Fast Combinatorial Algorithms for Packing and Covering Problems

Diplomarbeit von Jochen Könemann

angefertigt in der Arbeitsgruppe von Prof. Dr. Kurt Mehlhorn
am Max-Planck-Institut für Informatik in Saarbrücken
innerhalb des Fachbereichs 14, Informatik, der Universität des Saarlandes.

Contents

Chapter 1

Introduction

In this thesis, we will examine several fractional combinatorial optimization problems. Given such a problem, there are several ways to solve them. All of the examined problems have a natural linear programming formulation and therefore, since we consider fractional problems, all of them are exactly solvable in polynomial time using one of the polynomial time algorithms for linear programming (c.f. [22]).

However, the framework of linear programming is a very general purpose method. While being advantageous for the reason of having one routine which is applicable to a large class of problems, the combinatorial structure of the treated problem is completely ignored. The goal of this thesis is to construct efficient algorithms for some optimization problems. The procedures compute solutions of provable quality to distinct problems, gaining advantage from the combinatorial structure of the problem itself. Thus, our algorithms will outperform general linear programming methods concerning running time while sacrificing optimality. Hence, our methods will be amenable to handle large sized instances of optimization problems.

Another issue of this work will be the granularity of the computed solutions. While exact solutions, achieved by the previously mentioned methods, often show a bad integrality, we will be able to prove worst case bounds on the granularity. Proving a good integrality is equivalent to stating simplicity of the computed solution. A simple solution structure is desirable not only from a theoretical point of view. In practice, obtaining simple solutions while being close to optimality is often reasonable.

Besides the mentioned aspects, our methods turn out to be very simple in description. It turns out that most of the algorithms work in a greedy fashion, i.e. their way of building a solution seems to be very natural. Therefore, we will emphasize the intuition when describing the features of our algorithms. Moreover, we are able to present a simple analysis which proves theoretical running time and lower bounds on the quality of the obtained solutions.

As mentioned, the quality of the solutions that our algorithms compute can be exactly specified. More precisely, we develop *approximation algorithms*. An approximation algorithm constructs a feasible solution to a given instance of an optimization problem while proving a lower bound on its value. For many applications, it is sufficient to deal with solutions which are close to optimum. The reason for this might for example be input data of fixed accuracy.

In particular, we will develop algorithms for *covering* and *packing* problems. A *covering* problem can be described by the linear program

$$\begin{aligned} \mathbf{min} \quad & c^T x \\ \mathbf{s.t.} \quad & Ax \geq b \\ & x \geq 0 \end{aligned} \tag{1.1}$$

where A is an $m \times n$ matrix and b and c are m and n dimensional vectors of rational numbers. The name *covering* arises from the intuition. Picking column j of A to an extent of 1 *covers* row i by a_{ij} units. The goal in a covering problem is to find a minimum cost selection of columns such that each row i is covered by b_i units.

A *packing* problem can be described by the following linear program

$$\begin{aligned} \mathbf{max} \quad & y^T b \\ \mathbf{s.t.} \quad & A^T y \leq c \\ & y \geq 0 \end{aligned} \tag{1.2}$$

where A, b and c are defined as in the previous case. Observe, that the latter problem is the dual of LP (1.1). We will later elaborate on the role of linear programming duality in the design of our algorithms. Again, the name of the problem (1.2) is guided by an intuitive argument: the vector b can be viewed as an assignment of benefits to the columns of A^T. We now aim for a selection of columns of A^T, such that their sum fits into c while the incurred benefit is maximized, i.e. we pack the rows of A^T into a box of volume c.

Before outlining the ideas of the general style procedures, we comment on special covering and packing problems which are important in their own right. One of the most studied problems, that fits into the framework of covering is *set-cover*. Here, we consider a universe $U = \{e_1, \ldots, e_m\}$ of elements and a collection $\mathcal{S} = \{S_1, \ldots, S_n\}$ of sets. With each set S, we associate a cost c_S. Similarly, we are given a coverage demand b_e for each element e. We are obliged to find a minimum cost subset $S \subseteq \mathcal{S}$, such that the coverage demand of each element e is met. In the fractional variant of the latter problem, we are also allowed to pick sets to rational extents.

Note, that we can formulate the fractional set-cover problem in the form (1.1). For each set $S \in \mathcal{S}$, we have an associated column $(a_{1S}, \ldots, a_{mS})^T$ in the matrix A. Similarly, we assign a row $(a_{e1}, \ldots, a_{en})$ of A to each of the elements e from universe U. In this case, Matrix A contains only entries from $\{0, 1\}$. A column $A_{\cdot S}$ can then be viewed as the incidence vector of set S since $A_{eS} = 1$ iff $e \in S$. Consequently, b_e will be the coverage demand of element e and c_S is defined to be the unit picking cost of set S.

An important problem, that fits into the packing framework is the *multicommodity flow* problem. Here, we are given a flow network $G = (V, E)$, a capacity function c on the edges and a set of terminal pairs $T = \{(s_1, t_1), \ldots, (s_k, t_k)\}$. We associate a commodity with each of the pairs from T. The multicommodity flow problem now involves shipping the different commodities simultaneously from their respective sources to their sinks in the network G so that the total amount of flow going through edge e is no more than $c(e)$.

The first variant of the latter problem is the *maximum multicommodity flow* problem. Here, we want to maximize the total flow leaving the sinks $s_1, \ldots, s_k$. In particular, there is no rule specifying the distribution of the total amount of flow over the different commodities, i.e. it might happen that the total flow is routed along one commodity.

We will also present an algorithm for the special case of the multicommodity flow problem in which we associate a demand d_i with each terminal pair (s_i, t_i). The demand d_i specifies the amount of commodity i, that we wish to ship between s_i and t_i. Given such a problem, we would like to know, whether there is a *feasible* flow, i.e., if it is possible to find a flow that satisfies the demands and obeys the capacity constraints. In general, we would like to find the maximum z, such that z percent of each demand can be shipped. We will call the value z the *throughput*.

To see the immense importance of multicommodity flow problems, note that a solution to the dual of the maximum fractional multi-commodity flow problem, which is a *minimum fractional multi-cut*, can be used to obtain an integral multi-cut [17]. Similarly, we can use an optimal solution to the maximum concurrent flow problem to approximate the *sparsest cut* problem [15]. This in turn is used in the design of divide and conquer algorithms [5]. Multicommodity flow algorithms are also used as a basic routine in the problem of finding edge disjoint paths in planar graphs [10].

In chapter 4, we explain the approximation algorithm for covering problems. As pointed out earlier, we shall employ ideas used in the greedy set-cover algorithm presented by Chvátal [3]. In the greedy procedure, we pick in each iteration a set S, that minimizes the ratio of the cost of the set to the total requirement of the elements in the set. Inherent in the procedure is the simultaneous construction of a primal and a dual solution. We finally manage to prove that the gap between the values of both is sufficiently small.

The intuitive idea behind the algorithm is, that the elements of a set S vote for the selection of S with their current requirement. In each iteration we choose the set that minimizes the ratio of the cost of the set to the total requirement of the elements of it. Thus, we minimize the ratio of the cost of picking set S to the benefit incurred. Every time, we cover an element, we decrease its requirement.

In chapter 3, we present the algorithm for packing linear programs. The algorithm can be best understood at the example of multicommodity flow. The complexity of the problem arises from the fact that the different commodities are all competing for the scarce edge capacity resource. The described algorithms use flow routing techniques to construct a primal solution to an instance of a packing problem. As in the covering case, we construct again dual and primal solutions at the same time and shall finally argue, that the values of both are close to each other.

Intuitively, newly added flow paths should use edges with low congestion, i.e. the percentage of utilization of the edges should be low. This is modeled by associating a length with each edge. The length of edge e is exponentially related to its congestion. In each iteration, we compute a shortest path with respect to the current active length function. By this, a shortest path will use edges of low congestion.

In chapter (5), we consider the mentioned concurrent flow setting. We employ a slight modification of the ideas that prove their usefulness in the packing chapter to construct an algorithm for this case. We obtain running times, that match the best known bounds, proved by Radzik [21], while guaranteeing a good solution granularity.

The idea of exponential edge lengths, that guide the routing process was first proposed by Shahrokhi and Matula [7]. It was later used by Plotkin, Shmoys and Tardos [19] and Young [18] to present algorithms for the same class of problems, that we address. Both algorithms fail to achieve a running time which is independent of the size of the input. In the packing case we present strongly polynomial procedures while in the general covering case, our algorithms turn out to have polynomial running time.

Besides that, note, that we treat covering and packing problems separately, though both are duals of each other. We will later show, that both procedures compute primal solutions of good granularity, while there is no integrality guarantee for the produced dual solutions. We achieve the same integrality as Young does.

Moreover, the known procedures for covering and packing algorithms assume to be able to do exponentiation in a single step. We do not need to use exponentiation. Instead we show that the kept length of an edge e is approximately exponentially related to the flow on e. That applies also for the covering algorithm. Furthermore, our algorithms use nice numbers, i.e. the numbers used in our problems are exact powers of a constant. This can perhaps be an advantage when implementing the algorithm on the RAM.

Finally, we outline practical problems, that fall into the described class of problems. The *cutting-stock* problem is an example of a covering problem: paper is produced in wide rolls, called *raws*, and then subdivided into several different widths of narrower ones, called *finals*. There is a special demand of rolls of each final width. The aim is, to minimize the amount of raws needed.

There are a lot of applications in the packing framework. Packing problems occur in the fields of telecommunication, freight transport and scheduling, network design, traffic planning and import-export.

Chapter 2

Basics and Preliminaries

In this chapter, we elaborate on basics which are needed to follow the rest of the thesis. Besides that, we also comment on previous work in the area of packing and covering.

2.1 Basic Graph Theory

We denote a *graph* G by the tuple (V, E), where V is the finite set of *vertices* and $E \subseteq V^2$ is the finite set of *edges* of G. In this thesis, we will always work on *directed* graphs, i.e. we assume, that the pairs from E are ordered, i.e. (u, v) and (v, u) denote different edges of the graph. We say, edge $e = (u, v)$ is *incident from* or *leaves* vertex u and is *incident to* or *enters* vertex v. We also say that vertex v is *adjacent* to vertex u iff $(u, v) \in E$. In the following, we define two sets of edges: Γ_u contains all edges e that leave vertex u and Π_u contains all edges e, that enter u, i.e.

$$\Gamma_u = \{e = (u, v) \in E\} \tag{2.1}$$

$$\Pi_u = \{e = (v, u) \in E\} \tag{2.2}$$

We call graph G a *multi-graph*, if E can contain several copies of e, i.e. if we allow parallel edges.

A *path* P in G from node s to node t (we shall also use the expression s-t path) is a sequence $\langle s = v_1, v_2, \ldots, v_p = t\rangle$ of vertices from V, where for all $1 \leq i < p$, v_{i+1} is adjacent to v_i in G. We will sometimes consider P to be the set $\{(v_i, v_{i+1}) \in E : 1 \leq i < p\}$ of edges on the path. Let now $l : E \to \mathbb{R}_+$ be a function which assigns positive lengths to edges of G. Consequently, we compute the length of path P by

$$\texttt{length}_l(P) = \sum_{e \in P} l(e) \tag{2.3}$$

Accordingly, we say, P is a shortest path between node s and node t with respect to the length function l iff there is no s-t path P', such that $\texttt{length}_l(P') < \texttt{length}_l(P)$. We shall later use a distance function $\texttt{dist}_l(u, v) : V^2 \to \mathbb{R}_+$ that is defined as follows

$$\texttt{dist}_l(u, v) = \min_{P \in P_{uv}} \texttt{length}_l(P) \tag{2.4}$$

where P_{uv} is the set of u-v paths.

A *cut* $C = (S, V - S)$ in G is a partition of the vertex set V into the disjoint sets S and $V - S$. We say $e = (u, v) \in E$ is a *cut-edge* iff either $u \in S$ and $v \in V - S$ or $v \in S$ and $u \in V - S$. A cut C is called *s-t cut* for vertices s and t iff each s-t path P contains at least one cut-edge from C.

Consider now a set of distinguished pairs of vertices of G, $T = \{(s_1, t_1), \ldots, (s_k, t_k)\}$. We call s_i and t_i the *terminals* of pair i. Extending the cut definition of the last paragraph, we call cut $C = (S, V - S)$ a *multi-cut* for T iff for all $1 \leq i \leq k$, each s_i-t_i path P contains at least one cut-edge from C.

In the following, we define a function $c : E \to \mathbb{R}_+$ that assigns positive *capacities* to the edges of G. A graph G together with its associated capacity function c is sometimes called a *flow network*. Consider two distinguished vertices s and t of V. An *s-t flow* in G is a function $f : E \leftarrow \mathbb{R}_+$ that satisfies the following properties:

1. *Capacity Constraints:* $\forall e \in V : f(e) \leq c(e)$
2. *Flow Conservation:* $\forall u \in V - \{s, t\} : \sum_{e \in \Pi_u} f(e) - \sum_{e \in \Gamma_u} f(e) = 0$

We name s the *source* and t the *sink* of the flow network G. The *value* of flow f is defined as

$$|f| = \sum_{e \in \Gamma(s)} f(e) \tag{2.5}$$

Having a flow f in G, we define the *congestion* of edge e as the ratio $f(e)/c(e)$, i.e. the congestion indicates the amount of used capacity.

Additionally, we define the *cost* of a flow f with respect to edge costs l as

$$\texttt{cost}_l(f) = \sum_{e \in E} f(e) l(e) \tag{2.6}$$

i.e. $l(e)$ defines the costs of a unit of flow on edge e.

Let again s and t be the source and the sink of a flow network $G = (V, E)$. Let l be the cost and c be the capacity function defined on E. Assume, that a *demand* of d units of flow should be shipped from s to t obeying the capacity constraints. In the following let F_{st} be the set of feasible s-t flows in G. We define

$$\texttt{mincost-flow}_l(s, t) = \min_{f \in F_{st}, |f| = d} \texttt{cost}_l(f) \tag{2.7}$$

as the value of the flow f, which minimizes the cost of shipping d units of flow.

2.2 Linear Programming

In this thesis, we develop combinatorial algorithms for *optimization problems*. An instance of an optimization problem with input q consists of a non empty set $Z(q)$ of *feasible solutions*.

We also have a function $c : Z(q) \to \mathbb{R}_+$ that maps feasible solutions to a positive real cost. Solving a linear optimization problem now means, to compute $x^* \in Z(q)$, such that c is maximized, i.e.

$$c(x^*) = \max_{x \in Z(q)} c(x) \tag{2.8}$$

Each of the problems in this thesis can be described using linear programming techniques. This section provides basic definitions and techniques.

Throughout the thesis, we deal with the following general form of a *linear program* (LP)

$$\begin{aligned} \mathbf{max} \quad & c^T x \\ \mathbf{s.t.} \quad & Ax \leq b \\ & x \in \mathbb{R}^n_+ \end{aligned} \tag{2.9}$$

where A is an $m \times n$ matrix, b is a vector of dimension m and c is a vector of dimension n. The entries of A, b and c are assumed to be rational numbers. We say, $x \in \mathbb{R}^n_+$ is a *feasible* solution to problem (2.9) iff $A^T x \leq b$ and a feasible x is *optimal* iff there is no feasible x' such that $c^T x < c^T x'$. We call a linear program *feasible* iff there is a feasible solution $x \in \mathbb{R}^n_+$.

