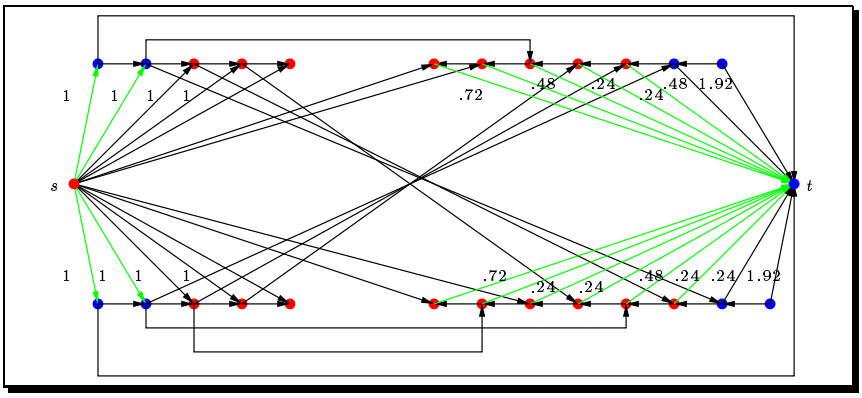

Figure 4.6: Minimum s-t cut in flow graph. Green edges are cut edges. Value of cut is 8.56.

from row vertices to column vertices.

The following theorem describes the close correspondence between min-cuts in G_μ and layouts of $\mathcal{T}$.

Theorem 4.3.2 (Anderson and Sobti (1999)): For each s-t min-cut Q in G_μ, there exists a layout $\mathcal{L}$ of $\mathcal{T}$ with capacity of Q being equal to the μ-weight of $\mathcal{L}$. Similarly, for each layout $\mathcal{L}$ of $\mathcal{T}$ that minimizes the μ-weight, there exists an s-t cut Q in G_μ with capacity of Q being equal to the μ-weight of $\mathcal{L}$.

Given a table $\mathcal{T}$ and a positive real number μ, a minimum μ-weight layout $\mathcal{L}$ of $\mathcal{T}$ can be computed in polynomial time. We first construct the flow network G_μ and then find an s-t minimum cut Q in G_μ. This can be done by finding a maximum flow in G_μ in time $O(n^3)$ with a preflow push algorithm and using the max-flow-min-cut theorem to obtain an s-t minimum cut in G_μ.
Layout $\mathcal{L}$ can be recovered from cut Q as follows: Let $S \cup T$ be the partition of the vertex set of G_μ dictated by Q, where $s \in S$ and $t \in T$. Now, one can observe that for every $i : 1 \leq i \leq M$, there exists some integer $f_i : 1 \leq f_i \leq d_i$ so that $\{u_{i,k} : 1 \leq k \leq f_i\} \subset T$ and $\{u_{i,k} : f_i < k \leq d_i + 1\} \subset S$. Similarly, for every $j : 1 \leq j \leq N$, there exists some integer $g_j : 1 \leq g_j \leq e_j$ so that $\{v_{j,k} : 1 \leq k \leq g_j\} \subset T$ and $\{v_{j,k} : g_j < k \leq e_j + 1\} \subset S$. This occurs due to the infinite capacity arcs in E_U, E_V, X, and Y, and due to the fact that Q is a cut with finite capacity. From the f_i's and the g_j's, we can construct a layout $\mathcal{L}$ of $\mathcal{T}$ in the following manner: Let h_i be a_{i,f_i} and w_j be b_{j,g_j}. For each i, j let l_{ij} be any l such that $h^l_{ij} \leq a_{i,f_i}$ and $w^l_{ij} \leq b_{j,g_j}$. Such an l is guaranteed to exist and the h_i's, w_j's and l_{ij}'s as defined above describe a valid layout with μ-weight equal to the capacity of Q.
Figure 4.6 gives an example for $\mu = 0.24$ and Table 4.4 shows the corresponding layout.

Sam I am	I don't like them
Green eggs and ham	at all

Table 4.4: Table layout resulting from minimum cut of Figure 4.6. Height is 4, width is 19.

4.3.4 Solving the Lagrangean Relaxation

Now we have seen that we can solve the weighted sum minimization problem with a parametric network flow problem which means that the hull approach can be used to solve the Lagrangean relaxation of the table layout problem.

Let L be the maximal number of words to be displayed in a cell, and let C be the maximum number of characters of a word in the content. Then we can conclude that the maximal height of a table layout is bounded by ML and the maximal width by $NL(C+1)$. Thus we get the following bound.

Corollary 4.3.1: The Lagrangean relaxation of the table layout problem can be solved with the hull approach using $O(\log(NMLC))$ min cut computations in the corresponding flow graph.

The bound on the total number of extreme points on the lower hull corresponding to extreme Pareto-optimal table layouts also extends:

Corollary 4.3.2: The maximal number of extreme points on the lower hull is bounded by $O((LC\max\{N,M\})^{2/3})$. The whole lower hull can be computed by a modified version of the hull approach with the same bound on the number of iterations.

Anderson and Sobti (1999) proposed a method equivalent to the hull approach for solving the relaxation but left the running time open. However, they pointed out that Gallo, Grigoriadis, and Tarjan (1989) studied the source parametric maximum flow problem, where the capacity of every source edge (s,j) is a non-decreasing linear function of a parameter $\mu \geq 0$ and all other edge capacities are fixed. This is symmetric to our problem and a general formulation. We now want to determine the maximum flow for p values $0 = \mu_1 \leq \mu_2 \leq \cdots \leq \mu_p$. Let $MF(\mu)$ denote the maximum flow problem for a specific value of μ, $v(\mu)$ its maximum flow value, and $[S(\mu), \bar{S}(\mu)]$ an associated minimum cut. Gallo, Grigoriadis, and Tarjan (1989) show how to modify a preflow-push algorithm to solve this problem by successively determining $MF(\mu_{i+1})$ from $MF(\mu_i)$ in a bound of $O(n^3)$ as for a single maximum flow computation.
It also turns out that the corresponding minimum cuts satisfy the nesting condition

Henning Mankell: Der Mann, der laechelte	Sebastian Haffner: Geschichte eines Deutschen	No Angels: Daylight in your eyes	No Angels: Elle' ments
Joanne Rowling: Harry Potter und der Feuerkelch	Norman Finkelstein: Die Holocaust- Industrie	Glashaus: Wenn das Liebe ist	Daft Punk: Discovery
Joanne Rowling: Harry Potter und der Stein der Weisen	Dietrich Schwanitz: Bildung	Crazy Town: Butterfly	Eric Clapton: Reptile
Joanne Rowling: Harry Potter und die Kammer des Schreckens	Guenther Ogger: Der Boersenschwindel	D-12: Shit on you	Dido: No angel
Joanne Rowling: Harry Potter und der Gefangene von Askaban	Florian Illies: Generation Golf	Daft Punk: One more time	De- Phazz: Death by chocolate
John Grisham: Die Bruderschaft	Hans- Olaf Henkel: Die Macht der Freiheit	Outkast: Ms. Jackson	Peter Maffay: Heute vor dreissig Jahren
Ingrid Noll: Selige Witwen	Dale Carnegie: Sorge dich nicht, lebe!	Rammstein: Sonne	Aerosmith: Just push play
Charlotte Link: Die Rosenzuechterin	Guenther de Bruyn: Preussens Luise	Wheatus: Teenage dirtbag	Goldfrapp: Felt mountain
Rosamunde Pilcher: Wintersonne	Bodo Schaefer: Der Weg zur finanziellen Freiheit	Niemann: Im Osten	Crazy town: The gift of game
Donna Leon: In Sachen Signora Brunetti	Lance Armstrong: Tour des Lebens	Ricky Martin: Nobody wants to be lonely	Robbie Williams: Sing when you're winning

Table 4.5: Table of book and music bestsellers in Germany (March 2001). Width of the table is 88 characters, the height is 27 rows.

$S(\mu_1) \subseteq S(\mu_2) \subseteq \cdots S(\mu_p)$ which allows us to conclude that $p \leq n$.

We can conclude that the piecewise linear function $v(\mu)$ has at most $n-1$ breakpoints, *i.e.*, the lower hull contains at most $n-1$ extreme points. Gallo *et al.* (1989) also present a quite involved algorithm to compute all breakpoints in $O(n^3)$ time.

This means we also have a strongly polynomial algorithm to compute the optimum of the relaxation. However, the nesting property does not allow us to prove an approximation guarantee for the bounds as in the case of CMST.

Let us now show an example that illustrates the quality of the obtained bounds. Table 4.5 is the Top 10 list of the German book and music bestsellers. With some "hand tuning" we came up with a table of 27 rows for a maximum width of 88 characters.

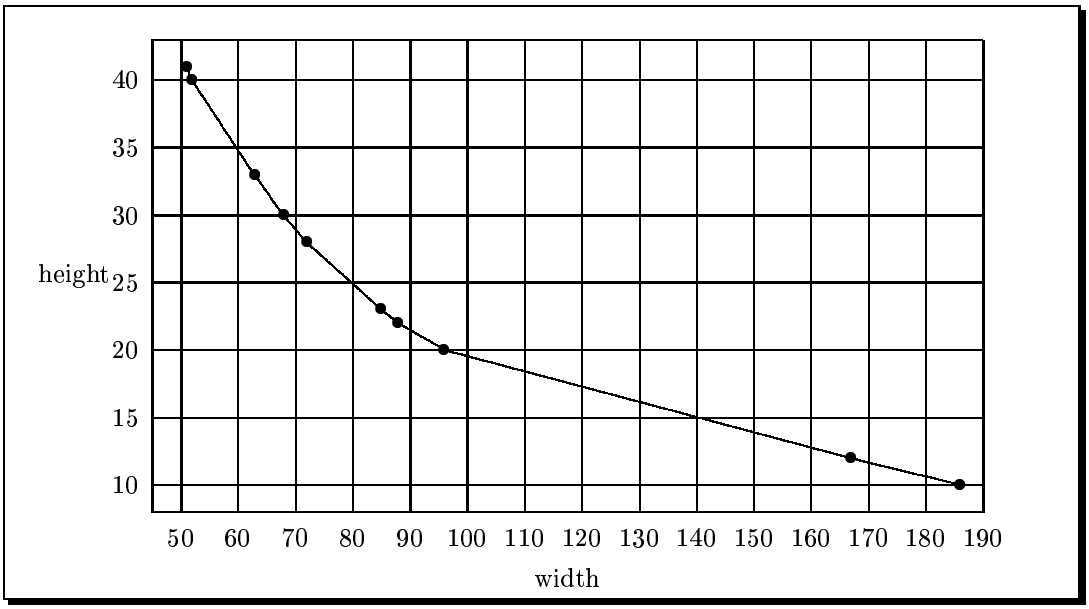

Figure 4.7: Whole hull of the layout problem for Table 4.5 showing the extreme Pareto-optimal layouts.