Linear programming duality plays an important role in the development of our algorithms. We call problem (2.9) the *primal* problem and state its *dual* as

$$\begin{aligned} \mathbf{min} \quad & b^T y \\ \mathbf{s.t.} \quad & A^T y \geq c \\ & y \in \mathbb{R}^m_+ \end{aligned} \tag{2.10}$$

In the following we will say, a solution is *primal feasible* if it is feasible to the primal problem and we will use *dual feasible* to identify feasible solutions to the dual program. It is important to recognize, that we could have defined the latter problem as the primal and the first one as the dual since the following proposition applies

Proposition 2.2.1 *The dual of the dual program is the primal LP.*

In the following we summarize important results on linear programming duality, which turn out to be important in the analysis of the algorithms, we present in the remainder of this work. Assume, that z_p^* is the value of an optimal solution to the primal problem and z_d^* is the value of an optimum solution to the dual problem. Then, feasible solutions to (2.9) provide lower bounds on z_p^* while feasible solutions to (2.10) provide upper bounds on z_d^*.

Proposition 2.2.2 (Weak Duality) *If x is primal feasible and y is dual feasible, then $c^T x \leq z_p^* \leq z_d^* \leq y^T b$.*

We now come to the fundamental result of linear programming duality, which says that if both, the primal and its associated dual problem are feasible then the values of their optimal solutions are equivalent.

Proposition 2.2.3 (Strong Duality) *If (2.9) and (2.10) are feasible, then $z_p^* = z_d^*$.*

2.3 Approximation Algorithms

Consider an optimization problem P with cost function c and let $\mathrm{Opt}(q)$ be the optimum cost of a feasible solution for input q. An *approximation algorithm* for P is an algorithm A, that computes for each possible input q a feasible solution $A(q)$. We always have

$$c(A(q)) \leq \mathrm{Opt}(q) \tag{2.11}$$

We are interested in the quality of the solution $A(q)$, i.e. how close is it to the optimum solution $\mathrm{Opt}(q)$.

We define the quality of the solution $A(q)$ in an absolute and a relative way. The approximation algorithm A is said to have an *absolute performance guarantee* of α, if for all possible inputs q, we have

$$c(A(q)) \geq \mathrm{Opt}(q) - \alpha \tag{2.12}$$

Similarly, we say that A has a *relative performance guarantee* of α, if for all possible inputs, we have

$$\alpha c(A(q)) \geq \mathrm{Opt}(q) \tag{2.13}$$

Absolute performance guarantees tend to be very rare. In fact, in this thesis we only present approximation algorithms that provide relative performance guarantees. Furthermore, we will say, an algorithm A is an *α-approximation* if it has a relative performance guarantee of α. Likewise, we call a solution $A(q)$ *α-approximate*, if (2.13) holds.

We shall now present a class of approximation algorithms A_ϵ that provide a $(1+\epsilon)$-approximation for problem P with arbitrarily small ϵ. We shall call algorithm A_ϵ an *approximation scheme*. If the running time of A_ϵ is bounded by a polynomial in the length of the input of q, we call A_ϵ a *polynomial time approximation scheme* (PTAS). If the running time is bounded by a polynomial in the length of the input and in $1/\epsilon$ we call A_ϵ a *fully polynomial time approximation scheme* (FPTAS). It is important, to note that the definition of the class PTAS does not specify the dependence of the running time of a_ϵ on $1/\epsilon$. It can be exponential.

2.4 Basic Recurrences

In the following, we would like to develop a closed formula for a recurrence that we will need several times throughout the thesis. The basis f_0 of the recurrence is given. Any further term is defined by

$$f_n = \nu_1 \sum_{j=0}^{n-1} f_j + \nu_2 \tag{2.14}$$

We use the definitions of f_n and f_{n-1} to introduce the closed formula for f_n.

$$\begin{aligned} f_n &= \nu_1 f_{n-1} + \nu_1 \sum_{j=0}^{n-2} f_j + \nu_2 && (2.15) \\ &= (1+\nu_1) f_{n-1} && (2.16) \\ &= (1+\nu_1)^{n-1} f_1 && (2.17) \end{aligned}$$

We know, that $f_1 = \nu_1 f_0 + \nu 2$ and hence obtain the desired closed formula for f_n.

Lemma 2.4.1 $f_n = (f_0\nu_1 + \nu_2)(1+\nu_1)^{n-1}$ *for* $n \geq 1$

In the following, we treat the slightly different recurrence

$$f'_n = \nu_1 \sum_{j=1}^{n-1} f'_j + \nu_2 \tag{2.18}$$

for all $n > 1$ and f'_1 is the recurrence basis. We obtain the closed formula by reducing f'_n to f_n. Let therefore be $f'_n = g_{n-1}$ and rewrite (2.18) using g_n.

$$f'_{n+1} = g_n = \nu_1 \sum_{j=0}^{n-1} g_j + \nu_2 \tag{2.19}$$

We now apply lemma 2.4.1 to g_n and get

Corollary 2.4.1 $f'_n = (f'_1\nu_1 + \nu_2)(1+\nu_1)^{n-2}$ *for all* $n \geq 2$

Now let $u(n)$ be a function, such that $u(0) = 0$ and $u(i+1) > u(i)$ for all i. At some later point we need the following recurrence

$$f''_n = \nu_1 \sum_{j=0}^{n-1} (u(j+1) - u(j)) f_j + \nu_2 \tag{2.20}$$

for all $n > 0$ and with basis f''_0.

Corollary 2.4.2 $f''_n = (f''_0\nu_1 + \nu_2)(1+\nu_1)^{u(n)-1}$ *for all* $n \geq 1$

Proof: We again reduce this recurrence to the form of f_n. Let therefore be

$$\forall n \in \{u(j), \ldots, u(j+1)-1\} : g_n = f''_j \tag{2.21}$$

Since the set $\{u(j), \ldots, u(j+1)-1\}$ contains exactly $u(j+1) - u(j)$ elements, we can rewrite (2.20) as

$$f''_n = g_{u(n)} = \nu_1 \sum_{j=0}^{u(n)-1} g_j + \nu_2 \tag{2.22}$$

Applying lemma (2.4.1) immediately gives us

$$f''_n = (f''_0\nu_1 + \nu_2)(1+\nu_1)^{u(n)-1} \tag{2.23}$$

■

In the later chapters, it will often be the case, that $\nu_2 \leq f_0$. In this case, we get a simpler upper bound on f_n.

Corollary 2.4.3 *If $f_0 \leq \nu_2$, we have $f_n \leq \nu_2 e^{\nu_1 n}$*

Proof: This follows by the inequality

$$f_n = (f_0\nu_1 + \nu_2)(1+\nu_1)^{n-1} \leq \nu_2(1+\nu_1)^n \leq \nu_2 e^{\nu_1 n} \tag{2.24}$$

The last step used $1 + x \leq e^x$. ■

The same results apply also for the other two recurrences.

Corollary 2.4.4 *If $f_0' \leq \nu_2$, we have $f_n' \leq \nu_2 e^{\nu_1 (n-1)}$*

Corollary 2.4.5 *If $f_0'' \leq \nu_2$, we have $f_n'' \leq \nu_2 e^{\nu_1 u(n)}$*

2.5 Previous Work

In this thesis, we present combinatorial approximation algorithms for general fractional packing and covering problems. Note, that all of the presented algorithms give approximate solutions to problems, which are exactly solvable using linear programming approaches.

The theoretically fastest known methods to solve linear programs are due to Vaidya [22]. The algorithm obtains optimal solutions to the presented problems using $O(mL)$ calls to an optimization subroutine over a convex polytope P plus $O(mM(m)L)$ additional time. Here, L and $M(m)$ denote the binary size of the problem and the time needed to invert an $m \times m$ matrix, respectively. In each of the problems, we will present a certain optimization subroutine, which in Vaidya's algorithm is used to solve the separation problem appearing in the ellipsoid method for linear programming.

In the past, people developed several combinatorial algorithms to avoid the bottleneck of linear programming. The first publications mainly concentrated on multicommodity flow problems. Shahrokhi and Matula [7] presented an approximation algorithm for the maximum concurrent flow problem with unit capacities which runs in $O(\epsilon^{-5}nm^7)$. They use flow rerouting techniques, i.e. they start with a flow which satisfies the demand constraints of all commodities but is infeasible in terms of edge capacities. Throughout the execution of their algorithm the congestion of the edges of the graph is lowered. Besides that, the authors were the first, who assigned a length to each edge which depends exponentially on the flow on this edge. These lengths guide the flow rerouting steps, i.e. high weights indicate heavily used edges which should not be used.

Subsequently, the running time of the latter work was extended by Klein et al [12]. Leighton et al [20] were the first, who considered the arbitrary capacity case of the maximum concurrent flow problem. In the latter paper the authors introduce the idea of doing the flow rerouting by computing single commodity minimum cost flows with respect to edge weights, which again depend exponentially on the flow on these edges. They also used Lagrangian relaxation techniques, i.e. if f_0 is the current flow, they compute a new flow f_0^* by the mentioned min-cost flow computation and define the flow for the next iteration by $f_1 = (1-\sigma)f_0 + \sigma f_0^*$ for some small constant σ. The deterministic algorithm developed in [12] solves the maximum concurrent flow problem, by computing $O(\epsilon^{-2}k^2 \log k \log n)$ single commodity minimum cost flows.

The authors also present a randomized version which uses $O(\epsilon^{-3}k \log k \log n)$ min-cost flows. The latter algorithm was simplified by Goldberg [2]. Goldberg also improved the running time by replacing ϵ^{-3} by ϵ^{-2}. Finally, Radzik [21] extended the line of work in the area of concurrent flow by giving a rerouting based approximation algorithm which uses $O(k(\epsilon^{-2} + \log k) \log n)$ min-cost flow computations. He basically showed how to replace the random selection of commodities from [2] by a deterministic round-robin version without increasing the running time. We match this running time as we show in chapter 5.

Our work points out general techniques comparable to the work of Plotkin, Shmoys and Tardos [19] and Young [18]. They both presented methods for problems which fit into the general framework of covering and packing problems. Both papers have the Lagrangian Relaxation technique in common.

Young addresses the general problems by introducing the method of oblivious rounding. At first, he assumes to know an optimal solution x^* to one of the general problems leading to a distribution over the solution space P, a convex polytope, with mean x^*. By sampling from P, he obtains a multi-set S and a point from P by computing the average $\bar{x}$ of the points from S.

Young bounds the probability of failure, i.e. the probability that $\bar{x}$ is no feasible solution to the addressed problem using Chernoff and Hoeffding bounds. These proofs involve an upper bound on the probability of failure in terms of an expected value. Introducing conditional expectation and showing that this expected value is proportional to a polynomial, which is independent of the optimal solution, is the key point in Young's paper. While the proofs for the algorithms in [18] use probability arguments, the methods itself proceed deterministically. In fact, the resulting methods are akin to the algorithms of previous papers but they were achieved using a completely different motivation.

Plotkin, Shmoys and Tardos assume that the mentioned convex polytope P embodies *easy* constraints, i.e. they use a fast optimization subroutine, which computes $z^* = \min\{cz : z \in P\}$, where $c = y^T A$ (in fact, Young also needs this optimization routine in his algorithms). In the packing case the main idea of their work is to maintain a solution $x \in P$ which is feasible with respect to edge capacities $\lambda c(e)$. Similarly, in the covering setting they maintain a solution $x \in P$, which covers the right hand side to an extent of $\lambda b(i)$.

In each iteration, the current solution is improved to obtain a feasible solution with better λ value, i.e. for the fractional packing problem, we try to reroute flow such that edges with low congestion are preferred. The rerouting is again guided by exponential length functions, i.e. in the multicommodity flow case the edge weights depend exponentially on the congestion of the edges. The adjustment of the current solution is again done in a Lagrangian manner.

Plotkin, Shmoys and Tardos present deterministic and randomized relaxed decision procedures for covering and packing problems. The authors show that the running time of their procedures depends on the *width* ρ of the given problem. The definition of ρ is given with respect to the solution space P

$$\rho = \max_i \max_{x \in P} \frac{a_i x}{b_i} \tag{2.25}$$

Observe, that ρ might not be sufficiently small, it might even be exponential in the input size. However, the authors show, that in some cases the width of a problem can be decreased.

In the latter paper, it is proven, that the number of calls of the optimization subroutine in the deterministic relaxed decision procedure for fractional packing is bounded by $O(\epsilon^{-2}\rho\log(m\epsilon^{-1}))$. Similarly, the number of calls to the optimization subroutine in the covering case is bounded by $O(m + \rho\log^2 m + \epsilon^{-2}\rho\log(m\epsilon^{-1}))$. The randomized version of the last two procedures considers the special case, when the polytope P is in product form, i.e. $P = P_1 \times \cdots \times P_k$. Let ρ_l be the width of polytope P_l. The running time of the randomized relaxed decision procedure for fractional packing is bounded by $O(\epsilon^{-2}(\sum_l \rho_l)\log(m\epsilon^{-1}) + k\log(\rho\epsilon^{-1}))$ calls to the optimization subroutine. The randomized covering procedure uses $O(mk + (\sum_l \rho_l)\log^2 m + k\log\epsilon^{-1} + \epsilon^{-2}(\sum_l \rho_l)\log(m\epsilon^{-1}))$ calls to the mentioned subroutine. Thus, randomization speeds up the algorithms roughly by a factor of k, if $\rho = \sum_l \rho_l$.

We shall present methods for the described general problems which have strongly polynomial running times which in particular do not depend on the problem's width. In addition to that, in the last paper the authors assume to be able to do exponentiation in a single step. We will also use the exponential edge label idea but use implicit exponentiation, i.e. we multiply labels continuously. Furthermore, we guarantee a strongly polynomial integrality of the solution.

Besides the mentioned work, Luby and Nisan [14] presented a parallel approximation algorithm for the case of positive linear programming, i.e. the entries of matrix A and vectors b and c are constrained to be positive numbers. The presented algorithm also keeps dual and primal solutions during its progress. The change in the primal solution is guided by the current dual one. For this purpose, the authors again use the idea of exponential weights on the rows of the primal program. The algorithm of Luby and Nisan proceeds in an iterative manner, i.e. it is no rerouting approach like most of the previously mentioned algorithms.

Chapter 3

Packing

In this chapter, we consider problems, which can be described by the following type of linear program.

$$\begin{aligned} \textbf{max} \quad & c^T x \\ \textbf{s.t.} \quad & Ax \leq b \end{aligned} \tag{3.1}$$

Intuitively, a feasible solution to this problem is a packing of the columns of A into b. For this reason, we shall call this LP a *packing* LP. The chapter starts with the description of an important subclass of packing problems: *multicommodity flow problems*. By studying these problems, we will develop the important ideas which will later lead to algorithms for the general packing setting.