The corresponding flow graph has 170 nodes and 515 edges. Using an implementation of the weighted sum minimization problem and our CNOP package[11] providing the general hull approach we computed the whole lower hull giving us extreme points corresponding to extreme Pareto-optimal table layouts. The lower hull consists of 10 points (see Figure 4.7).
Since heights and widths are integers, it is often possible to detect optimality even though upper and lower bound do not coincide. For the example of a width limit of 88 characters, we have an extreme point of the hull giving an optimal solution to the table layout problem that is given in Table 4.6. Now we only need 22 rows.

4.3.5 Closing the Gap

We have seen in the previous section that the bounds obtained by the hull approach are usually quite good (which was also observed by Anderson and Sobti (1999) in their experiments). If there still is a duality gap, we may close it by ranking minimum cuts with respect to the optimal reduced costs. Hamacher, Picard, and Queranne (1984) proposed an algorithm for finding the K best cuts in a network that runs in time $O(Kn^4)$ following the generic method of Lawler (1972). We can use this algorithm in our 2-step method to obtain the optimal solution to table layout problems.

[11] Refer to Chapter 5 for a detailed discussion.

Henning Mankell: Der Mann, der laechelte	Sebastian Haffner: Geschichte eines Deutschen	No Angels: Daylight in your eyes	No Angels: Elle' ments
Joanne Rowling: Harry Potter und der Feuerkelch	Norman Finkelstein: Die Holocaust- Industrie	Glashaus: Wenn das Liebe ist	Daft Punk Discovery
Joanne Rowling: Harry Potter und der Stein der Weisen	Dietrich Schwanitz: Bildung	Crazy Town Butterfly	Eric Clapton: Reptile
Joanne Rowling: Harry Potter und die Kammer des Schreckens	Guenther Ogger: Der Boersenschwindel	D-12: Shit on you	Dido: No angel
Joanne Rowling: Harry Potter: und der Gefangene von Asbakan	Florian Illies: Generation Golf	Daft Punk: One more time	De- Phazz: Death by chocolate
John Grisham: Die Bruderschaft	Hans- Olaf Henkel: Die Macht der Freiheit	Outkast: Ms. Jackson	Peter Maffay: Heute vor dreissig Jahren
Ingrid Noll: Selige Witwen	Dale Carnegie: Sorge dich nicht, lebe!	Rammstein: Sonne	Aerosmith: Just push play
Charlotte Link: Die Rosenzuechterin	Guenther de Bruyn Preussens Luise	Wheatus: Teenage dirtbag	Goldfrapp: Felt mountain
Rosamunde Pilcher: Wintersonne	Bodo Schaefer: Der Weg zur finanziellen Freiheit	Niemann: Im Osten	Crazy town: The gift of game
Donna Leon: In Sachen Signora Brunetti	Lance Armstrong: Tour des Lebens	Ricky Martin: Nobody wants to be lonely	Robbie Williams: Sing when you're winning

Table 4.6: Table of book and music bestsellers with minimal height of 22 rows for a width limit of 88 characters.

4.4 Constrained Geodesic Shortest Paths

In this section we examine a continuous version of the shortest path problem. We review an approximation algorithm of Lanthier, Maheshwari, and Sack (2001) for the weighted geodesic shortest path problem on a polyhedral terrain and use this method to get an approximation of the constrained geodesic shortest path problem where each face of the terrain has an additional weight and we have a constraint on the second weighted length of a path. We theoretically and experimentally investigate the approximation quality of our method and close with an ϵ-δ-approximation scheme for the problem.

4.4.1 Geodesic (Weighted) Shortest Paths

So far, the shortest path problems we have considered have been of discrete nature. However, there are important applications in geographical information systems (GIS) where we want to compute a shortest path between two points on a polyhedral terrain. Such a shortest path is a *geodesic* shortest path and this problem is continuous.
More formally, let s and t be two vertices on a given possibly non-convex polyhedron P in $\mathbb{R}^3$, consisting of n triangular faces on its boundary, and let each face have an associated positive weight. An *Euclidean shortest path* $\pi(s,t)$ between s and t is defined to be a path with minimum euclidean length among all possible paths joining s and t that lie on the surface of P. A *weighted shortest path* $\Pi(s,t)$ between s and t is defined to be a path with minimum cost among these paths, where the cost of the path is the sum of the euclidean lengths of all segments multiplied with the corresponding face weight[12].

Currently, the best known algorithm for euclidean shortest paths is due to Chen and Han (1996) and runs in $O(n^2)$ time but is very sensitive to numerical errors. Since most application models are approximations of reality and high-quality paths are favoured over optimal paths that are "hard" to compute, we review approximation algorithms of Lanthier, Maheshwari, and Sack (2001) for the weighted shortest path problem. Their idea is to add Steiner points and edges to discretize the problem:

Fixed Scheme: We place m Steiner points evenly along each edge of P. For each face f_i, $1 \leq i \leq n$ of P, we compute a face graph G_i as follows: The Steiner points, along with the original vertices of f_i, become vertices of G_i. Connect a vertex pair v_a, v_b of G_i to form an edge (v_a, v_b) of G_i if and only if v_a and v_b correspond to Steiner points (or vertices) that lie on different edges of f_i or are adjacent on the same edge of f_i. The weight of a graph edge (v_a, v_b) is the Euclidean distance $|v_a v_b|$ between v_a and v_b times the weight of f_i, and the magnitude of this weight will be denoted as $||v_a v_b||$.

[12] A path segment traveling along an edge of P uses the minimum weight of the adjacent faces.

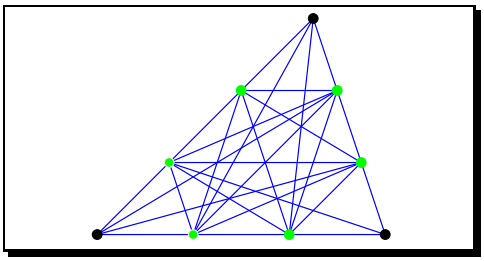

Figure 4.8: Adding Steiner points and edges to a face.

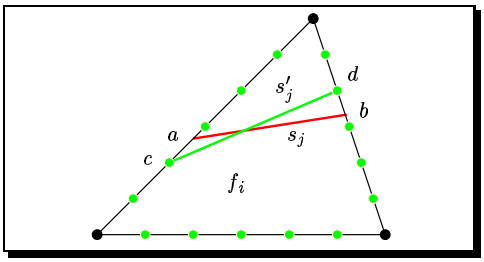

Figure 4.9: A face-crossing segment and its approximation of a weighted shortest path.

Figure 4.8 shows how 6 Steiner points and 26 edges are added to a face to form the face graph G_i when we have $m = 2$. Let L be the maximal length of an edge in P. Assuming that a segment of the optimal path runs through face f_i, what can we say about approximating this segment using edges of G_i ?

Claim 1 (Lanthier, Maheshwari, and Sack (2001)): Given a segment s_j crossing face f_i, there exists an edge s'_j in G_i so that $||s'_j|| \leq ||s_j|| + w_{f_i} \cdot \frac{L}{m+1}$.

Proof. Each edge in P is divided into $m+1$ intervals which have length at most $\frac{L}{m+1}$. Let $s_j = ab$, where without loss of generality, a and b are the end points of s_j lying on edges e_a and e_b of f_i, $e_a \neq e_b$, respectively. Let c (respectively, d) be the Steiner point in f_i, where c (respectively, d) is closest to a (respectively, b) among Steiner points on e_a (respectively, e_b). Since c and d lie on different edges of f_i, we know that there is an edge $e \in G_i$ joining them (see Figure 4.9) and we set $s'_j = e$. The triangle inequality assures that $|s'_j| \leq |\overline{ab}| + |s_j| + |\overline{bd}|$. Since we chose the closer interval endpoints, we have $|\overline{ca}| \leq \frac{L}{2(m+1)}$ and $|\overline{bd}| \leq \frac{L}{2(m+1)}$. Hence, $|s'_j| \leq |s_j| + \frac{L}{m+1}$. Now, multiplying with the face weight w_{f_i}, we get $||s'_j|| \leq ||s_j|| + w_{f_i}\frac{L}{m+1}$. □

A graph G is computed by forming the union of all face graphs G_i, $1 \leq i \leq n$. All edges of G lie on the surface of P hence a path exists in G that approximates the shortest path $\Pi(s,t)$. We get the following approximation guarantee:

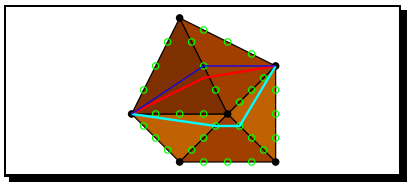

Figure 4.10: Optimal, guaranteed approximate and approximate path.

Lemma 4.4.1 (Lanthier, Maheshwari, and Sack (2001)): Given two vertices s and t of P, there exists a path $\Pi'(s,t)$ in G between the vertices corresponding to s and t so that $||\Pi'(s,t)|| \leq ||\Pi(s,t)|| + \frac{L}{m+1}kW$, where k is the number of segments of $\Pi(s,t)$.

Since a shortest path may have $\Theta(n^2)$ segments (Lanthier, Maheshwari, and Sack 2001) we get the following result.

Corollary 4.4.1 (Lanthier, Maheshwari, and Sack (2001)): Using the fixed scheme, we can compute an approximation $\Pi'(s,t)$ of the weighted shortest path $\Pi(s,t)$ between two vertices s and t on the polyhedral surface P so that $||\Pi'(s,t)|| \leq ||\Pi(s,t)|| + WL$, where L is the longest edge of P and W is the maximum weight among all face weights of P. Moreover, we can compute this path in $O(n^5)$ time.

Note: The bound above can be made tighter and written as $||\Pi'(s,t)|| \leq ||\Pi(s,t)|| + \frac{WL_\Pi}{E_\Pi}$, where L_Π is the sum of the edge lengths of P that $\Pi(s,t)$ passes through and E_Π is the number of edges that $\Pi(s,t)$ passes through.