3.1 Maximum Multicommodity Flow in Unit Capacity Networks

In the *maximum fractional multicommodity flow* (MFMF) problem we consider a graph $G = (V, E)$, in which each edge $e \in E$ has an associated nonnegative capacity $c(e)$. We are given k pairs of terminals $(s_1, t_1), \ldots, (s_k, t_k)$ each of which has a commodity associated with it. s_i and t_i are the source and the sink, respectively, for commodity i. We now route flow for each of the commodities, i.e. we ship a certain amount of commodity i from s_i to t_i for all i. The objective of the maximum multicommodity flow problem is to maximize the sum of the flow values while obeying the capacity constraints of the edges in the network.

Let $\mathcal{P}_i$ be the set of all possible paths between s_i and t_i, i.e. the paths between the ith source and sink. Consequently, we define $\mathcal{P}$ to be the union of all $\mathcal{P}_i$. Furthermore, let $\mathcal{P}_i(e)$ denote the paths from $\mathcal{P}_i$ that use edge e. In the linear programming formulation we associate a variable x_P with each path P.

$$\begin{aligned}\textbf{max} \quad & \sum_{P \in \mathcal{P}} x_P \\ \textbf{s.t.} \quad \forall e \in E: \quad & \sum_{P \in \mathcal{P}(e)} x_P \leq c(e) \\ \forall P \in \mathcal{P}: \quad & x_P \geq 0\end{aligned} \tag{3.2}$$

This is a packing problem and the dual covering problem is

$$\begin{aligned}\textbf{min} \quad & \sum_{e \in E} c(e) l(e) \\ \textbf{s.t.} \quad \forall P \in \mathcal{P}: \quad & \sum_{e \in P} l(e) \geq 1 \\ \forall e \in E: \quad & l(e) \geq 0\end{aligned} \tag{3.3}$$

Intuitively, we think of the variable $l(e)$ as the length of edge e. Each path between two terminals should have length more than one. In case of a solution with $l(e) \in \{0, 1\}$ for all e, we achieve a *multi-cut*, i.e. a collection of edges whose removal disconnects the terminal pairs. Since we do not insist on integral solutions the optimum vector l will constitute a *fractional multi-cut*, i.e. each pair of terminals is separated by a path of length at least 1 with respect to the edge lengths $l(e)$.

In the following, think of the graph G as a system of pipes. Each edge corresponds to a pipe. Consider $c(e)$ to be the cross-section area of the pipe associated with edge e. Thus, the objective function of (3.3) can be interpreted as minimizing the volume of the whole pipe system. Hence, the maximum fractional multicommodity flow is equal to the minimum volume pipe system such that the terminals of each pair are separated by at least a unit distance.

3.1.1 The Algorithm

The following results compare with previous work of Plotkin, Shmoys and Tardos [19] and Young [18]. The main ideas for the algorithm arise from the intuitive interpretation of the primal and dual programs. During the procedure we maintain primal and dual solutions. In each iteration, we compare the current primal feasible solution value to the best dual one. The termination condition is chosen such that the multiplicative gap between the two is small enough. We will prove a bound on the steps needed to achieve the desired approximation guarantee. In this section we assume unit edge capacities . This assumption will be relaxed at a later point.

With edge $e \in E$ we associate a length which is updated in each iteration. Let $l_0(e)$ be the initial length of e and $l_i(e)$ be its length after the ith iteration. We will choose $l_0(e)$ to be δ for all edges $e \in E$, where δ is a small constant, that we specify later. Let $\mathtt{dist}_l$ be the distance function induced by the length function l, i.e. $\mathtt{dist}_l(s_j, t_j)$ is the length of the shortest path between s_j and t_j with respect to length function l.

$$\mathtt{dist}_l(s_j, t_j) = \min_{P \in \mathcal{P}_j} \sum_{e \in P} l(e) \tag{3.4}$$

Initialize:	$\forall e \in E : l_0(e) = \delta$
	$D_0 = m\delta$
Iteration i:	$\alpha_i = \min_j\{\mathtt{dist}_{l_i}(s_j, t_j)\}$
	if $\alpha_i < 1$ **then**
	$x_{i+1,P_i} = x_{i,P_i} + 1$
	$D_{i+1} = D_i + \epsilon \sum_{e \in P_i} l_i(e) = D_i + \epsilon\alpha_i$
	$\forall e \in E : l_{i+1}(e) = \begin{cases} (1+\epsilon)l_i(e) : e \in P_i \\ l_i(e) : else \end{cases}$
	else
	stop
	fi
Solution:	$\left(\frac{t}{\log_{1+\epsilon}(1+\epsilon)/\delta}, \frac{x_t}{\log_{1+\epsilon}(1+\epsilon)/\delta}\right)$

Figure 3.1: The basic algorithm routes one unit of flow in each iteration.

In each iteration we compute the shortest path $P \in \mathcal{P}$ with respect to l_i. We use $\alpha(l)$ to denote the length of the shortest path under length function l, i.e.

$$\alpha(l) = \min_{P \in \mathcal{P}} \mathtt{length}_l(P) \tag{3.5}$$

For brevity, we use α_i instead of $\alpha(l_i)$. The algorithm terminates, whenever $\alpha_i \geq 1$. We assume, that this happens in iteration t.

Now, if in iteration i the value α_i is less than 1 we route a unit of flow along the corresponding shortest path, which we will denote by P_i, i.e. we increase x_{P_i} by 1. We will denote the value of x_{P_i} at the beginning of iteration i by x_{i,P_i},i.e.

$$x_{i+1,P_i} = X_{i,P_i} + 1 \tag{3.6}$$

After having routed flow along path P_i, we update the lengths of edges used. We increase the length of each edge e on P_i by a factor of $(1 + \epsilon)$. By this, every time we send flow through an edge, we increase its length; the longer the edge is the higher the flow through it. The shortest path computation with respect to the current edge lengths will select a path that uses edges of small congestion, i.e. the flow on these edges is not too high.

Let $M = \max_{e \in E} \sum_{P \in \mathcal{P}(e)} x_P$ be the maximum number of paths using an edge. A solution f is primal feasible, if the flow on each edge does not exceed the capacity of the edge. Thus, if we divide each variable x_P by the maximum congestion M, we obtain a feasible flow. A set of edge labels l constitutes a dual feasible solution if the length of every path $P \in \mathcal{P}$ with respect to l is at least 1. Thus, a dual feasible solution can be obtained by dividing each edge label by the current shortest path length α_i. We define

$$D(l) = \sum_{e \in E} l(e) \tag{3.7}$$

and introduce the abbreviation D_i, which denotes $D(l_i)$. In the following, let β be the value of the optimal dual solution.

For the analysis of this algorithm it will be useful to remove the initial value $l_0(e)$ from the edge labels. We will use the abbreviations α_i' and D_i' to denote $\alpha(l_i - l_0)$ and $D(l_i - l_0)$, respectively.Thus, it follows using the same argument as before that D_i'/α_i' is the value of a dual feasible solution. In the following we denote the latter ratio by β_i. Observe, that during the first iterations α_i' might be of value 0. In this case we define β_i as infinity.

3.1.2 Analysis

In this section, we are going to prove bounds on the approximation guarantee of the solutions we compute using algorithm (3.1). At first, we derive a relationship between the shortest path lengths α_i and the sum $\sum_{e \in E} l_i(e)$, i.e. D_i. Observe, that $D_{i+1} = D_i + \epsilon\alpha_i$ and therefore

$$D_i = \epsilon \sum_{j=0}^{i-1} \alpha_j + D_0 \tag{3.8}$$

which implies

$$D_i' = \epsilon \sum_{j=0}^{i-1} \alpha_j \tag{3.9}$$

By the definition of β, we know that $\beta \leq \beta_i$ for all i and hence, $D_i' \geq \beta\alpha_i'$ for all i. We use the last inequality and (3.9) to get

$$\beta\alpha_i' \leq \epsilon \sum_{j=0}^{i-1} \alpha_j \tag{3.10}$$

We know that α_i' is at least $\alpha_i - L\delta$, where L is the maximum number of edges on any path on which flow is routed and δ is the initial length of each edge. Combining the last facts, we derive a recurrence for α_i

$$\alpha_i \leq \frac{\epsilon}{\beta} \sum_{j=0}^{i-1} \alpha_j + L\delta \tag{3.11}$$

Lemma 3.1.1 *$\alpha_i \leq L\delta e^{(\epsilon i)/\beta}$ for all i*

Proof: Since $\alpha_0 \leq L\delta$, the lemma follows by an application of corollary (2.4.3) with $f_n = \alpha_n$,$\nu_1 = \epsilon/\beta$ and $\nu_2 = L\delta$. We obtain

$$\alpha_i \leq L\delta e^{\epsilon i/\beta} \tag{3.12}$$

■

The algorithm terminates after t iterations iff the length α_t of the shortest path satisfies $\alpha_t \geq 1$. Combining lemma (3.1.1) and the last inequality we get

$$L\delta e^{(\epsilon t)/\beta} \geq 1 \tag{3.13}$$

and hence

$$\frac{\beta}{t} \leq \frac{\epsilon}{\ln 1/(L\delta)} \tag{3.14}$$

Note, that t is the value of the flow, we routed. It is not necessarily feasible, i.e. we probably used each of the edges several times. Thus, let M be the maximum congestion. It follows that

$$\frac{\beta M}{t} \leq \frac{\epsilon M}{\ln 1/(L\delta)} \tag{3.15}$$

The left hand side of the last inequality is the ratio of the values of the dual and the primal solutions. We therefore need to bound the right hand side.

Note, that the last time, we increased the label of edge e, it was on a routing path of length less than 1 and hence $l(e) < 1$. At this point we increased $l(e)$ by a factor of $(1+\epsilon)$. Thus, its final length is at most $(1+\epsilon)$. For this reason, the amount of flow through edge e is bounded by M, where

$$\delta(1+\epsilon)^M \leq (1+\epsilon) \tag{3.16}$$

and hence $M \leq \log_{1+\epsilon} \frac{1+\epsilon}{\delta}$. By this, the right hand side of inequality (3.15) is bounded by

$$\frac{\epsilon \ln(1+\epsilon)/\delta}{\ln(1+\epsilon) \ln 1/(L\delta)} \tag{3.17}$$

We choose $\delta = (1+\epsilon)^{1-1/\epsilon} L^{-1/\epsilon}$. The last expression works out to

$$\frac{\epsilon \ln((1+\epsilon)L)^{1/\epsilon}}{\ln(1+\epsilon) \ln((1+\epsilon)L)^{1/\epsilon - 1}} = \frac{1}{\ln(1+\epsilon)(1/\epsilon - 1)} \tag{3.18}$$

which in turn is bounded by

$$\frac{1}{(\epsilon - \epsilon^2/2)(1/\epsilon - 1)} \tag{3.19}$$

since $\ln(1+\epsilon) \geq \epsilon - \epsilon^2/2$ by the Taylor series expansion of $\ln(1+\epsilon)$. To simplify the last expression, we observe, that the denominator of the last term is at least $(1-\epsilon)^2$ and thus, we achieve the following upper bound on the approximation ratio

$$\frac{1}{(1-\epsilon)^2} \tag{3.20}$$

It remains to choose ϵ such that the last term is at most $(1+\omega)$.

Theorem 3.1.1 *Algorithm (3.1) is a $(1+\omega)$ approximation for the MFMF problem by choosing $\delta = (1+\epsilon)^{1-1/\epsilon} L^{-1/\epsilon}$ and $\epsilon \leq 1 - \sqrt{\frac{1}{1+\omega}}$*

3.1.3 Modified Algorithm

Lemma (3.1.1) shows that algorithm (3.1) is robust in the sense that the computation of the exact shortest path in each iteration is not necessary. For the sake of the analysis it suffices to have in iteration i a path of length less than $L\delta e^{\epsilon i/\beta'}$, where β' in this case denotes the minimum β_j so far (which is in fact only larger than β and therefore, the last bound is only sharper). Besides that, why should we only route one unit of flow in each iteration? Routing as much flow in an iteration as long as we stay below the bound of lemma (3.1.1) will have no influence on the correctness of the previous analysis.

Initialize:	$\forall e \in E : l_0(e) = \delta$
	$f = 0$
	$D_0 = m\delta$
Phase i:	$\alpha_f = \min_{1 \le j \le k} dist_{l_f}(s_j, t_j)$
	$\beta_i = D'_f / \alpha'_f$
	$\beta' = \min_{0 \le j \le i} \beta_j$
	if $\alpha_f \ge 1$ **then stop**
	forall $1 \le j \le k$ **do**
	$P_{i,j} = \texttt{shortest-path}_{l_f}(s_j, t_j)$
	while $\texttt{length}_{l_f}(P_{ij}) < 1$ **and** $\texttt{length}_{l_f}(P_{ij}) \le L\delta e^{\epsilon f/\beta'}$ **do**
	$\alpha_f = \texttt{length}_{l_f}(P_{ij})$
	$x_{i+1,P_{ij}} = x_{i,P_{ij}} + 1$
	$D_{f+1} = D_f + \epsilon\alpha_f$
	$\forall e \in E : l_{f+1}(e) = \begin{cases} (1+\epsilon) l_f(e) : e \in P_{ij} \\ l_f(e) \quad\quad : else \end{cases}$
	$f = f + 1$
	od
	od
Solution:	$\left(\frac{f_t}{\log_{1+\epsilon}(1+\epsilon)/\delta}, \frac{x_t}{\log_{1+\epsilon}(1+\epsilon)/\delta} \right)$

Figure 3.2: The modified algorithm uses less shortest path computations and routes more then one unit of flow in each phase.

Algorithm (3.2) proceeds in phases. Each phase i consists of k steps, one for each commodity j. Assume, that at the beginning of step j, we have already routed f units of flow in previous phases and steps. We define $l_f(e)$ as the edge label of edge e at this point. We compute the shortest path $P_{ij} \in \mathcal{P}_j$ with respect to edge lengths l_f. If P_{ij} is a valid routing path, we denote its length by $\alpha(l_f)$ or equivalently α_f. Similarly, $\alpha(l_f - l_0) = \alpha'_f$ is the length of path P_{ij} under the length function $l_f - l_0$. Besides that,

$$D(l_f) = D_f = \sum_{e \in E} l_f(e) \tag{3.21}$$

The definition of $D(l_f - l_0)$ or D'_f follows similarly. We again associate a primal LP variable x_P with each path P and denote its value at the beginning of iteration i by $x_{i,P}$.

Having computed path P_{ij}, we check whether its length is at most $L\delta e^{(\epsilon f)/\beta'}$, where β' is the best dual solution so far, i.e.

$$\beta' = \min_{o \leq j \leq i} \tag{3.22}$$

If the length of $P_{ij}$is sufficiently small, we route one unit of flow along the path. Afterwards, we update the lengths of the used edges as in algorithm (3.1),i.e. we compute length function l_{f+1}. Instead of doing an additional shortest path computation for the same commodity, we just compute the length of P_{ij} with respect to l_{f+1}. If this length is at most $L\delta e^{(\epsilon(f+1))/\beta'}$, we let α_{f+1} be the current length of P_{ij} and continue routing. We proceed in this fashion until we exceed the mentioned bound. At this point we move to step $j+1$.

Observe, that in algorithm (3.2) we compute the length of the absolute shortest path only at the beginning of a phase. All shortest path computations in the steps are bound to one commodity. The computation of the shortest path $P \in \mathcal{P}$ involves k shortest path computations which formerly took place in each iteration, i.e. for each unit of flow routed. A phase in the new algorithm involves $2k$ shortest path computations overall and we expect to route more flow than in 2 iterations of the previous algorithm.

Furthermore, we compute upper bounds on the value of feasible dual solutions only at the beginning of a phase. We define

$$\beta_i = \frac{D(l_{f_i} - l_0)}{\alpha(l_{f_i} - l_0)} = \frac{D'_{f_i}}{\alpha'_{f_i}} \tag{3.23}$$

where f_i is the flow routed in phases 0 to $i-1$. To see, that theorem (3.1.1) also holds for the modified algorithm, observe, that at termination time we have $\alpha_f \geq 1$. We now ensure that $\alpha_f \leq L\delta e^{\epsilon f/\beta}$ for all f.

Lemma 3.1.2 *If algorithm (3.2) terminates with final flow f_t we have $\alpha_f \leq L\delta e^{\epsilon f/\beta}$ for $0 \leq f \leq f_t$*

Proof: At first note, that for $f = 0$ the assumption holds trivially since all paths are of length less than $L\delta$ by definition. Assume towards a contradiction that in any iteration $0 < f \leq f_t$ no flow path P of length less than $L\delta e^{\epsilon f/\beta}$ could be found. If this happens in the midst of a phase a new phase is started. The absolute shortest path length α'_f and its corresponding value α_f are computed. Since, at the beginning of a phase we have $\alpha'_f \geq \alpha_f - L\delta$ and $D'_f = \epsilon \sum_{j=0}^{f-1} \alpha_j$ lemma (3.1.1) can be applied and $\alpha_f \leq L\delta e^{\epsilon f/\beta}$ holds. ■

With this observation, the main theorem of the last section carries over without changing anything.

3.2 The Arbitrary Capacity Case

In this section we allow arbitrary integral capacities on the edges removing the unit capacity assumption of the last section. Note, that this setting can be viewed like having $c(e)$ parallel edges of unit capacity replacing the one edge e of capacity $c(e)$. In this case, we are back in

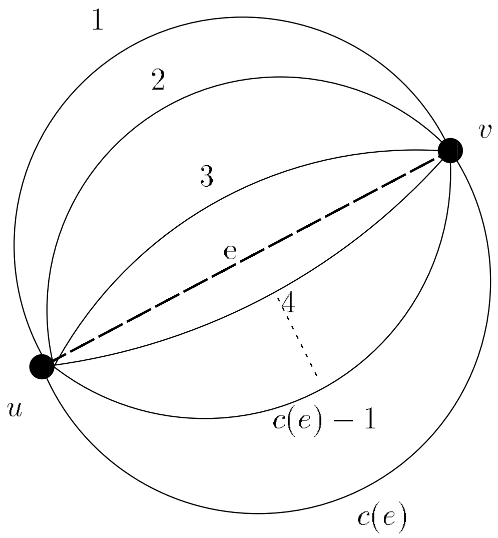

Figure 3.3: Considering an edge e of capacity $c(e)$ can be viewed as replacing e by $c(e)$ copies of unit capacity.

the last setting and can immediately apply the last algorithms. Note, however that replacing each edge by $c(e)$ copies will enlarge the row size of the system matrix. The size of the system will be dependent on the input data. It will turn out, that in this case, we won't get a running time which is polynomial in the input size. Since that, we develop a modification of the presented methods to treat this case directly instead of having to transform the graph at first.

3.2.1 Edge Lengths l^1 and l^2

Suppose, that we replace each edge e by $c(e)$ unit capacity edges in our network G. We again associate a length with each copy of edge e. We now proceed as in algorithm (3.1). In each iteration, we compute the shortest path P_i between the given terminal pairs, with respect to the current edge lengths $l_i(e)$. We route a unit of flow along this path and update the edge labels. Assume, that each edge e on the current routing path has a parallel copy whose length is also $l_i(e)$. Consequently, in the next iteration, the shortest path P_{i+1} would have the same length as P_i. In particular, the edges of P_{i+1} are all copies of edges on P_i. Hence, the path P_{i+1} can viewed as being identical to P_i. The algorithm sees the $c(e)$ copies of edge e as one edge. Its label does not change until all copies have been used.

We introduce the notion of a *reservoir* which captures the last argument. We associate $\texttt{res}_i(e)$ with each edge e. Initially, set $\texttt{res}_0(e) = 0$ and add the flow we send along the current routing path P to the reservoirs of all edges on P. Whenever $\texttt{res}(e) \bmod c(e) = 0$, i.e. all parallel copies of edge are used by the same amount of flow, we increase the length associated with edge e by a factor of $(1 + \epsilon)$. Denote the length of edge e at the beginning of iteration i by $l_i^1(e)$.

Consider again the multi-graph, that results from replacing each edge e by parallel copies. Take again the situation of iteration i in which we compute the shortest path P_i and route one unit of flow along it. Then in iteration $i+1$ it happens that we compute the shortest path P_{i+1} of the same length and that contains the same edges than P_i. Recall the definition of the dual objective function $D = \sum_e l(e)$ in the unit capacity case. Its value changes in iteration i, though the routing path does not change. For each used edge copy the value $\epsilon l_i^1(e)$ is added. Thus, the current dual objective function value can be described as follows

$$
\begin{aligned}
D_i &= \sum_{e\in E} c(e) l_i^1(e) + \sum_{e\in E} \epsilon \frac{\mathtt{res}_i(e) \bmod c(e)}{c(e)} c(e) l_i^1(e) &(3.24)\\
&= \sum_{e\in E} (1 + \epsilon \frac{\mathtt{res}_i(e) \bmod c(e)}{c(e)}) c(e) l_i^1(e) &(3.25)
\end{aligned}
$$

We define a second length function $l_i^2(e)$, that changes, whenever we use a copy of edge e, i.e.

$$
l_i^2(e) = (1 + \epsilon \frac{\mathtt{res}_i(e) \bmod c(e)}{c(e)}) l_i^1(e) \tag{3.26}
$$

and thus, we have that $D_i = D(l_i^2) = \sum_{e\in E} c(e) l_i^2(e)$.

Initialize:	$\forall e \in E : l_0^1(e) = \delta$
	$\forall e \in E : \mathtt{res}_0(e) = 0$
	$f_0 = 0$
	$D_0 = m\delta$
Iteration i:	$\alpha_i = \min_j \{\mathtt{dist}_{l_i^1}(s_j, t_j)\}$
	if $\alpha_i < 1$ **then**
	$\quad \rho_i = \min_{e\in P_i}(c(e) - (\mathtt{res}_i(e) - c(e)))$
	$\quad f_{i+1} = f_i + \rho_i$
	$\quad x_{i+1,P_i} = x_{i,P_i} + \rho_i$
	$\quad D_{i+1} = D_i + \epsilon\rho_i\alpha_i$
	$\quad \forall e \in P_i : \mathtt{res}_{i+1}(e) = \mathtt{res}_i(e) + \rho_i$
	$\quad \forall e \in E : l_{i+1}^1(e) = \begin{cases} (1+\epsilon) l_i^1(e) : \mathtt{res}_{i+1} \bmod c(e) = 0 \\ l_i^1(e) : else \end{cases}$
	else
	$\quad$ **stop**
	fi
Solution:	$\left(\frac{f_t}{\log_{1+\epsilon}(1+\epsilon)/\delta}, \frac{x_t}{\log_{1+\epsilon}(1+\epsilon)/\delta}\right)$

Figure 3.4: The algorithm for the maximum multi-commodity flow problem considering arbitrary capacities.

Summarizing the previous elaboration, we now have two different length functions l^1 and l^2. In iteration i, we compute the shortest path P_i with respect to length function l_i^1. We denote its length by $\alpha(l_i^1)$ or α_i. We shall again use α_i' and D_i' to denote $\alpha(l_i^1 - l_0^1)$ and $D(l_i^2 - l_0^2)$ respectively. We define $\beta_i = D_i'/\alpha_i'$ as in the last section and use β to denote the value of an optimal dual solution.

We now show, that β_i is at least of value β, i.e. it is only larger than the value of an optimal dual solution.

Lemma 3.2.1 *For all iterations i, $\beta_i \geq \beta$*

Proof: Observe, that

$$\frac{D_i'}{\alpha_i'} = \frac{D(l_i^2 - l_0^2)}{\alpha(l_i^1 - l_0^1)} \geq \frac{D(l_i^1 - l_0^1)}{\alpha(l_i^1 - l_0^1)} \tag{3.27}$$

This follows from the definition of l_i^2. The right hand side of the last inequality is a valid upper bound on the dual solution value and hence is the left hand side. ■

Instead of routing 1 unit of flow along path P_i, we route ρ_i units of flow, where

$$\rho_i = \min_{e \in P_i}(c(e) - (\texttt{res}_i(e) \bmod c(e))) \tag{3.28}$$

In other words, the amount is chosen such that for at least one edge $e \in P_i$, all copies are used,i.e. after having routed ρ_i units of flow along P_i, we have $\texttt{res}_{i+1}(e) \bmod c(e) = 0$ for at least one edge $e \in P_i$. In the following, let f_i be the total flow at the beginning of iteration i, i.e. $\rho_i = f_{i+1} - f_i$.

Lemma 3.2.2 *If we route ρ_i units of flow along path P_i of length $\alpha(l_i^1)$ in iteration i then $D_{i+1} - D_i = \epsilon\rho_i\alpha(l_i^1)$.*

Proof: We can express D_{i+1} as follows:

$$\begin{aligned} D_{i+1} &= \sum_{e \in E} l_{i+1}^2(e)c(e) && (3.29)\\ &= \sum_{e \in E} l_i^2(e)c(e) + \sum_{e \in P_i}\left(\epsilon\frac{\rho_i}{c(e)}l_i^1(e)\right)c(e) && (3.30)\\ &= D_i + \epsilon\rho_i\alpha(l_i^1) && (3.31) \end{aligned}$$

The assumption follows immediately. ■

3.2.2 Analysis

We now show, that this algorithm is a $(1+\omega)$-approximation for the maximum fractional multicommodity problem with arbitrary capacities by the same analysis as used in the first section. At first, we will show that $\alpha(l_i^1) = \alpha_i \leq e^{\epsilon f_i/\beta}$. This will be used to derive a similar bound for the approximation ratio as before.

Lemma 3.2.3 $\alpha(l_i^1) = \alpha_i \leq e^{\epsilon f_i/\beta}$ *for all iterations i of algorithm (3.4)*

Proof: Observe, that by the definition of algorithm (3.4), we have

$$\begin{aligned} D_i &= D_{i-1} + \epsilon(f_i - f_{i-1})\alpha_{i-1} && (3.32) \\ &= \epsilon \sum_{j=0}^{i-1} (f_{j+1} - f_j)\alpha_j + D_0 && (3.33) \end{aligned}$$

It follows directly, that

$$D'_i = \epsilon \sum_{j=0}^{i-1} (f_{j+1} - f_j)\alpha_j \quad (3.34)$$

and by the definition of β, we also know that $D'_i \geq \beta\alpha'_i$ for all i and thus

$$\alpha'_i \leq \frac{\epsilon}{\beta} \sum_{j=0}^{i-1} (f_{j+1} - f_j)\alpha_j \quad (3.35)$$

Furthermore, we know, that $\alpha(l_i^1 - l_0^1) \geq \alpha(l_i^1) - L\delta$ and thus have

$$\alpha_i \leq \frac{\epsilon}{\beta} \sum_{j=0}^{i-1} (f_{j+1} - f_j)\alpha_j - L\delta \quad (3.36)$$

Since $\alpha_0 \leq L\delta$, we can use corollary (2.4.5) and obtain

$$\alpha_i \leq L\delta e^{\frac{\epsilon f_i}{\beta}} \quad (3.37)$$

■

Assume, that $\alpha_t \geq 1$ in iteration t. Let $F = f_t$ and hence, by the same techniques that lead to (3.15), we obtain the following bound on the approximation guarantee

$$\frac{\beta M}{F} \leq \frac{\epsilon M}{\ln L\delta} \quad (3.38)$$

Note, that M is determined as in the last section since $l_0(e) = \delta$ for all edges e. By the same argument as used in the last section, we know that finally $l_t(e) \leq (1+\epsilon)$ and thus, the amount of flow on edge e is finally at most $c(e)M$, where

$$\delta(1+\epsilon)^M \leq (1+\epsilon) \quad (3.39)$$

and hence, $M \leq \log_{1+\epsilon}(1+\epsilon)/\delta$ as before. Therefore, we obtain the following theorem.

Theorem 3.2.1 *Algorithm (3.4) is a $(1+\omega)$ approximation algorithm for the maximum fractional multicommodity flow problem with arbitrary integral edge capacities by setting $\delta = (1+\epsilon)^{1-1/\epsilon}L^{-1/\epsilon}$ and $\epsilon = 1 - \sqrt{\frac{1}{1+\omega}}$.*

3.3 Positive Linear Programming

In this section, we would like to extend the multicommodity flow algorithm of the last sections to the setting of positive linear programming (PLP). Here, we deal with the general packing LP (3.1), where all vectors and matrices consist of positive entries. Since the dual and primal program differ slightly from what we have seen so far, we will have to change the known procedure at some points.

3.3.1 The Algorithm

In the last sections, we considered multi commodity flow problems. In these cases, the matrix A contains a column for each path P and a row for each edge e. Each column can be viewed as the incidence vector of path P, i.e. $a_{rq} = 1$ iff edge r is used in path q.

Algorithms (3.1),(3.2) and (3.4) maintain a length $l(e)$ with each edge e. In iteration i they compute a shortest path P_i with respect to edge labels l_i, i.e. they compute column q, such that

$$A_q^T l_i \tag{3.40}$$

is minimized.

The expression we minimize in each iteration of the former procedures is mainly guided by duality arguments. In the multicommodity case an edge length assignment l constitutes a dual feasible solution, if

$$\texttt{length}_l(P) \geq 1 \tag{3.41}$$

for all paths P. For this reason, we compute the length of the shortest path with respect to lengths l_i in iteration i and stop if it exceeds a one since at this point, we have reached dual feasibility.

In the PLP setting, we also associate a length $l(r)$ with each row r of the matrix A. A length assignment on the rows of A constitutes a dual feasible solution iff

$$A_q^T l \geq c_q \tag{3.42}$$

where A_q is the qth column of matrix A. The latter expression is equivalent to

$$\frac{A_q^T l}{c_q} \geq 1 \tag{3.43}$$

We will later see, that in each iteration i, we compute column P_i that minimizes (3.43).

We now try to point out more similarities of the PLP setting and the arbitrary capacity case of the last section. Routing a unit of flow along a path P in the last setting means, that for each edge $e \in P$, we consume one of the $c(e)$ copies of e in the mentioned multi-graph, i.e. we saturate edge e to an extent of $1/c(e)$. Likewise, choosing column A_q of A and picking it to an extent of 1 means using a fraction of a_{rq}/b_r of the total *capacity* b_r of row r.