Lanthier, Maheshwari, and Sack (2001) also proposed an *interval scheme* forcing the intervals between adjacent Steiner points on an edge to be of equal length. This typically reduces the number of Steiner points and edges although the worst cases analysis carries over.
They also presented a heuristic for an efficient refinement of an approximation. After a preliminary approximation $\Pi'(s,t)$, a *buffer* region P' is determined which is the union of all faces intersected by $\Pi'(s,t)$. Then the approximation scheme is applied to P' with an increased number of Steiner points to obtain a refined approximation $\Pi''(s,t)$. However, $\Pi''(s,t)$ is not guaranteed to converge to the optimal path $\Pi(s,t)$, only if the considered buffer contains it (see Figure 4.10).

4.4.2 Applying the Hull Approach

Now we consider a bicriteria variant of the weighted geodesic shortest path problem. Each face f now has two weight values, $w_f^{(1)}$ and $w_f^{(2)}$. Given an s-t path p on the terrain surface, we define the cost c_p of path p to be its $w^{(1)}$-weighted euclidean length, and the resource consumption r_p to be its $w^{(2)}$-weighted euclidean length.
The constrained geodesic (weighted) shortest path problem (CGSP) is to find the minimum cost path from s to t on the surface of P so that its resource consumption does not exceed a given upper limit λ[13].

This continuous problem can also be formulated as an ILP by a slight adaption of the formulation in Section 3.3.

$$x_p = \begin{cases} 1 & \text{if } p \text{ is a (simple) path from } s \text{ to } t \text{ on the surface of } P \\ 0 & \text{otherwise.} \end{cases}$$

$$\begin{aligned} \min \quad & \sum_p c_p x_p \\ \text{s.t.} \quad & \sum_p x_p = 1 \\ & \sum_p r_p x_p \leq \lambda \\ & x_p \in \{0,1\} \end{aligned}$$

In the discrete version (CSP), we had at most an exponential number of paths, but the continuous version (CGSP) has an infinite number of paths, hence, the dual relaxation also has an infinite number of constraints:

$$\begin{aligned} \max \quad & u + \lambda v \\ \text{s.t.} \quad & u + r_p v \leq c_p \qquad \forall p \\ & v \leq 0 \end{aligned}$$

The separation problem is the following: Is there a path q with $\bar{u} + \bar{v} r_q > c_q$ or $\bar{u} > c_q - \bar{v} r_q$ (where $\bar{u}, \bar{v}$ are optimal values for the variables for the current set of constraints). This separation problem is solved by a shortest path computation with the scaled costs $\tilde{c}_s = c_s - \bar{v} r_s = \ell_s(w_f^{(1)} - \bar{v} w_f^{(2)})$ for a segment s on the surface of P.
In the discrete case, we could solve the separation problem exactly, in the continuous case, we have to be satisfied with an approximate solution using one of the schemes discussed in Section 4.4.1.
We now bound the error of the approximate extreme points that we compute with one of the approximation schemes.

[13] A multiple resource definition and formulation is analogous.

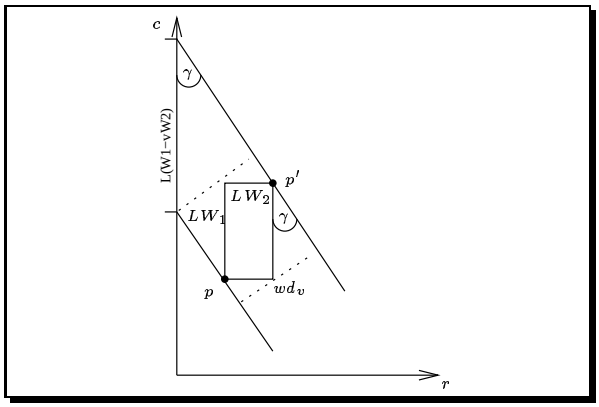

Figure 4.11: Exact and approximate extreme point and the width of the approximation strip.

Lemma 4.4.2: Let W_1 be the maximal face weight of the first weight function $w^{(1)}$, W_2 the maximal face weight of the second weight function $w^{(2)}$, and L be the maximal euclidean length of an edge of P. For any $v \leq 0$, there is an approximation $\Pi'(s,t)$ of the shortest weighted path $\Pi(s,t)$ with respect to weight $\tilde{w}_f = w_f^{(1)} - vw_f^{(2)}$ with

$$\Pi'_{\tilde{w}}(s,t) \leq \Pi_{\tilde{w}}(s,t) + \frac{k}{m+1} L(W_1 - vW_2)$$

Its weighted length with respect to $w^{(1)}$ is

$$\Pi'_{w^{(1)}}(s,t) \leq \Pi_{w^{(1)}}(s,t) + \frac{k}{m+1} LW_1$$

and its weighted length with respect to $w^{(2)}$ is

$$\Pi'_{w^{(2)}}(s,t) \leq \Pi_{w^{(2)}}(s,t) + \frac{k}{m+1} LW_2$$

where k is the number of segments of $\Pi(s,t)$ and m is the number of Steiner points per edge.

Proof. The first claim follows from Lemma 4.4.1 since the maximum scaled weight is $W_1 - vW_2$. The next two claims follow from Lemma 4.4.1 and Claim 1 since weights are multiplicative and since for every segment s_j of the optimal path there is an approximating segment s'_j with $|s'_j| \leq |s_j| + \frac{L}{m+1}$. □

This means that for any scaled weight we are guaranteed a bounded approximation with respect to the scaled weight and both weight functions. Hence, every extreme point p has a guaranteed approximating counterpart p' that lies within a rectangle of

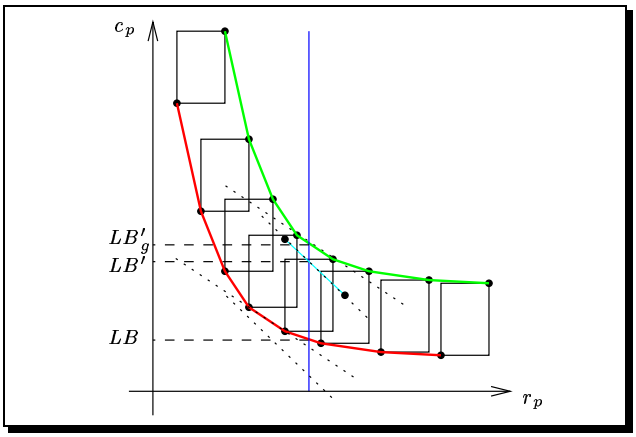

Figure 4.12: The guaranteed hull approximation.

width LW_2 and height LW_1 with lower left endpoint at p if we choose $m = n^2$ (see Figure 4.11).

Using Dijkstra to compute a shortest path $\Pi(s,t)$ with respect to the scaled weight function $\tilde{w}$ in the discretized graph, we get a path $\Pi''(s,t)$ with

$$\Pi''_{\tilde{w}}(s,t) \leq \Pi'_{\tilde{w}}(s,t)$$

where $\Pi'(s,t)$ is the guaranteed approximate path of $\Pi(s,t)$. However, we cannot guarantee that

$$\Pi''_{w_1}(s,t) \leq \Pi'_{w_1}(s,t) \text{ and } \Pi''_{w_2}(s,t) \leq \Pi'_{w_2}(s,t)$$

since $\Pi''(s,t)$ might pass through other faces than $\Pi'(s,t)$ and $\Pi(s,t)$ (see also Figure 4.10). We will argue in the following that this does not cause any problems.

We use the approximate scheme as a subproblem in the hull approach to solve the separation problem approximately. Hence, we compute approximate extreme points and again stop when no further constraint (corresponding to the discretized graph) is violated. Figure 4.12 shows an example of a continuous hull and its guaranteed approximate hull. We know that any approximate extreme point that we obtain must lie in the strip between these two hulls, so there are no problems if such a point "leaves" its guaranteed rectangle. The width wd_v of the strip between the two hull depends on the absolute value of the slopes $|v|$ and can be bounded as

$$L\min\{W_1, W_2\} \leq w_v = L(W_1 \sin\gamma + W_2 \cos\gamma) \leq L\sqrt{W_1^2 + W_2^2} \text{ with } \tan\gamma = \frac{1}{|v|}$$

as illustrated in Figure 4.11 if we choose $m = n^2$.

What can we say about the approximation of the optimum of the dual relaxation of CGSP?

Lemma 4.4.3: Let LB denote the optimal solution of the dual relaxation and $LB^{'}$ the corresponding approximate solution in the discretized graph. Let m denote the number of Steiner points per edge in the fixed scheme and L the maximum edge length in the terrain and W_1 and W_2 the maximum face weights in the terrain with respect to weight functions w_1 and w_2, respectively.
If $\hat{v}$ is the slope of the optimal hull segment in the approximation and $\hat{k}$ is the number of path segments in the exact solution of the last separation problem (with $\hat{v}$) then we get

$$LB^{'} \leq LB + \frac{\hat{k}}{m+1} L(W_1 - \hat{v}W_2).$$

Proof. The claim follows from Lemma 4.4.1. □

Since a weighted shortest path might pass through $\Theta(n^2)$ faces we get the following Corollary.

Corollary 4.4.2: Let LB denote the optimal solution of the dual relaxation and $LB^{'}$ the corresponding approximate solution in the discretized graph. If $\hat{v}$ is the slope of the optimal hull segment in the approximation we get

$$LB^{'} \leq LB + L(W_1 - \hat{v}W_2).$$

Moreover, we can compute this approximate solution to the dual relaxation with the hull approach in $O(n^5 \log(nLW_1W_2))$ time.

Proof. The bounds for the hull approach apply. We know that the maximal $w^{(2)}$-weight of a shortest $w^{(1)}$-weighted path is at most n^2LW_2. Similarly, the maximal $w^{(1)}$-weight of a shortest $w^{(2)}$-weighted path is at most n^2LW_1. The rest follows from Corollary 4.4.1. □

Unfortunately, we cannot bound the approximate optimal slope $\hat{v}$. Indeed, it is possible that the absolute value of the slope of the optimal hull segment returned by the hull approach is much larger than the slope of the guaranteed optimal hull segment, resulting in an even more pessimistic bound (see Figure 4.12).
However, we can also give a bound using the continuous hull that is independent of the outcome of the hull approach:

Lemma 4.4.4: Let $p_0, p_1, \ldots$ denote the sequence of points on the continuous lower hull between the extreme point with respect to resource and the extreme point with

respect to cost in nondecreasing resource coordinate. Let p_{i^*} denote the point with smallest cost coordinate so that $r_{p_{i^*}} + LW_2 < \lambda$ and let v^* denote the slope of the hull segment $\overline{p_{i^*}p_{i^*+1}}$.