For the latter explanation, the positive linear programming setting fits into the framework of multicommodity flow. In the last algorithm, after we have computed path P_i in iteration i, we route ρ_i units of flow along it, where

$$\rho_i = \min_{e \in P_i} c(e) - (\mathtt{res}_i(e) \bmod c(e)) \tag{3.44}$$

Initialize:	$\forall 1 \leq r \leq m : l_0^1(r) = \delta / b_r$
	$\forall 1 \leq r \leq m : \mathtt{res}_0(r) = 0$
	$f_0 = 0$
	$D_0 = m\delta$
Iteration i:	$\alpha_i = \min_q \frac{A_q^T l_i^1}{c_q}$
	if $D_i < 1$ **then**
	$\rho_i = \min_{a_{r,P_i} \neq 0} \left\lceil \frac{b_r - (\mathtt{res}_i(r) \bmod b_r)}{a_{r,P_i}} \right\rceil$
	$f_{i+1} = f_i + \rho_i \cdot c_{P_i}$
	$x_{i+1,P_i} = x_{i,P_i} + \rho_i$
	$D_{i+1} = D_i + \epsilon(1+\epsilon)(f_{i+1} - f_i)\alpha_i$
	$\forall 1 \leq r \leq m : \mathtt{res}_{i+1}(r) = \mathtt{res}_i(r) + \rho_i a_{r,P_i}$
	$\forall 1 \leq r \leq m : l_{i+1}^1(r) = \begin{cases} (1+\epsilon) l_i^1(r) : \mathtt{res}_{i+1}(r) \leq \mathtt{res}_i(r), \\ a_{r,P_i} \neq 0 \\ l_i^1(r) : else \end{cases}$
	else
	stop
	fi
Solution:	$\left(\frac{f_t}{\log_{1+\epsilon}(1+\epsilon)/\delta}, \frac{x_t}{\log_{1+\epsilon}(1+\epsilon)/\delta} \right)$

Figure 3.5: The algorithm for Positive Linear Programming

Now, we again associate a reservoir $\mathtt{res}(r)$ with the rth row of A. Let $\mathtt{res}_i(r)$ denote its value at the beginning of the ith iteration. As in the last section, we pick column P_i to an integral extent of c such that at least one reservoir overflows.

$$\rho_i = \min_{a_{r,P_i} \neq 0} \left\lceil \frac{b_r - (\mathtt{res}_i(r) \bmod b_r)}{a_{r,P_i}} \right\rceil \tag{3.45}$$

We update the reservoir of each row r

$$\mathtt{res}_{i+1}(r) = \mathtt{res}_i(r) + \rho_i a_{rP_i} \tag{3.46}$$

Assume for now, that $a_{rq} \leq b_r$ for all r, q. By this assumption, we guarantee, that we can find c such that $\mathtt{res}_{i+1}(r) - \mathtt{res}_i(r) \leq b_r$ for all r, i.e. at most one overflow per reservoir happens.

Actually, we again consider 2 length functions on the rows of A. $l^1(r)$ changes, whenever $\mathtt{res}(r)$ overflows, i.e. whenever the space b_r of row r is completely used. $l^2(r)$ changes in every iteration, independent of the reservoir state. The motivation for 2 different length functions arises from the same argument as the one we used in the last section.

$$l_i^2(r) = \left(1 + \epsilon \frac{\texttt{res}_i(r) \bmod b_r}{b_r}\right) l_i^1(r) \tag{3.47}$$

Initially, we set $l_0^1(r) = \delta / b_r$ for all rows r, where δ is again a small constant, that we choose later. We define

$$\alpha(l_i^1) = \alpha_i = \frac{A_{P_i}^T l_i^1}{c_{P_i}} \tag{3.48}$$

As in the arbitrary capacity multicommodity setting, we denote by

$$D(l_i^2) = \sum_{1 \le r \le m} l_i^2(r) b_r \tag{3.49}$$

the value of the dual objective function with respect to length function l_i^2.

We associate a variable x_q with each column A_q of A and denote its value at the beginning of iteration i by $A_{i,q}$. Note, that picking column P_i in iteration i to an extent of ρ_i will raise the primal solution by $\rho_i \cdot c_{P_i}$. We define f_i as the value of the primal solution at the beginning of iteration i, i.e.

$$f_{i+1} - f_i = \rho_i \cdot c_{P_i} \tag{3.50}$$

The next lemma shows, how $D(l_i^2)$ is related to α_i.

Lemma 3.3.1 *For all iterations i, we have that $D(l_{i+1}^2) \le D(l_i^2) + (1+\epsilon)\epsilon(f_{i+1} - f_i)\alpha_i$*

Proof: By definition, we have that

$$D(l_{i+1}^2) = \sum_{1 \le r \le m} l_{i+1}^2 b_r \tag{3.51}$$

Assume, that we pick column A_{P_i} to an extent of ρ_i in iteration i. This means, that we raise the label $l^2(r)$ of row r by at most

$$\epsilon \frac{\rho_i a_{r,P_i}}{b_r} l_{i+1}^1(r) \tag{3.52}$$

Observe, that we used $l_{i+1}^1(r)$ instead of $l_i^1(r)$, since $\texttt{res}_i(r)$ might be near to a multiple of b_r and thus the increase $\rho_i a_{r,P_i}$ in the reservoir might contribute at most the value (3.52) to $l^2(r)$. With this observation, we can split the sum in equality (3.51).

$$\begin{aligned} D(l_{i+1}^2) &\le \sum_{1 \le r \le m} l_i^2 b_r + \sum_{a_{r,P_i} \ne 0} \epsilon \frac{\rho_i a_{r,P_i}}{b_r} l_{i+1}^1(r) b_r && (3.53) \\ &\le D(l_i^2) + \sum_{a_{r,P_i} \ne 0} \epsilon(1+\epsilon)\rho_i a_{r,P_i} l_i^1(r) && (3.54) \\ &= D(l_i^2) + \epsilon(1+\epsilon)\rho_i c_{P_i} \alpha_i && (3.55) \end{aligned}$$

Hence, we can use equation (3.50) to express $D(l^2_{i+1})$.

$$D(l^2_{i+1}) \leq D(l^2_i) + \epsilon(1+\epsilon)(f_{i+1} - f_i)\alpha_i \tag{3.56}$$

With this observation, the lemma follows. ■

In the algorithm, we define D_i to be larger than $D(l^2_i)$, i.e. let $D_0 = D(l^2_0)$ and define

$$D_{i+1} = D_i + \epsilon(1+\epsilon)(f_{i+1} - f_i)\alpha_i \tag{3.57}$$

3.3.2 The Analysis

We now show, that algorithm (3.5) is a $(1+\omega)$-approximation for the PLP problem. At first, observe, that $\beta_i = D_i/\alpha_i$ is always at least as large as the value of an optimal dual solution.

Lemma 3.3.2 *In each iteration i, $\beta_i = \frac{D_i}{\alpha_i}$ is at least as large as the value of an optimal dual solution.*

Proof: The lemma follows by the observation

$$\frac{D_i}{\alpha_i} \geq \frac{D(l^2_i)}{\alpha(l^1_i)} \geq \frac{D(l^1_i)}{\alpha(l^1_i)} \tag{3.58}$$

The right hand side of the last equation is an upper bound for the value of dual feasible solution. The assumption follows. ■

In the sequel, we again use β to denote the optimum dual solution.

Instead of deriving a closed formula for the upper bound on α_i, we derive an upper bound on D_i. We will later comment on this decision (c.f. section (3.3.3).

Lemma 3.3.3 *For all iterations i: $D_i \leq D_0 e^{\epsilon(1+\epsilon)f_i/\beta}$*

Proof: We know, that

$$D_i = D_{i-1} + (1+\epsilon)\epsilon(f_i - f_{i-1})\alpha_{i-1} \tag{3.59}$$

and thus, it follows that

$$D_i = D_0 + (1+\epsilon)\epsilon\sum_{j=0}^{i-1}(f_{j+1} - f_j)\alpha_j \tag{3.60}$$

Since we defined β as the value of an optimal dual solution, we obtain that $\alpha_i \leq D_i/\beta$ for all i. This gives us the following recurrence for D_i:

$$D_i \leq \frac{(1+\epsilon)\epsilon}{\beta}\sum_{j=0}^{i-1}(f_{j+1} - f_j)D_j + D_0 \tag{3.61}$$

The lemma follows by an application of corollary (2.4.5).

$$D_i \leq D_0 e^{\frac{(1+\epsilon)\epsilon f_i}{\beta}} \tag{3.62}$$

■

Finally, we know that $D(t) \geq 1$ and since $D_0 = m\delta$, we obtain

$$m\delta e^{\frac{\epsilon(1+\epsilon)f_t}{\beta}} \geq 1 \tag{3.63}$$

and hence

$$\frac{\beta M}{f_t} \leq \frac{\epsilon(1+\epsilon)M}{\ln \frac{1}{m\delta}} \tag{3.64}$$

M is again defined, such that f_t/M is the value of a feasible primal solution. The next lemma now bounds M.

Lemma 3.3.4 $M \leq \log_{1+\epsilon} \frac{1+\epsilon}{\delta}$

Proof: Initially, we choose $l_0^1(r) = \delta/b_r$ for each row r. Assume, that iteration i is the last iteration in which $a_{r,P_i} \neq 0$, i.e. in iteration i, we modify `res`(r) the last time. We must have had

$$D(l_i^1) \leq D_i < 1 \tag{3.65}$$

and thus it follows that

$$b_r l_i^1(r) < 1 \tag{3.66}$$

because otherwise, (3.65) would not hold. We now increase $l_i^1(r)$ by at most a factor of $(1+\epsilon)$ and thus, finally, we obtain

$$l_t^1(r) \leq \frac{1+\epsilon}{b_r} \tag{3.67}$$

Consequently, knowing initial and final value of $l^1(r)$, we know that M has to satisfy

$$\frac{\delta}{b_r}(1+\epsilon)^M \leq \frac{1+\epsilon}{b_r} \tag{3.68}$$

This immediately yields the proposed upper bound on M. ■

Plugging the upper bound on M into (3.64) yields

$$\frac{\epsilon(1+\epsilon)\ln \frac{1+\epsilon}{\delta}}{\ln(1+\epsilon)\ln \frac{1}{m\delta}} \tag{3.69}$$

Choosing $\delta = (1+\epsilon)^{1-1/\epsilon}L^{-1/\epsilon}$ as in the last section and using the fact that $\ln(1+\epsilon) \geq \epsilon - \epsilon^2/2$, we obtain

$$\frac{1+\epsilon}{(1-\epsilon/2)(1-\epsilon)} \tag{3.70}$$

We now have to choose ϵ such that the last expression is at most $(1+\omega)$. This amounts to choose ϵ, such that the following quadratic inequality is satisfied

$$\epsilon^2 - \frac{3+2\omega}{1+\omega}\epsilon + \frac{\omega}{1+\omega} \geq 0 \tag{3.71}$$

Since our choice of ϵ is limited to the range $[0, 1]$, we obtain the following final theorem.

Theorem 3.3.1 *Algorithm (3.5) is a $(1+\omega)$-approximation for the positive linear programming packing problem by choosing $\delta = (1+\epsilon)^{1-1/\epsilon} m^{-1/\epsilon}$ and $\epsilon \leq \frac{3+2\omega-\sqrt{9+8\omega}}{2+2\omega}$*

Now, suppose that there are r, q, such that $a_{rq} > b_r$. In this case we define a new problem, by introducing new variables $x' = zx$, where

$$z = \max\{\max_{rq} a_{rq}, \max_{r} 1/b_r\} \tag{3.72}$$

We solve the problem

$$\begin{array}{ll} \textbf{max} & {c'}^T x' \\ \textbf{s.t.} & {A'}^T x' \end{array} \tag{3.73}$$

where c' and A' result from c and A, respectively, by dividing its entries by z. Observe, that for this problem, we have $a'_{rq} \leq b_r$ for all r, q. There is a one-to-one correspondence between a solution to the latter problem and a solution to the original problem. We divide the entries of an optimal solution x' to (3.73) by z and obtain an optimal solution for the original problem.

3.3.3 Comments on the Analysis

In the analysis of the positive linear programming case, we mainly used a recurrence on D_i, while in the last sections, we developed an analysis based on α_i. Both were developed using a dependence of the sum D on the α values of the former iterations. Why did we choose the α-recurrence in the first section and the D-recurrence in the current section? The difference becomes clear, if we compare the closed formulas for the upper bounds on D_i and α_i. They both have the general form

$$ce^{ai} \tag{3.74}$$

For the α-recurrence, we chose $c = L\delta$, which is an upper bound on α_0. In the D-recurrence, we have $c = D_0 = m\delta$. The factor c turns out to be important for the analysis of the running time of the algorithms. In the previous sections, we bounded the maximum number of reservoir overflows by $\log_{1+\epsilon} 1/\delta$. The choice of δ was mainly guided by c, since in bounding the relative performance guaranty, we got a factor $\log 1/c$ in the denominator (see (3.15) and (3.64)). Consequently, in the first sections, we chose $\delta = (1+\epsilon)^{1-1/\epsilon} L^{-1/\epsilon}$. In the PLP section, we selected the same choice of δ with the only difference of replacing L by m.

Observe now, that it is reasonable in the positive linear programming case to choose $c = D_0$ rather than choosing it to be α_0. As we saw in algorithm (3.5), $D_0 = m\delta$, while α_0 would have involved the entries of matrix A, precisely $\alpha_0 \leq L_p\delta/c_p$. Here, L_j denotes the sum of the entries of column A_j and c_j is the benefit scalar of row A_j. Furthermore, L_p/c_p maximizes the ration L_j/c_j over all columns j.

The running time and the integrality of the solution involve the factor $\log_{1+\epsilon} 1/\delta$ as we shall see at the end of this chapter. Thus, our decision to take the D-recurrence basically replaced $L_p\delta/c_p$, by m in the log factor, i.e. the running time and integrality turn out to be strongly polynomial instead of polynomial.

3.4 Fractional Packing

We now show how to apply our algorithms to the fractional packing problem, addressed by Plotkin, Shmoys and Tardos [19] and Young [18]. We address the following problem

$$\exists ? x \in P : Ax \leq b \tag{3.75}$$

where $b > 0$ and P is a convex polytope with $Ax \geq 0$ for all $x \in P$. It is assumed to have a fast optimization subroutine that computes $\tilde{x} = \min\{x : cx, x \in P\}$, where $c = y^T A$. We denote $\tilde{x}$ as $M(y)$. We call $x \in P$ σ-good[1] if $Ax \leq (1+\sigma)b$. The following optimization version of this problem is used to find such a point.

$$\begin{aligned} \textbf{min} \quad & \lambda \\ \textbf{s.t.} \quad & Ax \leq \lambda b \\ & x \in P \end{aligned} \tag{3.76}$$

Observe, that a $(1+\sigma)$-approximate solution (λ, x) to (3.76) suffices to decide whether the original problem has a σ-good solution. In the case of $\lambda \leq (1+\sigma)$, x is such a point. Let (λ^*, x^*) denote an optimal solution to (3.76). Since (λ, x) is $(1+\sigma)$-approximate, we have that $\frac{\lambda}{1+\sigma} \leq \lambda^*$. If $\lambda > (1+\sigma)$, we have $\lambda^* > 1$ and thus, no optimal solution exists. A procedure that either decides that no optimal solution exists or returns a σ- good solution is called a *relaxed decision procedure.*

To solve problem (3.76) we solve a more general LP, in which we are looking for a point x out of the cone defined by P, which we will denote by $\mathcal{C}(P)$, such that $Ax \leq b$. Let in the following be $V_1, \ldots, V_p$ the set of vertices of P and additionally let $x = \sum_i \nu_i V_i$, i.e. $(\nu_1, \ldots, \nu_p)$ is the parameter vector of x with respect to V. Then, we solve

$$\begin{aligned} \textbf{max} \quad & \sum_i \nu_i \\ \textbf{s.t.} \quad & Ax \leq b \\ & x \in \mathcal{C}(P) \end{aligned} \tag{3.77}$$

which is equivalent to

$$\begin{aligned} \textbf{max} \quad & \sum_i \nu_i \\ \textbf{s.t.} \quad & (AV)\nu \leq b \\ & \nu \geq 0 \end{aligned} \tag{3.78}$$

[1] In [19, 18] such a solution is called σ-approximate. We use σ-good since we used σ-approximate in a slightly different manner.