Then the following holds:

$$LB^{'} \leq LB + L(W_1 - v^* W_2).$$

Proof. See Figure 4.12. □

The preceding Lemma implies that our approximations cannot be too bad even though we don't know how good they are since this depends on the continuous hull. Therefore, we introduce a notion of *sensitivity*. Let

$$c(\lambda) = \min\{c_p : r_p \leq \lambda\}$$

be the optimum dependent on λ. This is a step function. Now the sensitivity $s(\lambda)$ is defined as

$$s(\lambda) = \max\{\frac{c(\lambda - d) - c(\lambda)}{d} : d > 0\}.$$

This is the absolute value of the lower hull slope. Hence, intuitively, the approximation $LB^{'}$ is bad when the optimal solution to the problem is very sensitive to small changes of the resource limit λ. However, it also depends on LW_2 since if this is large then v^* might be much steeper then the optimal lower hull slope.

Other Weight Functions: So far, the only difference between our cost and resource function is the different face weight, whereas the euclidean length is in both functions. We give two examples for an alternative resource function:

- **Constant resources on the faces:** The scaled costs of a path segment s running through face f are now $\tilde{c}_s = \ell_s w_f^{(1)} - v w_f^{(2)}$. Using the same arguments as in Claim 1 we can show that there exists an approximating segment s' in the discretized graph so that

$$\tilde{c}_{s'} \leq \tilde{c}_s + \frac{L}{m+1} w_f^{(1)}$$

$$c_{s'} \leq c_s + \frac{L}{m+1} w_f^{(1)}$$

$$r_{s'} = r_s$$

  In Corollary 4.4.2, the approximation of LB then turns into $LB^{'} \leq LB + LW_1$ hence is independent of $\hat{v}$ and W_2.

- **Weighted height difference as resource:** The scaled costs of a path segment s running through face f are now $\tilde{c}_s = \ell_s w_f^{(1)} - vhd_s w_f^{(2)}$, where hd_s is the absolute value of the height difference of segment s. Since the height differences cannot exceed the length of a segment we can use the same arguments as in Claim 1 and obtain the result that there is an approximating segment s' with

$$\tilde{c}_{s'} \leq \tilde{c}_s + (w_f^{(1)} - vw_f^{(2)}) \frac{L}{m+1}.$$

  Hence, the bounds of Corollary 4.4.2 are also valid for this resource function.

4.4.3 Experiments

We have implemented the fixed scheme of Lanthier, Maheshwari, and Sack (2001) and provide it as an (approximate) network optimization function. The implementation is a Dijkstra variant and deals implicitly with the discretized graph, the nodes are represented by a pair (edge, Steiner point number), and the Steiner edges are also computed "online". This was suggested by Lanthier, Maheshwari, and Sack (2001) since for a larger number of Steiner points, space will become a problem.

Now we can use our CNOP package[14] providing the generic hull approach to compute an approximation for the dual relaxation. Using a k-shortest path algorithm working on the discretized graph[15], we can also obtain the optimum on the discretized graph that provides an approximation to the optimum of CGSP.

We show by example how the approximation of the hull improves as we increase the number of Steiner points per edge. The underlying polyhedral terrain is a triangulated irregular network (TIN) being a part of the Swiss Alps. We use unweighted euclidean length as cost function and slope-weighted height difference as resource function (see Figure 4.14 for the minimum cost and minimum resource path).

Here are the TIN details: The TIN consists of 369 faces, the maximum and average euclidean edge lengths are $L = 1499.47$ and $L_{avg} = 151.58$. Since the cost function is unweighted, we have $w_f^{(1)} = 1$ for all f, hence $W_1 = 1$. The maximum and average height difference of an edge are $HD = 1435.4$ and $HD_{avg} = 89.81$. The maximal and average slope of a face is $W_2 = 69.12$ and $W_{2_{avg}} = 2.84$. We encountered between 22 and 73 path segments for a path from s to t.

Let us take a look at the refinement of the approximation of the whole lower hull for an increasing number of Steiner points per edge starting with no Steiner points (see Figure 4.13). We observe that the approximation for 12 Steiner points does not improve significantly on the 8 Steiner points approximation, especially for smaller absolute slope

[14] Refer to Chapter 5 for a detailed discussion.

[15] or a modification working implicitly as suggested above

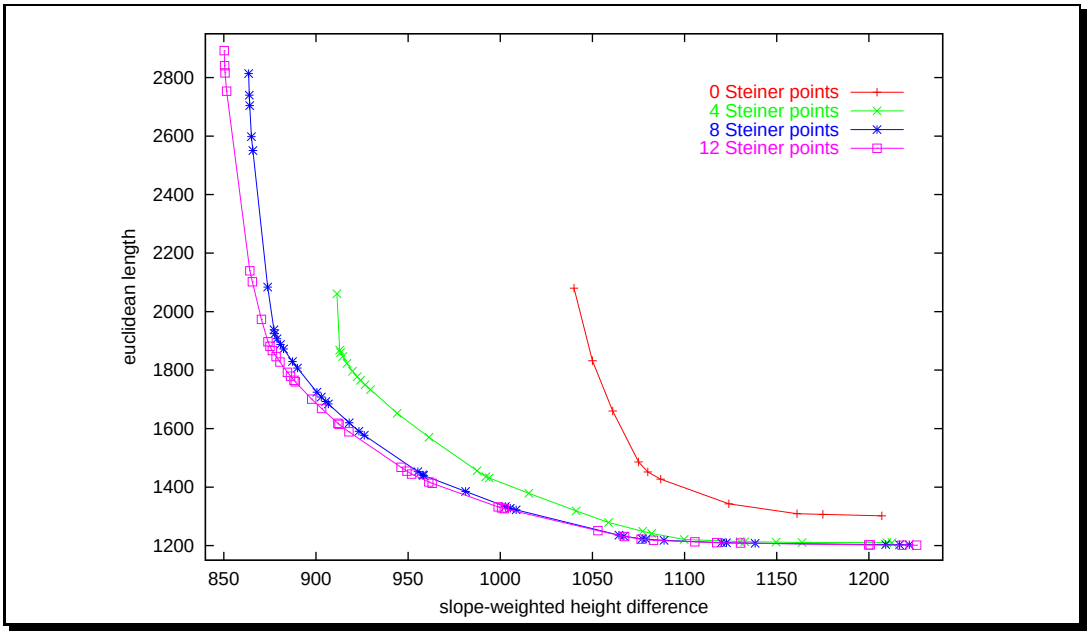

Figure 4.13: Refining the hull approximation by increasing the number of Steiner points per edge.

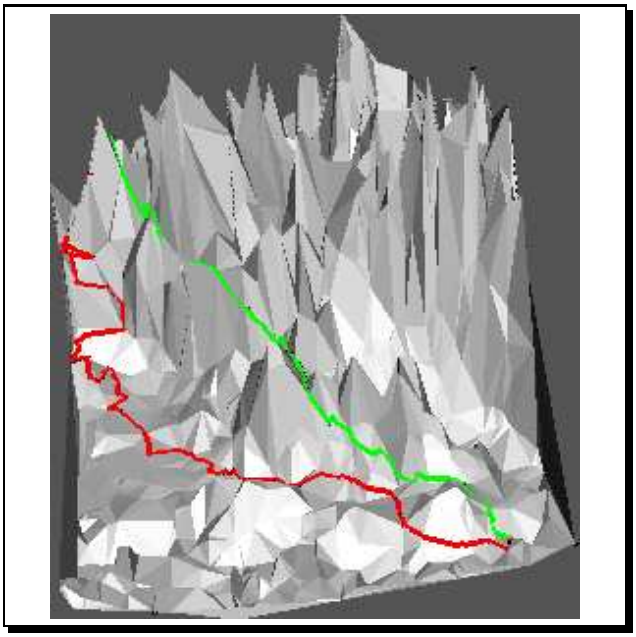

Figure 4.14: Euclidean shortest path (green) and path of minimum slope-weighted height difference (red) in a TIN.

# Steiner points	0	4	8	12
# hull points	10	28	40	42
time/extreme point	0.02s	0.91s	2.30s	4.28s

Table 4.7: Number of hull points and time for extreme point computation for an increasing number of Steiner points per edge.

values. So we expect that the exact continuous hull is approximated reasonably well by the 12 Steiner points hull, even though the guaranteed approximation bound is much worse. The reason for the varying difference in the hull refinement for varying slopes is that we have an unweighted problem in the cost coordinate and a weighted problem in the resource coordinate.

Table 4.7 shows what we have to pay for the refined approximation of the hull. We see that there is a large trade-off between approximation quality and running time[16]. We also observe that the number of hull points increases drastically from the simple computation without Steiner points to the computations with a larger number of Steiner points. This can be explained by the fact that although the continuous hull is a step function, it contains portions of continuity, hence a good approximation should reflect this.

4.4.4 An ϵ-δ-Approximation Scheme

CGSP is $\mathcal{NP}$-hard (since even the discrete version CSP is) and we present an ϵ-δ-approximation scheme for CGSP that is a simple combination of the ϵ-schemes of Aleksandrov, Maheshwari, and Sack (2000) and Hassin (1992).

Aleksandrov, Maheshwari, and Sack (2000) present an algorithm that approximates the geodesic (weighted) length of a path p by a path $\tilde{p}$ using a discretization of P so that $c(\tilde{p}) \leq (1+\epsilon)c(p)$. Their approximation algorithm runs in $O(n/\epsilon \log 1/\epsilon(1\sqrt{\epsilon} + \log n))$ while producing a discretized graph $G_\epsilon = (V(G_\epsilon), E(G_\epsilon))$ of size $|V(G_\epsilon)| = \frac{n}{\epsilon}\log\frac{1}{\epsilon}$ and $|E(G_\epsilon)| = \frac{n}{\epsilon}\frac{1}{\epsilon}\log\frac{1}{\epsilon}$.

We now use their algorithm to discretize the graph giving an ϵ_c-approximation for the cost-weighted geodesic length, then we apply their algorithm again to obtain an ϵ_r-approximation for the resource-weighted geodesic length. The two discretized graphs are merged into a graph $G_{\epsilon_c,\epsilon_r} = (V_{\epsilon_c,\epsilon_r}, E_{\epsilon_c,\epsilon_r})$ so that for every path p in $\mathcal{P}$ of cost $c(p)$ and resource $r(p)$ there is a path $\tilde{p}$ in $G_{\epsilon_c,\epsilon_r}$ with $c(\tilde{p}) \leq (1+\epsilon_c)c(p)$ and $r(\tilde{p}) \leq (1+\epsilon_r)r(p)$.

[16]This trade-off can be improved using the buffer heuristic described in Section 4.4.1.