We shall later use the PLP algorithm to obtain a $(1+\sigma)$-approximate solution (3.78). We can use the PLP algorithm, since every column of (AV) is positive by the definition of the problem (remember: $Ax \geq 0, \forall x \in P$). Assume now, that we have an optimal solution ν^* to (3.78). Assume furthermore, that $z^* = \sum_i \nu_i^*$. By dividing the entries of ν^* by z^* we obtain the parameters of a point in P, since $\frac{\nu^*}{z^*}$ describes a convex combination of vertices of P. Let now $x^* = \sum_i \nu_i^* V_i$ and observe, that $(1/z^*, x^*/z^*)$ is a solution of (3.76). We shall now claim optimality.

Lemma 3.4.1 *If ν^* is an optimal solution of (3.78) with value z^*, then $(1/z^*, x^*/z^*)$ is an optimal solution of (3.76)*

Proof: Suppose, that there is an $x' \in P$, such that $Ax' \leq \frac{1}{z'}b$ and $1/z' < 1/z^*$, i.e $z' > z^*$. Let furthermore be $x' = \sum_i \nu_i' V_i$, then $z'x'$ is a solution of (3.78) as $AVz'\nu' = z'Ax' \leq b$ and the value of this solution is $z' \sum_i \nu_i' = z' > z^*$ which is the value of the optimal solution to (3.78). A contradiction. ■

Assume, that we obtained a $(1+\sigma)$-approximate solution ν to problem (3.78). Let $z = \sum_i \nu_i$ and $x = \sum_i \nu_i V_i$. Hence, we have

$$(1+\sigma)z \geq z^* \tag{3.79}$$

Equivalently, this means that

$$\frac{1}{z} \leq \frac{1+\sigma}{z^*} \tag{3.80}$$

Thus, it follows that $(1/z, x/z)$ is a $(1+\sigma)$-approximate for problem (3.76).

Theorem 3.4.1 *The developed procedure for packing problems can also be applied to the general setting of a fractional packing problem and constitutes a $(1+\sigma)$ approximation algorithm for problem (3.76)*

Therefore, we can use the known methods for packing to develop a relaxed decision procedure for fractional packing problems.

In the former sections, the polytope P was just the standard simplex, i.e. the matrix V was the identity matrix and thus we did not differentiate between parameters and points from P because both representations were equivalent.

3.5 Generalized Packing

In the generalized packing problem we are faced with the same setting as in the last section, i.e. we are obliged to find a point $x \in P$ such that $Ax \leq b$. But this time, vector b is not necessarily positive. Given a parameter $\sigma > 0$ and a positive tolerance vector d, we call a point $x \in P$ σ-good, if $Ax \leq b + \sigma d$. Plotkin, Shmoys and Tardos show in [19] how to reduce this problem to a fractional packing problem. We therefore define

$$\rho = \max_i \max_{x \in P} \frac{|a_i x - b_i|}{d_i} \tag{3.81}$$

where a_i denotes the ith row of matrix A. This value ρ is called the *width* of P in [19]. Let now be $\tilde{P} = \{(x,s) : x \in P, s = 1\}$ a convex polytope and try to find $(x,s) \in \tilde{P}$, such that $Ax + (\rho d - b)s \leq \tilde{b} = (\rho d, \ldots, \rho d)^T$. Observe at first, that $\tilde{b}$ is positive and besides that, $Ax + (\rho d - b)s \geq 0$ for all $(x,s) \in \tilde{P}$, since $|a_i x - b_i| \leq \rho d$ for all i. Notice, that there is a one to one correspondence between this packing problem and the original general packing problem. In particular, we obtain a σ-good solution to the general problem by computing an σ/ρ-good solution for the packing problem.

3.6 Performance and Integrality

This section is devoted to the running time of the presented algorithms. We will also concentrate on the integrality of the primal solutions, that our procedures obtain. This means, that we will address the important question of solution-granularity.

The running time of the algorithms will be given in terms of iterations used. We neglect the running time of the optimization subroutine, which has to be called within each iteration, i.e. we will for instance not account for the running time of the shortest path procedure for the maximum multicommodity flow problem.

The basic observation for the analysis of the maximum number of iterations used by our procedures is the fact that the reservoir associated with an edge or with a row respectively, can overflow only $\log_{1+\epsilon} 1/\delta$ times in all of our algorithms. We ensure that at least one of the reservoirs overflows in each iteration and therefore, the maximum number of iterations is bounded by $O(m \log_{1+\epsilon} 1/\delta)$ in all cases.

The exact running time depends on the choice of δ. In the MFMF case with unit and arbitrary capacities we set $\delta = (1+\epsilon)^{1-1/\epsilon} L^{-1/\epsilon}$. By this, we obtain a running time of

$$\begin{aligned} m \log_{1+\epsilon} 1/\delta &= m \frac{(1/\epsilon - 1)\log(1+\epsilon) + 1/\epsilon \log L}{\log(1+\epsilon)} && (3.82) \\ &= m\left(\frac{1}{\epsilon} - 1 + \frac{\log L}{\epsilon \log(1+\epsilon)}\right) && (3.83) \\ &= O\left(\frac{m \log L}{\epsilon \log(1+\epsilon)}\right) && (3.84) \end{aligned}$$

The same applies to positive linear programming, with the only change of replacing L by m in equation (3.84). Thus we get a running time of $O\left(\frac{m \log m}{\epsilon \log(1+\epsilon)}\right)$. In the case of fractional packing and generalized packing δ is defined as for the PLP setting and m denotes the row size of AV or $(AV|\rho d - b)$ respectively, which is equivalent to the number of rows in A.

We observe, that the running time is polynomial in the size of the system matrix A and in $(\epsilon \log(1+\epsilon))^{-1}$ and since it does not involve the entries of matrix A, it is strongly polynomial.

The analysis of the integrality is similar. We divide the primal solution by $M \leq \log_{1+\epsilon} \frac{1+\epsilon}{\delta}$ to obtain primal feasibility. Therefore, as before, the integrality is dependent on the choice of δ. In the MFMF case, independent of the choice of capacities, we get an integrality of $O\left(\frac{m \log L}{\epsilon \log(1+\epsilon)}\right)$, where L is defined as before.

For the case of positive linear programming note, that we assumed $a_{rq} \leq b_r$ for all r, q. For this case we achieve the same integrality as in the MFMF case with L replaced m. The same also holds for the fractional packing and generalized packing cases.

In the case of $a_{rq} > b_r$ for some r, q note, that we reduced this problem to the case $a_{rq} \leq b_r$ by a redefinition of the variables and that we obtained an optimal solution to the original problem from a solution to the auxiliary problem by dividing the entries of the solution vector by $z = \max\{\max_{rq} a_{rq}, \max_r 1/b_r\}$. Thus, the integrality for the cases of large a_{rq} and relatively small b_r is only $O\left(\frac{z}{\epsilon \log(1+\epsilon)} \log m\right)$. It is not any longer strongly polynomial. This was the main reason for subdividing the presentation of the PLP algorithm into two cases.

Observe, that the obtained dual solutions will not have a good integrality. We start with an initial edge label of δ or δ/b_r, respectively, at the beginning of the procedures and raise these labels by multiplying them continuously by $1+\epsilon$. It is hard to argue about the final integrality of the dual solution. This fact was the basic motivation to develop different procedures for covering and packing problems, though both are duals of each other.

Chapter 4

Covering

In this chapter we discuss problems which can be described by linear programs of the form

$$\begin{aligned} \textbf{min} \quad & c^T x \\ \textbf{s.t.} \quad & A^T x \geq b \end{aligned} \tag{4.1}$$

Intuitively, assume that we want to assign weights on the columns ,i.e. on the x_j's such that i is covered by at least b_i, where each unit of weight on the jth column puts a_{ij} weight on element i. For this reason we call this type of problem a *covering problem.* The algorithms we present rely heavily on the dual of this LP. We therefore introduce it now

$$\begin{aligned} \textbf{max} \quad & y^T b \\ \textbf{s.t.} \quad & A^T y \leq c \end{aligned} \tag{4.2}$$

Again, the intuitive view of this program suggests its name. We call problems having a linear programming formulation of the latter style *packing problems.*

The basic ideas we use to develop algorithms for general covering problems arise from the greedy algorithm for *integral set-cover* (ISC) that was first analyzed independently by Johnson and Lovász [11, 13] and it was later extended by Chvátal [3].

4.1 Integral Set-Cover

In the ISC problem we have a universe U of elements $e_1, \ldots, e_m$ and a collection of subsets of U, $\mathcal{S} = \{S_1, \ldots, S_n\}$. We are now interested in finding the minimal subset S^* of $\mathcal{S}$, such that all elements from the universe are covered by at least one set out of S^*.

In this section we address the weighted version of this problem in which we are interested in finding the minimal weight collection of sets that cover each of the elements. This problem fits into the covering framework by defining c to be the weight vector, b to consist of 1-entries only and A to be the $\{0, 1\}$ system matrix, such that $a_{rq} = 1$ iff element r is in set q.

The greedy algorithm for this problem starts with an empty collection of sets. In each iteration i, we pick the set S_i from $\mathcal{S}$ which minimizes $c(S)/|T_i \cap S|$, where T_i contains the elements from U, which have not been covered in iterations $0, \ldots, i-1$. Let s_i be defined as

$$s_i = \frac{c(S_i)}{|T_i \cap S_i|} \tag{4.3}$$

Intuitively the set S_i minimizes the ratio of the cost to the benefit we incur by picking it. The algorithm stops if no uncovered element remains.

The approximation ratio of this algorithm can be derived from a duality argument. The dual of ISC is a packing problem. Intuitively a solution to it can be viewed as a volume assignment to the elements of U, such that the elements of set S fit into a box of volume $c(S)$. It follows from duality theory that a feasible volume assignment on U induces a lower bound on the optimum solution value of an instance of the ISC problem.

Hence, we shall maintain a dual variable for each element while running the greedy algorithm. We denote the volume assigned to element e as $y(e)$. The way we assign volumes is the following: assume that element e gets covered for the first time when set S_i is picked. We than assign the defined ratio s_i to $y(e)$. We do so for each of the elements from $T_i \cap S_i$.

Let $x_i(S)$ denote the value of the LP variable associated with set S at the beginning of iteration i. Let $P(x)$ denote the primal objective function value induced by variables x, i.e.

$$P(x) = \sum_{S \in \mathcal{S}} c(S)x(S) \tag{4.4}$$

In the following, we define the abbreviation P_i to denote $P(x_i)$, i.e. P_i is the objective function value at the beginning of iteration i. Similarly, we denote by $D(y)$ the dual objective function value induced by y, i.e.

$$D(y) = \sum_{e \in U} y(e) \tag{4.5}$$

Furthermore, let D_i be the abbreviation for $D(y_i)$, i.e. the dual objective function value at the beginning of iteration i.

Observe, that the dual and primal variables for the element and the sets, respectively, are set at most once during the execution of algorithm (4.1). For the latter reason, the pseudo code of algorithm (4.1) mentions only changes in the primal and dual variables. If a variable is not mentioned we assume, that its value in iteration $i+1$ is equivalent to its value in iteration i.

Lemma 4.1.1 *In each iteration i, we have that $P_i = D_i$*

Proof: Since initially, $x_0(S) = 0$ and $y_0(e) = 0$ for all sets S and all elements e, respectively, we have $P_0 = D_0$. Suppose, the assumption holds for iterations $0, \ldots, i$. In iteration i, we pick set S_i that minimizes the ratio

$$s_i = \frac{c(S_i)}{|T_i \cap S_i|} \tag{4.6}$$

We now set $y_i(e) = s_i$ for all $e \in T_i \cap S_i$ and thus

$$D_{i+1} = D_i + |T_i \cap S_i| s_i = D_i + c(S_i) \tag{4.7}$$

Initialize:	$\forall e \in U : y_0(e) = 0$
	$\forall S \in \mathcal{S} : x_0(S) = 0$
	$T_0 = U$
Iteration i:	**if** $T_i = \emptyset$ **then stop**
	$s_i = \min_{S \in \mathcal{S}} \frac{c(S)}{T_i \cap S}$
	$\forall e \in S_i : y_{i+1}(e) = s_i$
	$x_{i+1}(S_i) = 1$
	$T_{i+1} = T_i - S_i$
Solution:	$\left(x_t, \frac{y_t}{\mathcal{H}(d)}\right)$

Figure 4.1: The Integral Set-Cover Algorithm.

and since the primal objective function value also increases by $c(S_i)$ in iteration i, the assumption follows. ■

Suppose, that the described algorithm terminates in iteration t. The primal solution x_t is feasible since all elements are covered. However, the dual solution need not to be feasible. A feasible dual solution y has to satisfy

$$\sum_{e \in S} y(e) \leq c(S) \tag{4.8}$$

for all sets $S \in \mathcal{S}$. The next lemma bounds the sum of the dual values of the elements of a set.

Lemma 4.1.2 *For any set S, $\sum_{e \in S} y_t(e) \leq c(S)\mathcal{H}(|S|)$*

Proof: Let $\nu_1, \ldots, \nu_{|S|}$ be the elements of set S in the order, greedy covers them. Elements which are covered for the first time by the same set are ordered arbitrarily. We shall now upper bound the volume that is assigned to element ν_j. At the time we cover this element, there are at least $|S| - j + 1$ uncovered elements from S. Hence, $y_t(e) \leq c(S)/(|S| - j + 1)$. Summing over all elements yields the assumption. ■

Thus, the dual variables we maintain should be divided by $\mathcal{H}(d)$, where d is the size of the largest set. This scaling factor is also the factor by which dual feasible and primal feasible solution differ. Since every dual feasible solution is smaller than the optimal primal solution we immediately obtain the following corollary.

Corollary 4.1.1 *The described greedy algorithm is a $\mathcal{H}(d)$-approximation algorithm for the ISC*

4.2 Fractional Set-Cover

From now on, we concentrate on the relaxation of the problem described in the last section. Instead of allowing the algorithm to pick sets only to an integral extent, we now accept also fractional solution vectors.

4.2.1 The Algorithm

The algorithm for fractional set-covering again proceeds in iterations. We keep primal and dual variables at the same time. Let in the following denote $x_i(S)$ the primal variable of set S at the beginning of iteration i. Similarly, $y_i(e)$ denotes the dual volume associated with element e.

In contrast to the last section, we also keep a requirement $r_i(e)$ with each element. This becomes necessary, since we pick sets to small extents and thus need to cover each element several times in order to meet its coverage demand.