Now we apply the ϵ-scheme of Hassin (1992) for CSP to the discretized graph $G_{\epsilon_c,\epsilon_r}$ with a modified resource limit $\tilde{\lambda} = (1+\epsilon_r)\lambda$. Hassin's scheme approximates the CSP optimal path p_{opt_G} in $G_{\epsilon_c,\epsilon_r}$ with $c(p_{opt_G}) = \min c(p)$ s.t. $r(p) \leq \tilde{\lambda}$ with a path p_{approx} that is an ϵ_H-approximation of CSP on $G_{\epsilon_c,\epsilon_r}$, *i.e.*, $c(p_{approx}) \leq (1+\epsilon_H)c(p_{opt_G})$ and $r(p_{approx}) \leq \tilde{\lambda}$.

Since the path p_{opt_G} is just an ϵ_c-approximation with respect to the cost and an ϵ_r-approximation with respect to the resource of the optimal path p_{opt_P} of CGSP, the path p_{approx} is an $(1+\epsilon_H)(1+\epsilon_c)$-$(1+\epsilon_r)$-approximation of the CGSP optimal path p_{opt_P} in the sense that $c(p_{approx}) \leq (1+\epsilon_H)(1+\epsilon_c)c(p_{opt_P})$ and $r(p_{approx}) \leq (1+\epsilon_r)\lambda$.

Theorem 4.4.1: There is an ϵ-δ-approximation scheme for CGSP approximating the optimal path p_{opt} with a path p_{approx} so that $c(p_{approx}) \leq (1+\epsilon)c(p_{opt})$ and $r(p_{approx}) \leq (1+\delta)\lambda$. The approximation can be computed in $O(\frac{n^2}{\epsilon(\epsilon+\delta)^2}\log^3(\frac{1}{\epsilon+\delta})\log\frac{UB}{LB})$ time where UB and LB are upper and lower bounds for GCSP.

Proof. We combine the ϵ-scheme of Aleksandrov, Maheshwari, and Sack (2000) and Hassin (1992) as described above. Just set $\epsilon = \epsilon_c\epsilon_H + \epsilon_c + \epsilon_H$ and $\delta = \epsilon_r$.
The running time is obtained as follows: The discretized graph $G_{\epsilon_c,\epsilon_r}$ is of size $|V(G_{\epsilon_c,\epsilon_r})| = \frac{n}{\epsilon_c+\epsilon_r}\log\frac{1}{\epsilon_c+\epsilon_r}$ and $|E(G_{\epsilon_c,\epsilon_r})| = \frac{n}{\epsilon_c+\epsilon_r}\frac{1}{\epsilon_c+\epsilon_r}\log\frac{1}{\epsilon_c+\epsilon_r}$.
The scheme of Hassin runs in $O(\frac{|V(G_{\epsilon_c,\epsilon_r})||E(G_{\epsilon_c,\epsilon_r})|}{\epsilon_H}\log\frac{UB}{LB}$ which is

$$O(\frac{1}{\epsilon_H}\frac{1}{\epsilon_c+\epsilon_r}\frac{n^2}{\epsilon_c+\epsilon_r}\log^3(\frac{1}{\epsilon_c+\epsilon_r})\log\frac{UB}{LB})$$

and can be simplified to

$$O(\frac{n^2}{\epsilon(\epsilon+\delta)^2}\log^3(\frac{1}{\epsilon+\delta})\log\frac{UB}{LB}).$$

□

4.5 Conclusion

In this chapter we have seen that the 2-step approach for constrained shortest paths can be extended to a generic method to solve general constrained network optimization problems.

This triggered simple exact algorithms for the constrained minimum spanning tree problem and the table layout problem. Both problems allow a strongly polynomial algorithm for solving the relaxation in the single resource case. However, our generic hull approach is much easier to implement and seems to be the method of choice in practice. Moreover, it also works for the multiple resource case.

We also obtained an approximate algorithm for the continuous constrained geodesic shortest path problem that has not been studied before. By combining two PTAS for discrete CSP and continuous geodesic paths, we also presented an ϵ-δ-approximation scheme for the constrained geodesic shortest path problem.

Chapter 5

CNOP - A Constrained Network Optimization Package

In this chapter we describe our software package CNOP, which provides a generic framework for constrained network optimization. CNOP is flexible, easy to use, and offers rich functionality. It provides all state of the art methods for constrained shortest paths, as well as the first publically available implementation for constrained minimum spanning trees. Moreover, it contains a large number of demo programs for route planning, curve approximation, table layout, etc.

5.1 Design of the CNOP Package

We decided to develop CNOP, a platform for constrained network optimization[1]. Our design goals were ease of use, flexibility, and a large functionality.

We implemented the package in C++ using LEDA (Mehlhorn and Näher 1999; Mehlhorn *et al.* 2001b). In the following presentation, we are often sloppy with giving the complete types for the interfaces of our functions to preserve readability. A more "technical" documentation can be found in the manual of the CNOP package (Ziegelmann 2001).

The main function of our package is

```
RESULT_TYPE cnop(G,s,t,cost,resource,limit, I);
```

[1] CNOP stands for Constrained Network Optimization Package.

The first 6 parameters contain the problem description. Here `G` is the graph of the underlying network problem, `s` and `t` are nodes, `cost` is the cost function defined on the edges, `resource` is the resource function defined on the edges and `limit` is the resource limit. In the multiple resource case, `resource` is an array of resource functions on the edges and `limit` is an array of resource limits.

All the flexibility is encapsulated in the choice class `I` whose methods are explained below. To make use of the generic 2-step approach for constrained network optimization, the user has to specify two generic functions, `netopt` and `ranking`.
The function `netopt` is used as core routine by the relaxation methods to solve the separation problem. It is of the general form

```
list<edge> netopt(G,s,t,cost,resource, scalevector, c, newpoint);
```

where `scalevector` specifies how to scale the new weight function for the network optimization (it contains the absolute values of the dual variables v_i $(1 \leq i \leq k)$). The result of the network optimization with the scaled weight is returned as a list of edges, its optimal value in `c`, and `newpoint` is the resulting extreme point, *i.e.*, its coordinates are cost value and resource value(s) of the solution.
This function can be specified with the method `set_network_optimization(netopt)` of the choice class `I`. Default is a dummy function. When the function `netopt` is specified, CNOP solves the relaxation of the constrained network optimization problem.

To close the possible duality gap, the user also has to specify the function `ranking` which has to be of the form

```
bool ranking(G,s,t,cost,resource,limit, scaledcost, scalevector, UB,
                                    LB, UBsol);
```

where `scaledcost` is the new weight function scaled with `scalevector` leading to the optimum of the relaxation, `UB` and `LB` are upper and lower bounds, and `UBsol` is the feasible solution resulting from the relaxation.
Now the `ranking` function should rank the solutions with respect to weight function `scaledcost` and update the bounds `UB, LB` and the best feasible solution `UBsol` until the optimum is found (return value `true`) or nonfeasibility is detected (return value `false`).
This function can be specified for the gap closing step with the choice method `set_gap_close_approach(ranking)`. Default is a dummy function.

A specification of these two functions enables CNOP to perform the 2-step method to obtain the optimum of the underlying constrained network optimization problem.
A user has many other possibilities to adapt the generic process to his or her own needs:

The CNOP package offers different methods for solving the relaxation of a constrained network optimization problem:

- `MZhull = 0`, the hull approach that computes the exact relaxation result. The implementation for the single resource case is trivial. The basis update for the multiple resource case uses the implementation of the dynamic $k+1$-dimensional convex hull algorithm of Clarkson, Mehlhorn, and Seidel (1993) that is offered by the `dd-geokernel` LEP (Mehlhorn, Näher, and Seel 2001a).

- `MZcplex = 1`, the hull approach that computes the exact relaxation result. The basis update is done by calling CPLEX (CPLEX 2001).

- `BC = 2`, the subgradient method of Beasley and Christofides (1989) that computes an approximation of the relaxation.

- `BS = 3`, the binary search method that computes the exact relaxation result (only in the single resource case).

Apart from the binary search approach, all other methods also work in the multiple resource case. However, the hull approach for multiple resources needs either the `dd-geokernel` LEP (`MZhull`) or the CPLEX library (`MZcplex`), only the subgradient method works without additional software.
The default is `MZhull` for the single resource case and `BC` for the multiple resource case. To change the relaxation approach, we use the choice method `set_rel_method(int m)`.

It is also possible to enforce a premature break in the relaxation approaches to directly control the running time. Using the method `set_max_it(int it)`, it is possible to stop the iteration after a specified number of iterations. Using the method `set_break_delta(double d)`, it is possible to stop the iteration when the slope difference[2] is less than a given limit delta. Default is 50 iterations and a delta of 0.
The method `set_do_relax(bool b)` can be used to switch the relaxation computation on or off (default is relaxation turned on).

Solving the relaxation in general does not yield the optimal solution, therefore a gap closing step is necessary. This step is turned on or off with the method `set_gap_close(bool b)` (default is gap closing turned on).
It is also possible to enforce a premature break in the gap closing approach since the number of solutions to be ranked may be huge. Using the method `set_gc_max_num(int n)`, it is possible to stop the solution ranking in the gap closing approach after a specified number of iterations. Using the method `set_gc_delta(double d)`, it is possible

[2] *i.e.*, difference in the dual variables v_i

to stop the gap closing when the difference between the upper and lower bound is less than a given limit. Default is `n=-1` (meaning no iteration limit) and `d=0`.

It is also possible to provide a special function testing the feasibility of a solution (*e.g.*, the ability to integrate lower bounds on the resource consumption, etc.). This function should have the form

```
bool is_feasible(solution, solution_value, limit);
```

where `solution` is a list of edges specifying the solution, `solution_value` is an array of the cost-resource consumption of the solution, and `limit` is the resource limit(s). This function can be specified with the method `set_is_feasible(is_feasible)`. Default is a function that tests whether the given solution satisfies the given resource limit(s).

The return type of the result of our customized `cnop` function is the following quadruple:

```
typedef four_tuple<list<edge>, double, double, bool> RESULT_TYPE;
```

The first component returns the solution in a list of edges, the second gives the value of the best solution that was found. The third component is the lower bound value. If this is greater or equal to the second component, we have found the optimal solution, otherwise we only have an approximate solution. The fourth component returns `false` if the problem was detected to be nonfeasible[3].

In addition to the generic `cnop` function, CNOP offers a function `whole_hull` that computes all extreme Pareto-optimal solutions[4] for constrained network optimization problems in the single resource case. The function has the general form

```
list<point_solution_pair> whole_hull(G,s,t,cost,resource, netopt);
```

where `point_solution_pair` denotes the list of edges and the cost-resource value of a solution.

Example:

```
#include <CNOP/cnop.h>

list<edge> s_t_min_cut(const graph& G, const node& s, const node& t,
```

[3]In the case of a premature break when no feasible solution has yet been found, it will return `true` since we do not know whether there is a feasible solution.

[4]geometrically speaking, the whole lower hull

```
  const edge_array<double>& cost, const edge_array<double>& resource,
  const double& slope,  double& c, array<double>& sol_val) {
  ...
}

int main() {
  ...

  choice I;
  I.set_gap_close(false);
  I.set_network_optimization(s_t_min_cut);
  RESULT_TYPE res=cnop(G,s,t,cost,resource,res_limit, I);
  list<edge> sol=res.first();
  double UB=res.second();
  double LB=res.third();
  return 0;
}
```

Here we define a function solving the minimum cut problem. Then, the hull approach is used to obtain upper and lower bounds for the constrained minimum cut problem. The gap closing step is turned off.

We have used the generic `cnop` function to compute the relaxation of the table layout problem and of the constrained geodesic shortest path problem. In the first case we provided a special minimum cut function solving the weighted sum minimization problem (see Section 4.3.3) as core function `netopt`. In the second case, the core function `netopt` was provided by an implementation of the approximate Steiner scheme for weighted geodesic paths (see Section 4.4.1). Both problems are available as demo programs in the CNOP package.

5.2 Special Case: Constrained Shortest Paths

Since our focus in constrained network optimization problems was on constrained shortest paths (see Chapter 3), we offer a special function `csp` for single and multiple resource CSP. The function `csp` is of the same form as `cnop`, all the choice methods also apply. The main difference is that we already provide functions for `netopt` and `ranking`.

We can use the shortest path implementations offered by LEDA as core algorithm `netopt` for the relaxation methods. We offer

- `shortest_path_DIJKSTRA`, the bidirectional Dijkstra variant of LEDA,
- `shortest_path_BELLMAN_FORD`, LEDA's Bellman-Ford implementation that also handles negative edge costs (provided that there are no negative cycles),
- `shortest_path_PAPE`, a shortest path implementation of Pape's algorithm,
- `shortest_path_ACYCLIC`, a pulling implementation for acyclic graphs.

The default network optimization function is `shortest_path_DIJKSTRA`. Of course, a user may also choose his or her own shortest path implementation.

We also offer several methods for the gap closing step, *i.e.*, the function `ranking`:

- `k_shortest_path_rea`, an implementation of the k-shortest path algorithm of Jimenez and Marzal (1999),
- `k_shortest_path_rea_loopless`, an adaption of the preceding implementation that only considers loopless paths,
- `labeling_approach_lb_cost`, labeling algorithm starting from lower bound costs,
- `label_correcting`, label correcting algorithm,
- `labeling_approach_cost`, labeling algorithm starting from cost,
- `labeling_approach_resource`, labeling algorithm starting from resource,
- `dynamic_programming_cost`, dynamic programming algorithm starting from cost,
- `dynamic_programming_resource`, dynamic programming algorithm starting from resource.

The first five approaches also work in the multiple resource case, the last three methods are only available in the single resource case. For a more detailed explanation of the different gap closing methods refer to Section 3.4.2. The default gap closing function is `labeling_approach_lb_cost`. Of course, a user may also provide his or her own gap closing method.

In addition to the general framework, we also offer problem reduction methods as described in Section 3.4.1. Nodes and edges of the graph are removed whose inclusion in a path would force the resource consumption over the resource limit or the cost over the upper bound on the optimal solution. They can be switched on or off (default) by the user with the choice methods `set_res_red(bool b)` and `set_len_red(bool b)`.

Example:

```
#include <CNOP/csp.h>

int main() {
  choice I;
  RESULT_TYPE res1=csp(G,s,t,cost,resource,upper_bound, I);

  I.set_do_relax(false);
  I.set_gap_close_approach(label_correcting);
  I.set_res_red(true);
  RESULT_TYPE res2=csp(G,s,t,cost,resource,upper_bound, I);

  return 0;
}
```

The first call of `csp` uses the hull approach to solve the relaxation, no problem reductions and the labeling approach starting from the lower bound costs. The second call does not solve the relaxation but performs resource reductions before applying the label correcting approach (see Section 3.4.2).

The CNOP package enables a user to experiment with all the known "state of the art" methods for CSP, trying out various combinations to see which setting fits best for a particular application. Indeed, we also used CNOP as experimental platform for our experiments in Section 3.5. The route planning and curve approximation applications of CSP are also available as demo programs in the CNOP package.

5.3 Special Case: Constrained Minimum Spanning Trees

We also offer a special function `cmst` for the constrained minimum spanning tree problem (CMST) for single and multiple resources.

As core algorithm `netopt` for the relaxation methods, we can use LEDA's implementation of Kruskal's minimum spanning tree algorithm that runs in $O(m \log n)$. So we provide `min_spanning_treeLEDA` as default function for the relaxation.

For the gap closing step we implemented the spanning tree ranking algorithm of Katoh, Ibaraki, and Mine (1981)[5]. So we provide `k_min_spanning_tree` as default `ranking` function.

[5] We implemented only the simpler $O(Km)$ space variant.

As in the constrained shortest path case, it is possible to perform problem reductions (see Section 4.2.5). We again provide functions for the problem reductions in the constrained spanning tree case[6]. They can be turned on or off (default) by the user as before.

Example:

```
#include <CNOP/cmst.h>

int main() {
  choice I;
  RESULT_TYPE res1=cmst(G,s,t,cost,resource,upper_bound, I);

  I.set_do_relax(false);
  I.set_res_red(true);
  RESULT_TYPE res2=cmst(G,s,t,cost,resource,upper_bound, I);

  return 0;
}
```

The first call of `cmst` uses the hull approach to solve the relaxation, no problem reductions, and the k minimum spanning tree approach starting from the lower bound costs. The second call does not solve the relaxation but performs resource reductions before applying the k minimum spanning tree algorithm.

To the best of our knowledge, this is the first publically available implementation for the constrained minimum spanning tree problem. Only Aggarwal, Aneja, and Nair (1982) report about an implementation that is not publically available and deals only with the single resource problem.

5.4 CNOP Availability

We have seen that our CNOP package provides a generic platform for constrained network optimization problems. It is very flexible and adaptable to the user's needs. Moreover, it offers rich functionality in the area of constrained shortest paths and constrained minimum spanning trees, as well as many demo programs for applications that we have discussed in this thesis.

[6]However, our simplified implementation uses $O(m^2)$ time, instead of the $O(m\alpha(m,n))$ variant presented in Section 4.2.5.

CNOP is tested for GNUs `g++` compiler version `2.95.2` under Solaris and Linux using LEDA 4.3. CNOP will become a LEDA extension package (LEP) in the near future. A more detailed documentation including installation information can be found in the CNOP manual (Ziegelmann 2001). The CNOP package including demos and testdata can be downloaded from the URL

`http://www.mpi-sb.mpg.de/~mark/cnop`

The first version of CNOP had more than 50 downloads from academic, commercial, and military sites in the period January–July 2001.

Chapter 6

Discussion

In this thesis, we have studied constrained shortest paths and related network optimization problems with resource constraints. These problems are motivated by a large number of practical applications but usually are $\mathcal{NP}$-hard as opposed to their unconstrained counterparts that can be efficiently solved with polynomial time algorithms.

Our main goal was to come up with efficient practical algorithms to solve such problems exactly or at least approximately.

We studied the constrained shortest path problem in Chapter 3. Using a new ILP formulation, we combined geometric intuition and linear programming theory to derive a simple combinatorial approach to solve the Lagrangean relaxation of CSP. The method was previously known in the single resource case but we were the first to prove a tight polynomial running time. We were also the first to extend the method to solve the multiple resource relaxation exactly. The running time for the multiple resource case is still open but we proposed a conjecture that is supported by our experiments.
We obtained an exact 2-step algorithm for CSP by closing the duality gap with path ranking that was suggested earlier for the single resource case. We also proposed a new gap-closing method, an adapted labeling approach that makes use of the upper and lower bounds of the relaxation.
A detailed experimental study showed that the 2-step approach using our new gap closing approach is a competitive method for all considered test instances while often even being clearly superior to other state of the art methods.

In Chapter 4 we showed that a 2-step method, solving the relaxation with our hull approach and closing the gap with solution ranking, also works for the general con-

strained network optimization problems.
Subsequently, we illustrated the general method using three examples: constrained minimum spanning trees, table layout, and constrained geodesic shortest paths. In these examples we compared the application of our generic method with previous work that considered the specific problem structure.

A generic framework for constrained network optimization problems is provided by our CNOP package that we presented in Chapter 5. The package combines ease of use, flexibility, and rich functionality. A user may chose between different relaxation methods, and provide own network optimization and solution ranking implementations to obtain an approximate or exact method for the considered application as we have demonstrated with the table layout problem and the constrained geodesic shortest path problem, which are available as demo programs. CNOP is the first publically available package that offers all state of the art methods for constrained shortest paths and can be used as a testbed to see which approach is most suited for a special CSP application. It also offers the first publically available implementation for the constrained minimum spanning tree problem.

The generic 2-step approach is a simple and elegant technique and a variety of applications can be modeled as constrained network optimization problems. However, we sometimes also pay for the genericity since some special problems that are easily formulated as constrained network optimization problems might allow a different, more efficient solution technique. For example, we can formulate the degree constrained minimum spanning tree problem, where we want to compute a spanning tree of minimum cost so that each node in the tree has maximal degree D, as resource constrained minimum spanning tree problem with a resource for each node. But this does not seem to be a promising method for the problem. The same deficit applies if we use a modified function for testing the feasibility of a solution in order to incorporate additional structural constraints (like requiring that a constrained shortest path also passes through a specified set of nodes) into the 2-step method. If a problem has special cost or resource functions (*e.g.*, only having cost/resource values of 0 or 1) it might also be more promising to look for a special solution method for the particular case. However, the CNOP package helps us to identify the cases where we should look for other, more efficient methods, *e.g.*, if the gap closing step takes too much time or if the relaxation bounds are bad.

Bibliography

Aggarwal, V., Aneja, Y., and Nair, K. 1982. Minimal spanning tree subject to a side constraint. *Computers and Operations Research* 9, 287–296.

Ahuja, R., Magnanti, T., and Orlin, J. 1993. *Network Flows*. Prentice Hall.