Now, we initially set $x_0(S) = 0$ for each set $S \in \mathcal{S}$. We also initialize $y_0(e) = 0$ for all $e \in U$. The requirements $r_0(e)$ are uniformly set to 1.

In iteration i, we compute set S_i, which minimizes the ratio

$$\frac{c(S)}{\sum_{\substack{e \in S \\ r_i(e) > \delta}} r_i(e)} \tag{4.9}$$

which is the ratio of the cost of the set S to the requirement of the its *active* elements. Elements are called active, if they have a requirement of more than δ. If the requirement of an element drops below δ we take it out of consideration, i.e. we mark this element as *inactive*.

After having computed set S_i in iteration i, we pick it to an extent of ϵ, i.e.

$$x_{i+1}(S_i) = x_i(S) + \epsilon \tag{4.10}$$

At the same time, we also alter the requirements of the elements of S_i:

$$\forall e \in S_i, r_i(e) > \delta : r_{i+1}(e) = \frac{r_i(e)}{1+\epsilon} \tag{4.11}$$

Furthermore, we update the dual variables

$$\forall e \in S, r_i(e) > \delta : y_{i+1}(e) = y_i(e) + \frac{\epsilon}{1+\epsilon} s_i r_i(e) \tag{4.12}$$

The procedure terminates, when no active element remains.

4.2.2 The Analysis

The structure of the analysis is quite similar to the greedy analysis from the last section. We start by pointing out a relationship between the dual and primal objective function values in

Initialize:	$\forall e \in U : r_0(e) = 1, y_0(e) = 0$
	$\forall S \in \mathcal{S} : x_0(S) = 0$
Iteration i:	**if** $\nexists e \in U : r_i(e) > \delta$ **then stop**
	$s_i = \min_{S \in \mathcal{S}} \frac{c(S)}{\sum_{\substack{e \in S \\ r_i(e) > \delta}} r_i(e)}$
	$x_{i+1}(S_i) = x_i(S_i) + \epsilon$
	$\forall e \in S_i, r_i(e) > \delta : r_{i+1}(e) = \frac{r_i(e)}{1+\epsilon}$
	$y_{i+1}(e) = y_i(e) + s_i \frac{\epsilon}{1+\epsilon}$
Solution:	$\left(\frac{x_t}{\epsilon \log_{1+\epsilon} 1/\delta}, \frac{y_t}{\ln(1+\epsilon)d/\delta} \right)$

Figure 4.2: The algorithm for fractional set-cover.

all iterations i. Afterwards, we show how to obtain primal and dual feasible solutions from the constructed vectors x and y. Finally, we put everything together and hence prove an upper bound on the approximation guarantee.

We again define $P(x)$ as the primal objective function value induced by variables x, i.e.

$$P(x) = \sum_{S \in \mathcal{S}} c(S)x(S) \tag{4.13}$$

and the abbreviation P_i to denote $P(x_i)$. Similarly let

$$D(y) = \sum_{e \in U} y(e) \tag{4.14}$$

be the dual objective function value induced by y and for brevity define $D_i = D(y_i)$.

Lemma 4.2.1 *In each iteration i, we have $P_i \leq (1+\epsilon)D_i$*

Proof: Observe, that initially $P_0 = D_0$ and hence the assumption holds for $i = 0$. For $i > 0$, we pick set S_i to an extent of ϵ, i.e.

$$P_{i+1} = P_i + \epsilon c(S_i) \tag{4.15}$$

The dual objective function value in iteration $i+1$ arises from D_i in the following way

$$\begin{aligned} D_{i+1} &= D_i + \frac{\epsilon}{1+\epsilon} s_i \sum_{\substack{e \in S_i \\ r_i(e) > \delta}} r_i(e) && (4.16) \\ &= D_i + \frac{\epsilon}{1+\epsilon} c(S_i) && (4.17) \end{aligned}$$

and hence

$$\begin{aligned} P_{i+1} &= P_i + \epsilon c(S_i) && (4.18) \\ &\leq (1+\epsilon)\left(D_i + \frac{\epsilon}{1+\epsilon} c(S_i) \right) && (4.19) \\ &= (1+\epsilon) D_{i+1} && (4.20) \end{aligned}$$

■

Assume, that the given procedure terminates in iteration t. Primal and dual variables x_t and y_t need not necessarily constitute feasible solutions. The following 2 lemmas prove scaling factors.

Lemma 4.2.2 $\frac{x_t}{\epsilon \log_{1+\epsilon} 1/\delta}$ *is a primal feasible solution to the given problem instance*

Proof: Note, that $r_0(e) = 1$ for all $e \in U$. Since finally all elements are inactive and since each time the requirement of an element decreases by $(1+\epsilon)$, the coverage increases by ϵ, we obtain that each element is covered at least to an extent of ϵk. Here, k is the least number such that

$$(1+\epsilon)^{-k} \leq \delta \tag{4.21}$$

and thus $k \geq \log_{1+\epsilon} 1/\delta$. This yields the lemma. ■

Lemma 4.2.3 $\frac{y_t}{\ln \frac{d(1+\epsilon)}{\delta}}$ *is a dual feasible solution to the given problem instance*

Proof: Observe, that y is dual feasible, if

$$\sum_{e \in S} y_t(e) \leq c(S) \tag{4.22}$$

for all sets $S \in \mathcal{S}$. In the following, define

$$Y_i(S) = \sum_{e \in S} y_i(e) \tag{4.23}$$

and

$$R_i(S) = \sum_{e \in S} r_j(e) \tag{4.24}$$

Suppose now, $R(S)$ decreases by β, i.e. $R_{i+1}(S) = R_i(S) - \beta$. Let Δ be the set of elements from U, that changed their requirements in iteration i, i.e. the elements which caused the decrease in $R(S)$. Thus,

$$\beta = \sum_{e \in \Delta} \frac{\epsilon}{1+\epsilon} r_i(e) \tag{4.25}$$

Simultaneously, $Y(S)$ increases. This increase can be expressed in the following way

$$\sum_{e \in \Delta} \frac{\epsilon}{1+\epsilon} s_i r_i(e) = \beta s_i \tag{4.26}$$

Observe, that initially $Y_0(S) = 0$. Since $R_0(S) = |S|$ and finally, $R_t(S) > \frac{\delta}{1+\epsilon}$, we obtain that $Y_t(S)$ is bounded by the area under the curve $c(S)/x$, where x goes from $\frac{\delta}{1+\epsilon}$ to $|S|$ (see figure (4.3)). Hence,

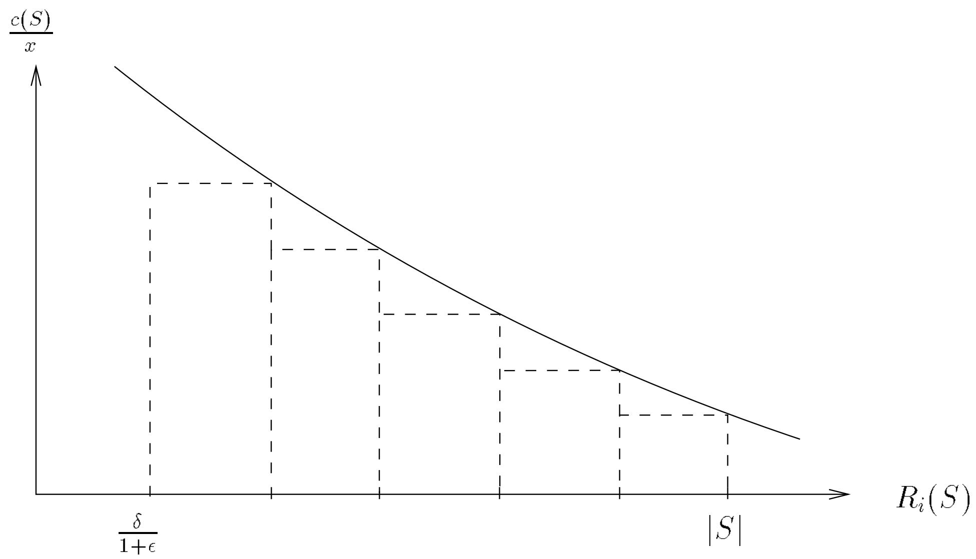

Figure 4.3: The relation between $R_i(S)$ and $Y_t(S)$. A dashed square surrounds an area of size $s_i\beta$ from equation (4.26). The whole dashed area is the final value of $Y_t(S)$.

$$Y_t(S) \le \int_{\frac{\delta}{1+\epsilon}}^{|S|} \frac{c(S)}{x} dx = \ln \frac{(1+\epsilon)|S|}{\delta} \tag{4.27}$$

The right hand side of the last equation is bounded by

$$\ln \frac{(1+\epsilon)d}{\delta} \tag{4.28}$$

where d is the size of the largest set. This observation shows the assumption. ■

Finally, we bound the ratio of primal to dual feasible solution.

$$\frac{P_t \ln \frac{(1+\epsilon)d}{\delta}}{D_t \epsilon \log_{1+\epsilon} 1/\delta} = \frac{P_t \ln(1+\epsilon) \ln \frac{(1+\epsilon)d}{\delta}}{D_t \epsilon \ln 1/\delta} \tag{4.29}$$

With lemma (4.2.1) and the fact that $\ln(1+\epsilon)/\epsilon \le 1$ the latter term is bounded from above by

$$\frac{(1+\epsilon) \ln \frac{(1+\epsilon)d}{\delta}}{\ln 1/\delta} \tag{4.30}$$

Now, we set δ to $((1+\epsilon)d)^{-1/\epsilon}$ and the last term transforms to $(1+\epsilon)^2$. We now choose ϵ, such that this term is at most $1+\omega$.

Theorem 4.2.1 *The greedy algorithm for the FSC problem is a $(1+\omega)$- approximation by setting $\delta = ((1+\epsilon)d)^{-1/\epsilon}$ and $\epsilon \le \sqrt{1+\omega} - 1$.*

4.3 Fractional Multi-Cover

The fractional multi-cover problem (FMC) relaxes the FSC setting in that we allow an integral coverage requirement for each element $e \in U$. Let in the following be $b(e)$ the extent to which a feasible solution should cover element e.

4.3.1 Dual Variables y^1 and y^2

In the following we introduce an intuitive view of an instance of the FMC problem. We view each element e as consisting of $b(e)$ *copies* $e_1, \ldots, e_{b(e)}$ (see figure (4.4)). We show how to derive the FMC algorithm from a variant of the FSC algorithm, which we apply on the modified problem.

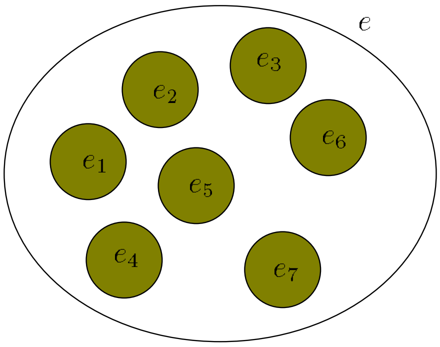

Figure 4.4: We view each element e of the universe in the fractional multi-cover setting as encapsulating $b(e)$ copies of unit requirement. In this case: $b(e) = 7$

We again consider each element $e \in U$ to have an associated requirement $r(e)$ and dual variable $y^1(e)$.

$$r_i(e) = \max_{1 \le j \le b(e)} r_i(e_j) \tag{4.31}$$

$$y_i^1(e) = \max_{1 \le j \le b(e)} y_i(e_j) \tag{4.32}$$

Thus, the copy with the largest requirement or dual value, respectively, dictates the value of the variable which is associated with element e.

As before, in each iteration i, we pick set S_i which minimizes the ratio

$$\frac{c(S)}{\sum_{\substack{r_i(e) > \delta \\ e \in S}} r_i(e)} \tag{4.33}$$

If $e \in S_i$ is active, i.e. $r_i(e) > \delta$, , we reduce the requirement of its largest copy. Denote in the following the largest copy of e by e_p. Then the change in the requirement of the copy is defined as

$$r_{i+1}(e_p) = \frac{r_i(e_p)}{1 + \epsilon} \tag{4.34}$$

Observe, that $r(e)$ and $y^1(e)$ change when all copies $e_1, \dots, e_{b(e)}$ are covered to the same extent, i.e. when e has been covered a multiple of $b(e)$ times. We capture this idea by introducing a *reservoir* $\texttt{res}(e)$ with each element $e \in U$. Denote by $\texttt{res}_i(e)$ the value of this reservoir at the beginning of iteration i. Initially, set $\texttt{res}_0(e) = 0$ for all elements $e \in U$. Suppose, we pick set S_i in iteration i to an extent of ϵ. Then, one of the copies of each $e \in S_i$ is covered. Hence, we raise the reservoir of the active elements of S_i

$$\texttt{res}_{i+1}(e) = \texttt{res}_i(e) + 1 \tag{4.35}$$

In the case of $\texttt{res}_i(e) \bmod b(e) = 0$, we know that all copies of element e have been covered to the same extent, i.e. we decrease $r(e)$ and increase $y^1(e)$

$$\begin{aligned} r_{i+1}(e) &= \frac{r_i(e)}{1+\epsilon} && (4.36) \\ y^1_{i+1}(e) &= y^1_i(e) + s_i r_i(e) \frac{\epsilon}{1+\epsilon} && (4.37) \end{aligned}$$

We define the dual objective function value at the beginning of iteration i as

$$\begin{aligned} D_i &= \sum_{e \in U} y^1_i(e) b(e) + \sum_{e \in U} \frac{\epsilon(\texttt{res}_i(e) \bmod b(e))}{(1+\epsilon) b(e)} r_i(e) s_{i-1} b(e) && (4.38) \\ &= \sum_{e \in U} \left(y^1_i(e) + \frac{\epsilon(\texttt{res}_i(e) \bmod b(e))}{(1+\epsilon) b(e)} r_i(e) s_{i-1} \right) b(e) && (4.39) \end{aligned}$$

By this, we account for the coverage of the copies of e. Define now

$$y^2_i(e) = y^1_i(e) + \frac{\epsilon(\texttt{res}_i(e) \bmod b(e))}{(1+\epsilon) b(e)} r_i(e) s_{i-1} \tag{4.40}$$

We define $D(y) = \sum_{e \in U} y(e) b(e)$, i.e. the dual objective function value induced by y. Observe, that $D_i = D(y^2_i)$.