Aleksandrov, L., Maheshwari, A., and Sack, J.-R. 2000. Approximation algorithms for geometric shortest path problems. In: *32nd ACM Symposium on Theory of Computing (STOC)*, 286–295.

Anderson, R., and Sobti, S. 1999. The table layout problem. In: *Proc. 15th Symposium on Computational Geometry (SoCG)*, 115–123.

Aneja, Y., Aggarwal, V., and Nair, K. 1983. Shortest chain subject to side conditions. *Networks* 13, 295–302.

Aneja, Y., and Nair, K. 1978. The constrained shortest path problem. *Nav. Res. Log. Q.* 25, 549–553.

Azevedo, J., Madeira, J., and Martins, E. 1993. An algorithm for the ranking of k shortest paths. *European Journal on Operational Research* 69, 97–106.

Beasley, J., and Christofides, N. 1989. An algorithm for the resource constrained shortest path problem. *Networks* 19, 379–394.

Bertsimas, D., and Tsitsiklis, J. 1997. *Linear Optimization*. Athena Scientific.

Borndörfer, R., and Löbel, A. 2001. Scheduling duties by adaptive column generation. Tech. Rep. ZIB-01-02, Konrad-Zuse-Zentrum für Informationstechnik Berlin.

Brumbaugh-Smith, J., and Shier, D. 1989. An empirical investigation of some bicriterion shortest path algorithms. *European Journal on Operational Research* 43, 216–224.

Chen, J., and Han, Y. 1996. Shortest paths in a polyhedron. *International Journal of Computational Geometry and Applications* 127–144.

Chvátal, V. 1983. *Linear Programming*. W.H. Freeman and Company.

Clarkson, K., Mehlhorn, K., and Seidel, R. 1993. Four results on randomized incremental construction. *Computational Geometry: Theory and Applications* 3(4), 185–212.

Cormen, T., Leiserson, C., and Rivest, R. 1993. *Introduction to algorithms*. MIT Press.

CPLEX. 2001. *Using the CPLEX callable library*. CPLEX Optimization, Inc., `http://www.cplex.com`.

Dahl, G., and Realfsen, B. 2000. The cardinality-constrained shortest path problem in 2-graphs. *Networks* 36(1), 1–8.

Desrochers, M., and Soumis, F. 1988. A generalized permanent labeling algorithm for the shortest path problem with time windows. *INFORMS* 26, 191–212.

Desrosiers, J., Dumas, Y., Solomon, M., and Soumis, F. 1995. *Time constrained routing and scheduling*, vol. 8 of *Handbooks in Operations Research and Management Science: Network Routing*, chap. 2.4 Constrained shortest path problems, 70–80. Elsevier Science, Netherlands.

Dumitrescu, I., and Boland, N. 2000. The weight-constrained shortest path problem: preprocessing, scaling and dynamic programming algorithms with numerical comparisons. In: *International Symposium on Mathematical Programming (ISMP)*.

Elimam, A., and Kohler, D. 1997. Two engineering applications of a constrained shortest path model. *European Journal of Operational Research* 103, 426–438.

Eppstein, D. 1990. Finding the k smallest spanning trees. In: *Proc. 2nd Scandinavian Worksh. Algorithm Theory*, no. 447 in Lecture Notes in Computer Science, 38–47, Springer-Verlag.

Eppstein, D. 1999. Finding the k shortest paths. *SIAM Journal on Computing* 28(2), 652–673.

Gallo, G., Grigoriadis, M., and Tarjan, R. 1989. A fast parametric maximum flow algorithm and applications. *SIAM Journal on Computing* 18(1), 30–55.

Garey, M., and Johnson, D. 1979. *Computers and Intractibility: A Guide to the Theory of NP-Completeness*. W. H. Freeman, New York.

Hamacher, H., Picard, J., and Queranne, M. 1984. On finding the k-best cuts in a network. *Operations Research Letters* 2(6), 303–305.

Hamacher, H., and Ruhe, G. 1994. On spanning tree problems with multiple objectives. *Annals of Operations Research* 52, 209–230.

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

Hansen, P. 1980. Bicriterion path problems. In: G. Fandel, and T. Gal, eds., *Multiple Criteria Decision Making: Theory and Application*, 109–127, Springer verlag, Berlin, Proceedings of a Conference held at Hagen, Germany, 1979.

Har-Peled, S. 1998. An output sensitive algorithm for discrete convex hulls. *Computational Geometry: Theory and Applications* 10, 125–138.

Hassin, R. 1992. Approximation schemes for the restricted shortest path problem. *Math. Oper. Res.* 17(1), 36–42.

Henig, M. 1985. The shortest path problem with two objective functions. *European Journal of Operational Research* 25, 281–291.

Holmberg, K., and Yuan, D. 1997. A multicommodity network flow problem with side constraints on paths solved by column generation. In: *International Symposium on Mathematical Programming (ISMP).*

Jahn, O., Möhring, R., and Schulz, A. 1999. Optimal routing of traffic flows with length restrictions in networks with congestion. Tech. Rep. 658-1999, TU Berlin.

Jimenez, V., and Marzal, A. 1999. Computing the K shortest paths. A new algorithm and an experimental comparison. In: *Proc. 3rd Workshop on Algorithm Engineering (WAE99)*, LNCS 1668, 15–29, Springer, Berlin.

Joksch, H. 1966. The shortest route problem with constraints. *Journal of Mathematical Analysis and Application* 14, 191–197.

Karp, R., and Orlin, J. 1981. Parametric shortest path algorithms with an application to cyclic scheduling. *Discrete Applied Mathematics* 3, 37–45.

Katoh, N., Ibaraki, T., and Mine, H. 1981. An algorithm for finding k minimum spanning trees. *Siam Journal on Computing* 10(2), 247–255.

Katoh, N., Ibaraki, T., and Mine, H. 1982. An efficient algorithm for k shortest simple paths. *Networks* 12, 411–427.

Lanthier, M., Maheshwari, A., and Sack, J.-R. 2001. Approximating shortest paths on weighted polyhedral surfaces. *Algorithmica* (to appear), preliminary version in SoCG97.

Lauther, U. 2001. personal communication.

Lawler, E. 1972. A procedure for computing the k best solutions to discrete optimization problems and its application to the shortest path problem. *Management Science B* 18, 401–405.

Lawler, E. 1976. *Combinatorial Optimization: Networks and Matroids*. Holt, Rhinehart and Winston.

Lübbecke, M., and Zimmermann, U. 2000. Computer aided scheduling of switching engines. In: *CASPT2000*.

Martins, E. 1984. On a multicriterion shortest path problem. *European Journal of Operational Research* 16, 236–245.

Martins, E., and Santos, J. 1996. A new shortest paths ranking algorithm. Tech. rep., Univ. de Coimbra, `http://www.mat.uc.pt/~eqvm`.

Megiddo, N. 1983. Applying parallel computation algorithms in the design of serial algorithms. *Journal of the ACM* 30, 852–865.

Mehlhorn, K., and Näher, S. 1999. *LEDA - A platform for combinatorial and geometric computing*. Cambridge University Press.

Mehlhorn, K., Näher, S., and Seel, M. 2001a. *The Geokernel LEP User Manual Version 2.1*. Max-Planck-Institut für Informatik, `http://www.mpi-sb.mpg.de/LEDA/friends/dd_geokernel.html`.

Mehlhorn, K., Näher, S., Seel, M., and Uhrig, C. 2001b. *The LEDA User Manual*. Max-Planck-Institut für Informatik, `http://www.mpi-sb.mpg.de/LEDA`.

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

Mehlhorn, K., and Ziegelmann, M. 2001. CNOP - A package for constrained network optimization. In: *3rd Workshop on Algorithm Engineering and Experiments (ALENEX), LNCS 2153*, 15–28.

Miaou, S., and Chin, S. 1991. Computing k-shortest paths for nuclear spent fuel highway transportation. *European Journal on Operational Research* 53, 64–80.

Minoux, M., and Ribero, C. 1984. A transformation of hard (equality constrained) knapsack problems into constrained shortest path problems. *Operations Research Letters* 3(4), 211–214.

Minoux, M., and Ribero, C. 1985. A heuristic approach to hard constrained shortest path problems. *Discrete Applied Mathematics* 10, 125–137.

Minoux, M., and Ribero, C. 1986. Solving hard constrained shortest path problems by Lagrangean relaxation and branch-and-bound algorithms. *Methods of Operations Research* 53, 304–316.

Modesti, P., and Sciomachen, A. 1998. A utility measure for finding multiobjective shortest paths in urban multimodal transportation networks. *European Journal on Operational Research* 111, 495–508.

Mote, J., Murthy, I., and Olson, D. 1991. A parametric approach to solving bicriterion shortest path problems. *European Journal on Operational Research* 53, 81–92.

Müller-Hannemann, M., and Weihe, K. 2001. Pareto shortest paths is often feasible in practice. In: *5th Workshop on Algorithm Engineering (WAE)*, to appear.

Nygaard, R. 2000. *Shortest Path Methods in Representation and Compression of Signals and Image Contours*. Ph.D. thesis, Norwegian University of Science and Technology.

Orda, A. 1998. Routing with end to end QoS guarantees in broadband networks. In: *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, 27–34.

Pape, U. 1974. Implementation and efficiency of Moore-algorithms for the shortest route problem. *Mathematical Programming* 7, 212–222.

Phillips, C. 1993. The network inhibition problem. In: *25th ACM Symposium on Theory of Computing (STOC)*, 776–785.

Preparata, F. 1978. New parallel sorting schemes. *IEEE Trans. Comput.* C-27, 669–673.

Pujari, A., Agarwal, S., and Gulati, V. 1984. A note on the constrained shortest path problem. *Nav. Res. Log. Q.* 31, 87–94.

Ravi, R., and Goemans, M. 1996. The constrained minimum spanning tree problem. In: *Proc. 5th Scandinavian Workshop on Algorithmic Theory (SWAT)*, LNCS 1097, 66–75.

Roessl, M. 1968. Comments on a paper of R. Saigal, a constrained shortest route problem. *Operations Research* 16, 1232–1234.

Saigal, R. 1968. A constrained shortest route problem. *Operations Research* 16, 205–209.

Schrijver, A. 1986. *Theory of Linear and Integer Programming*. John Wiley.

Skicism, C., and Golden, B. 1989. Solving k-shortest and constrained shortest path problems efficiently. *Annals of Operations Research* 20, 249–282.

Sleator, D., and Tarjan, R. 1983. A data structure for dynamic trees. *Journal of Computer and System Sciences* 26, 362–391.

Stroetmann, K. 1997. The constrained shortest path problem: A case study in using ASMs. *Journal of Universal Computer Science* 3(4), 304–319.

Tarjan, R. 1979. A data structure for dynamic trees. *Journal of the ACM* 26, 690–715.

Tung, C., and Chew, K. 1992. A multicriteria pareto-optimal path algorithm. *European Journal on Operational Research* 62, 203–209.

Warburton, A. 1987. Approximation of pareto-optima in multiple-objective shortest path problems. *Operations Research* 35(1), 70–79.

Xue, G. 2000. Primal-dual algorithms for computing weight-constrained shortest paths and weight-constrained minimum spanning trees. In: *19th IEEE International Performance, Computing, and Communications Conference (IPCCC)*, 271–277.

Yen, J. 1971. Finding the K shortest loopless paths in a network. *Management Science* 17, 712–716.

Ziegelmann, M. 2001. *The* CNOP *Manual*. Max-Planck-Institut für Informatik, `http://www.mpi-sb.mpg.de/~mark/cnop`.

Summary

The shortest path problem is one of the fundamental problems in Computer Science. It is well studied and efficient polynomial time algorithms are known. Shortest path problems frequently arise in practice since in a variety of application settings we wish to send some material (*e.g.*, a computer data packet, a telephone call, or a vehicle) between two specified points in a network as quickly, as cheaply, or as reliably as possible.

However, in practice we are often not only interested in a cheapest path or a quickest path but rather in a combination of different criterias, *e.g.*, we want to have a path that is both cheap and quick. This is known as the bi- or multicriteria shortest path problem. Since optimizing over all criteria at once is not possible we choose one criteria as cost function that we want to minimize, the other as resource functions and impose resource (or budget) limits on the maximal resource consumption of a path. The *constrained shortest path problem* is to find a minimum cost path between two nodes whose resource consumptions satisfy the resource limits.

The constrained shortest path problem is of immense practical interest in different areas of operations research. Applications include route planning, quality of service routing, duty scheduling, route guidance, and even curve approximation.

Unfortunately, the introduction of even a single resource constraint turns the problem into a *hard* problem where we do not know a polynomial time algorithm to solve it. However, regarding its huge practical importance we would still like to solve the problem (or at least get an approximation) as efficiently as possible.

In this thesis we study the constrained shortest path problem both theoretically and experimentally. We also consider related problems, like constrained minimum spanning trees where we want to find a spanning tree of minimal cost while its resource consumptions satisfy the resource limits.

There is a variety of work on constrained shortest paths coming from different communities like operations research, algorithms, communication networks, and even signal processing. Almost all papers come up with essentially the same algorithm solving a relaxation of the problem for the single resource case. They only differ in the presentation of the method. Some derive it from geometric intuition, others adopt the Lagrangean relaxation viewpoint. Starting from a new ILP formulation of the problem, we combine geometric intuition and linear programming theory to obtain a unified understanding of the method. Using this combined view, we are the first to prove a tight polynomial

runtime bound for this method. We showed that the relaxation can be solved with $O(\log(nRC))$ parametric shortest path computations, where n is the number of nodes in the network and C and R denote the maximal cost and resource consumption of an edge, respectively. Our reformulation also allows us to extend the method to the multiple resource case, which has been an open problem up to now.

Solving the relaxation gives us upper and lower bounds for our problem. Previous papers suggested different gap closing steps to obtain a 2-step method for constrained shortest paths. We again give a geometric intuition of the gap-closing step and propose a special labeling approach for the gap-closing step.

Then we experimentally compare all state of the art methods for constrained shortest paths on different benchmarks. This is the first detailed experimental runtime comparison for constrained shortest paths.

We then show that the 2-step method can be generalized to a broader class of constrained network optimization problems. All we need is a function returning the unconstrained optimum and a function ranking solutions. The single resource runtime bound for the relaxation extends. We illustrate the generic method using three examples: constrained minimum spanning trees, table layout, and constrained geodesic shortest paths. In these examples we compare the application of our generic method with previous work that considered the specific problem structure.

We developed a software package CNOP that implements the generic 2-step approach. A user only has to specify a function solving the corresponding unconstrained problem and a function ranking problem solutions. Additionally, CNOP offers all state of the art methods for constrained shortest paths and can be used as a testbed to see which approach is most suited for a special application. While several implementations of different methods exist, this is the first package that makes all state of the art methods publicly available. It also offers the first publicly available implementation for the constrained minimum spanning tree problem. The flexibility of the CNOP package allows the user to experiment with other bi- or multicriteria network optimization problems.

Zusammenfassung

Das Kürzeste-Wege-Problem ist eine der fundamentalen Problemstellungen der Informatik. Es ist wohlstudiert und es existieren effiziente polynomielle Algorithmen. Kürzeste-Wege-Probleme treten oft in der Praxis auf, da wir in vielen Anwendungen daran interessiert sind, in einem Netzwerk etwas (ein Datenpaket, einen Telefonanruf oder ein Fahrzeug) von einem Startpunkt so schnell, billig oder zuverlässig wie möglich zu einem Zielpunkt zu senden.

Häufig sind wir in Anwendungen nicht nur an einem billigsten oder kürzesten Pfad interessiert, sondern vielmehr an einer Kombination verschiedener Kriterien. Wir suchen zum Beispiel einen Pfad, der sowohl billig als auch kurz ist. Dies ist als Kürzeste-Wege-Problem mit mehreren Kriterien bekannt. Da es nicht möglich ist, alle Kriterien gleichzeitig zu optimieren, wählen wir ein Kriterium als zu optimierende Kostenfunktion aus, die anderen Kriterien werden als Ressourcenfunktionen betrachtet, wobei wir obere Schranken für den maximalen Ressourcenverbrauch eines Pfades festlegen. Das *Kürzeste-Wege-Problem mit Nebenbedingungen* ist nun, einen Pfad mit minimalen Kosten zwischen zwei gegebenen Punkten zu finden, dessen Ressourcenverbrauch diese Schranken nicht übersteigt.

Die Lösung dieses Problems ist von großer praktischer Bedeutung. Anwendungen finden sich zum Beispiel auf den Gebieten der Routenplanung, dem sogenannten *quality of service routing*, der Dienstplangenerierung, der Routenführung und sogar der Kurvenapproximation. Leider macht schon eine zusätzliche Ressourcennebenbedingung aus dem "einfachen" Kürzeste-Wege-Problem ein "schweres" Problem, für das wir keinen polynomiellen Algorithmus kennen. Aufgrund der großen praktischen Relevanz wollen wir jedoch dieses Problem so effizient wie möglich lösen oder zumindest approximieren können.

In dieser Arbeit untersuchen wir das Kürzeste-Wege-Problem mit Nebenbedingungen sowohl theoretisch als auch praktisch. Wir behandeln auch verwandte Probleme, wie minimale Spannbäume mit Nebenbedingungen.

Es gibt eine Reihe von Arbeiten über Kürzeste-Wege mit Nebenbedingungen aus verschiedenen Bereichen der Informatik. Fast alle Arbeiten stellen im Prinzip denselben Algorithmus zum Lösen einer Relaxierung des Problems vor. Der einzige Unterschied liegt in der Präsentation der Methode. Die einen leiten das Verfahren allein von der geometrischen Intuition her, andere wenden allgemeine Techniken der Lagrange Relaxierung an. Wir kombinieren einfache geometrische Intuition mit der Theorie des

Linearen Programmierens und erhalten ein umfassendes Verständnis der Methode. Diese kombinierte Sichtweise erlaubt es uns, zum ersten Mal eine scharfe polynomielle Laufzeitschranke für das Verfahren zu zeigen. Wir beweisen, dass man die Relaxierung mit $O(\log(nRC))$ Kürzeste Wege Berechnungen mit skalierten Kosten lösen kann, wobei n die Zahl der Knoten des Netzwerks ist und C und R die maximalen Kosten beziehungsweise Ressourcen einer Kante darstellen. Unsere kombinierte Sichtweise erlaubt es uns auch, das Verfahren auf mehrere Nebenbedingungen zu erweitern, was bislang ein offenes Problem war.

Das Lösen der Relaxierung gibt uns eine obere und eine untere Schranke für unser Problem. Frühere Arbeiten schlugen verschiedene Ansätze zum Schließen dieser Lücke vor, um somit eine 2-Schritt-Methode zu erhalten. Wir präsentieren wieder eine kombinierte theoretische und geometrisch intuitive Sichtweise des zweiten Schrittes und stellen ein neues Labeling-Verfahren zum Schließen der Lücke vor.

Danach vergleichen wir alle gängigen und neu vorgeschlagenen Methoden für Kürzeste-Wege mit Nebenbedingungen experimentell. Dies ist der erste detaillierte experimentelle Vergleich verschiedener Lösungsverfahren.

Weiterhin zeigen wir, dass die 2-Schritt-Methode auf eine allgemeinere Klasse von Netzwerkoptimierungsprobleme mit Nebenbedingungen anwendbar ist. Wir benötigen nur eine Funktion, die das Netzwerkoptimierungsproblem ohne Nebenbedingung löst, sowie eine Funktion, die Lösungen in aufsteigender Kostenreihenfolge auflistet. Die Schranke für die Zahl der Iterationen zum Lösen der Relaxierung gilt weiter. Wir illustrieren diese generische Methode anhand von drei Beispielen: Minimale Spannbäume mit Nebenbedingungen, Tabellenlayout und Geodätische-Kürzeste-Wege mit Nebenbedingungen. Hierbei vergleichen wir die Anwendung unseres generischen Verfahrens mit früheren Arbeiten, die die jeweilige spezielle Struktur des Problems berücksichtigen.

Wir haben ein Softwarepaket CNOP entwickelt, das dieses generische Verfahren implementiert. Ein Benutzer muss nur eine Funktion zum Lösen des Problems ohne Nebenbedinungen und eine Funktion zum Auflisten von Lösungen bereitstellen. Darüber hinaus stellt CNOP alle gängigen Verfahren für das Kürzeste-Wege-Problem mit Nebenbedingungen zur Verfügung und kann somit als ideale Testumgebung dienen, um herauszufinden, welche Methode für eine spezielle Anwendung am geeignetsten ist. Es existieren zwar schon Implementierungen einiger Verfahren, aber CNOP macht zum ersten Mal alle gängigen Methoden öffentlich nutzbar. CNOP bietet ebenso die erste öffentliche Implementierung für das Minimale-Spannbaum-Problem mit Nebenbedingungen. Die Flexibilität des CNOP-Pakets ermöglicht es dem Benutzer, auch mit anderen Netzwerkoptimierungsproblemen mit Nebenbedingungen zu experimentieren.

Index