4.3.2 The Algorithm

The algorithm itself works in the same manner as the FSC algorithm does. In each iteration i, we compute S_i, which minimizes the ratio

$$\frac{c(S)}{\sum_{\substack{e \in S \\ r_i(e) > \delta}} r_i(e)} \tag{4.41}$$

We denote the minimal ratio in iteration i by s_i. We pick set S_i to an extent of $\rho_i \epsilon$ such that at least one of the element reservoirs overflows, i.e.

$$\rho_i = \min_{\substack{e \in S_i \\ r_i(e) > \delta}} b(e) - (\texttt{res}_i(e) \bmod b(e)) \tag{4.42}$$

Afterwards, we update the reservoirs of the elements from S_i

$$\forall e \in S_i : \texttt{res}_{i+1}(e) = \texttt{res}_i(e) + \rho_i \tag{4.43}$$

If now a reservoir overflows, i.e. $\mathtt{res}_{i+1}(e) \bmod b(e) = 0$ and $r_i(e) > \delta$, we update $r(e)$ and $y^1(e)$ simultaneously.

$$r_{i+1}(e) = \frac{r_i(e)}{1+\epsilon} \tag{4.44}$$

$$y^1_{i+1}(e) = y^1_i(e) + s_i r_i(e) \frac{1}{1+\epsilon} \tag{4.45}$$

Initialize:	$\forall e \in U : r_0(e) = 1, y^1_0(e) = 0, y^2_0(e) = 0, \mathtt{res}_0(e) = 0$
	$\forall S \in \mathcal{S} : x_0(S) = 0$
Iteration i:	**if** $\not\exists e \in U : r_i(e) > \delta$ **then stop**
	$s_i = \min_{S \in \mathcal{S}} \frac{c(S)}{\sum_{e \in S, r_i(e) > \delta} r_i(e)}$
	$\rho_i = \min_{e \in S_i, r_i(e) > \delta} b(e) - (\mathtt{res}_i(e) \bmod b(e))$
	$x_{i+1}(S_i) = x_i(S_i) + \rho_i \epsilon$
	$\forall e \in S_i, r_i(e) > \delta : \mathtt{res}_{i+1}(e) = \mathtt{res}_i(e) + \rho_i$
	$\forall e \in S_i, r_i(e) > \delta, \mathtt{res}_{i+1}(e) \bmod b(e) = 0 :$
	$r_{i+1}(e) = \frac{r_i(e)}{1+\epsilon}$
	$y^1_{i+1}(e) = y_i(e) + s_i \frac{\epsilon}{1+\epsilon}$
Solution:	$\left(\frac{x_t}{\epsilon \log_{1+\epsilon} 1/\delta}, \frac{y_t}{\ln(1+\epsilon) d/\delta} \right)$

Figure 4.5: The algorithm for fractional multi cover.

4.3.3 The Analysis

The structure of the following analysis is quite similar to what we have seen in the last section. Assume, that we terminate in iteration t. Thus, we shall show at first, that $P(x_t) \leq (1+\epsilon)D(y^1_t)$. Afterwards, we comment on how to determine the scaling factors to obtain primal and dual feasible solution values. Finally, we put everything together and obtain an upper bound on the approximation guarantee.

Lemma 4.3.1 *$P_i \leq (1+\epsilon)D_i$ for all iterations i*

Proof: Initially, we have $P_0 = 0 = D_0$ and thus the assumption holds. For iteration $i \geq 0$, assume that $P_i \leq (1+\epsilon)D_i$. We pick set S_i to an extent of $c\epsilon$, i.e.

$$P_{i+1} = P_i + \epsilon \rho_i c(S_i) \tag{4.46}$$

The change in $D(y^2)$ can be expressed in the following way

$$\begin{aligned}
D_{i+1} &= D_i + \sum_{\substack{e \in S_i \\ r_i(e) > \delta}} s_i r_i(e) \frac{\epsilon \rho_i}{(1+\epsilon) b(e)} b(e) && (4.47)\\
&= D_i + \rho_i s_i \frac{\epsilon}{1+\epsilon} \sum_{\substack{e \in S_i \\ r_i(e) > \delta}} r_i(e) && (4.48)\\
&= D_i + \frac{\epsilon}{1+\epsilon} \rho_i c(S_i) && (4.49)
\end{aligned}$$

and hence

$$\begin{aligned}
P_{i+1} &= P_i + \epsilon \rho_i c(S_i) && (4.50)\\
&\leq (1+\epsilon) \left(D_i + \frac{\epsilon}{1+\epsilon} \rho_i c(S_i) \right) && (4.51)\\
&= (1+\epsilon) D_{i+1} && (4.52)
\end{aligned}$$

■

Observe, that P_i denotes $P(x_i)$ and D_i denotes $D(y_i^2)$. For the final bound of the approximation guarantee, we need that $P(x_t) \leq (1+\epsilon) D(y_t^1)$. We therefore prove

Corollary 4.3.1 $P(x_t) \leq (1+\epsilon) D(y_t^1)$

Proof: From lemma (4.3.1), we know that $P(x_i) \leq (1+\epsilon) D(y_i^2)$ for all iterations i. We also know, that finally all elements are inactive and thus $\texttt{res}_t(e) \bmod b(e) = 0$ for all elements. The assumption follows immediately since $y_t^1(e) = y_t^2(e)$ for all e by definition. ■

We now show, how to determine the scaling factors to obtain primal and dual feasible solutions from x_t and y_t.

Lemma 4.3.2 $\frac{x_t}{\epsilon \log_{1+\epsilon} 1/\delta}$ *is a primal feasible solution*

Proof: Initially, $r_0(e) = 1$ for all $e \in U$. Each time, $r(e)$ drops by a factor of $(1+\epsilon)$, we cover element e by $\epsilon b(e)$. Finally, we know that e is inactive and thus, we know that $r_t(e) \leq \delta$.

Hence, each element e is covered to an extent of $\epsilon k b(e)$, where k is the least number, such that

$$(1+\epsilon)^{-k} \leq \delta \qquad (4.53)$$

The lemma follows from this fact. ■

Lemma 4.3.3 $\frac{y_t}{\ln \frac{(1+\epsilon)d}{\delta}}$ *is a dual feasible solution, where* $d = \max_{S \in \mathcal{S}} |S|$

Proof: As in the last section, we define

$$\begin{aligned}
Y_i(S) &= \sum_{e \in S} y_i^1(e) && (4.54)\\
R_i(S) &= \sum_{e \in S} r_i(e) && (4.55)
\end{aligned}$$

Observe, that y_t is a dual feasible solution, if $Y_t(S) \leq c(S)$ for all S.

Note, that initially $R_0(S) = |S|$ and $Y_0(S) = 0$. By the same argument as we used in lemma (4.2.3), we can show that if $R(S)$ decreases by β in iteration i, then the simultaneous increase in $Y(S)$ is $s_i\beta$. Hence, it follows using the area argument of the last section, that

$$Y_t(S) \leq \ln \frac{(1+\epsilon)|S|}{\delta} c(S) \tag{4.56}$$

Replacing $|S|$ by d yields the lemma. ■

Finally, observe that the approximation ratio is bounded by the term

$$\frac{P(x_t)\ln\frac{(1+\epsilon)d}{\delta}}{D(y_t^1)\epsilon\log_{1+\epsilon} 1/\delta} \tag{4.57}$$

which is by corollary (4.3.1) and the fact that $\ln(1+\epsilon)/\epsilon \leq 1$ bounded by

$$\frac{(1+\epsilon)\ln\frac{1+\epsilon}{\delta}}{\ln 1/\delta} \tag{4.58}$$

The last term is equivalent to (4.30) and thus, we obtain the final theorem.

Theorem 4.3.1 *Algorithm (4.5) for the FMC problem is a $1+\omega$ approximation by setting $\delta = ((1+\epsilon)d)^{-1/\epsilon}$ and $\epsilon \leq \sqrt{1+\omega} - 1$.*

4.4 Positive Linear Programming

We now generalize the covering approach, which we described in the last sections, to *positive linear programming* (PLP), which was previously analyzed by Luby and Nisan [14] who gave a parallel approximation algorithm for this problem. In this section we concentrate on the covering variant of PLP. A positive linear program is of the form (4.1) where A, b and c consist of arbitrary positive numbers.

4.4.1 The Algorithm

The algorithm for PLP arises from the ideas of the fractional multi-cover section. To keep this section intuitive, we firstly redefine the set $\mathcal{S}$ as consisting of the column indices of matrix A.

$$\mathcal{S} = \{1, \ldots, n\} \tag{4.59}$$

Furthermore, we redefine U as the set of row indices of matrix A.

$$U = \{1, \ldots, m\} \tag{4.60}$$

Now note, that in the previous sections $e \in S$ iff $A_{eS} = 1$. Thus, the column $(a_{1S}, \ldots, a_{mS})^T$ (for brevity, we will say: column S of A) can be viewed as the incidence vector of set S. Similarly, row $(a_{e1}, \ldots, a_{en})$ (we will say: row e of A) corresponds to element e.

We retain the terminology of the last section, i.e. we associate a variable $x(S)$ with each row S of A and variables $y^1(e), y^2(e)$ and $r(e)$ with each row e of A. We use index i to denote the value of these variables at the beginning of iteration i. As in the last section, we initialize $x_0(S)$ with value 0 for all columns S. Similarly, $y_0^1(e)$ and $y_0^2(e)$ get the value 0 and $r_0(e)$ is set to 1 for all rows e of A.

In an iteration of algorithm (4.5), we pick set S_i, which minimizes the ratio of the cost of the set to the requirement of its active elements. In the PLP setting, we pick column S_i which minimizes

$$\frac{c(S)}{\sum_{r_i(e)>\delta} a_{e,S} r_i(e)} \tag{4.61}$$

We again denote the minimal ratio in iteration i by s_i. Observe, that the FMC algorithm minimizes the same ratio, since A consists of $\{0,1\}$ entries only.

If we pick column S_i to an extent of 1, we satisfy $\frac{a_{eS}}{b(e)}$ percent of the requirement of row e. This can be viewed as covering a_{eS} of the small elements which we defined as replacing e in U' in the last section. The latter similarity suggests the definition of a reservoir $\texttt{res}(e)$ for each row e of A. Denote its value at the beginning of iteration i as $\texttt{res}_i(e)$.

Suppose now, that we pick column S_i in iteration i to an extent of $\epsilon\rho_i$. We then update the reservoirs of active rows e in the following way

$$\texttt{res}_{i+1}(e) = \texttt{res}_i(e) + \rho_i a_{eS_i} \tag{4.62}$$

As in the FMC case, we want to saturate at least one reservoir in each iteration i. Thus, we choose ρ_i in the following way

$$\rho_i = \min_{\substack{a_{eS_i}\neq 0 \\ r_i(e)>\delta}} \left\lceil \frac{b(e) - (\texttt{res}_i(e) \bmod b(e))}{a_{eS_i}} \right\rceil \tag{4.63}$$

We shall assume, that $a_{eS} \leq b(e)$ for all e, S. This assumption guarantees, that in each iteration i, we can find ρ_i, such that

$$\texttt{res}_{i+1}(e) - \texttt{res}_i(e) \leq b(e) \tag{4.64}$$

for all e and hence each reservoir overflows at most once per iteration.

Whenever we saturate a reservoir, i.e. we cover row e by at least $b(e)$, we update $y^1(e)$ and $r(e)$ as follows

$$y_{i+1}^1(e) = y_i^1(e) + s_i r_i(e)\frac{\epsilon}{1+\epsilon} \tag{4.65}$$

$$r_{i+1}(e) = \frac{r_i(e)}{1+\epsilon} \tag{4.66}$$

Whenever $r(e)$ drops below δ, we mark e as inactive. The procedure stops, if no active element remains.

The second dual variable is defined in a continuous way. The reason for this arises from the same argument that we used in the last section. The value of the variable of row e changes independent of the value of $\texttt{res}_i(e)$.

$$y_i^2(e) = y_i^1(e) + s_{i-1} r_i(e) \frac{\epsilon(\texttt{res}_i(e) \bmod b(e))}{(1+\epsilon)b(e)} \tag{4.67}$$

Initialize:	$\forall e \in U : r_0(e) = 1, y_0^1(e) = 0, y_0^2(e) = 0, \texttt{res}_0(e) = 0$
	$\forall S \in \mathcal{S} : x_0(S) = 0$
Iteration i:	**if** $\nexists e \in U : r_i(e) > \delta$ **then stop**
	$s_i = \min_{S \in \mathcal{S}} \frac{c(S)}{\sum_{r_i(e)>\delta} r_i(e)}$
	$\rho_i = \min_{\substack{a_{eS_i} \neq 0 \\ r_i(e) > \delta}} \left\lceil \frac{b(e) - (\texttt{res}_i(e) \bmod b(e))}{a_{eS_i}} \right\rceil$
	$x_{i+1}(S_i) = x_i(S_i) + \rho_i \epsilon$
	$\forall e \in U, r_i(e) > \delta : \texttt{res}_{i+1}(e) = \texttt{res}_i(e) + \rho_i$
	$\forall e \in U, r_i(e) > \delta, \texttt{res}_{i+1}(e) \bmod b(e) = 0 :$
	$r_{i+1}(e) = \frac{r_i(e)}{1+\epsilon}$
	$y_{i+1}^1(e) = y_i(e) + s_i \frac{\epsilon}{1+\epsilon}$
Solution:	$x_{t+1} = x_t$
	$\forall e \in U : y_{t+1}^1 = y_t^1 + s_{t-1} r_t(e) \frac{\epsilon}{1+\epsilon}$
	$\left(\frac{x_{t+1}}{\epsilon \log_{1+\epsilon} 1/\delta}, \frac{y_{t+1}}{\ln(1+\epsilon)^2 d/\delta} \right)$

Figure 4.6: The algorithm for covering positive linear programming.

4.4.2 The Analysis

The analysis part of the PLP algorithm is structured as before. At first, we prove a relationship between primal and dual final objective function values. This part will differ from what we have seen so far. Afterwards, we show, how to scale primal and dual variables at the end in order to obtain primal and dual feasible solutions. Finally, we derive an upper bound for the approximation ratio.

Lemma 4.4.1 *$P_i \leq (1+\epsilon)^2 D_i$ for all iterations i*

Proof: The assumption holds for $i = 0$ since $P_0 = 0 = D_0$. Assume, that we are in iteration i and pick set S_i to an extent of $\epsilon \rho_i$. The change in the primal objective function is

$$P_{i+1} = P_i + \epsilon \rho_i c(S_i) \tag{4.68}$$

Now, we show, how variables $y^2(e)$ of active rows e change.

$$y_{i+1}^2(e) \geq y_i^2(e) + \frac{\epsilon \rho_i a_{eS_i}}{(1+\epsilon)b(e)} s_i r_{i+1}(e) \tag{4.69}$$

The inequality arises from the use of $r_{i+1}(e)$ on the right hand side of the last inequality. We did not use $r_i(e)$, since the reservoir of row e might be almost saturated in iteration i. Thus, the majority of the increase is weighed with the requirement $r_{i+1}(e)$. Thus, it follows that

$$y_{i+1}^2(e) \geq y_i^2(e)\frac{\epsilon\rho_i a_{eS_i}}{(1+\epsilon)^2 b(e)} r_i(e)s_i \tag{4.70}$$

If we sum over all rows e, we obtain the change in the dual objective function value.

$$\begin{aligned} D_{i+1} &\geq D_i + \frac{\epsilon}{(1+\epsilon)^2} s_i\rho_i \sum_{r_i(e)>\delta} a_{eS_i} r_i(e) && (4.71)\\ &= D_i + \frac{\epsilon}{(1+\epsilon)^2}\rho_i c(S_i) && (4.72)\end{aligned}$$

It follows by the induction hypothesis, that

$$\begin{aligned} P_{i+1} &\leq (1+\epsilon)^2\left(D_i + \frac{\epsilon}{(1+\epsilon)^2}\rho_i c(S_i)\right) && (4.73)\\ &\leq (1+\epsilon)^2 D_{i+1} && (4.74)\end{aligned}$$

■

Observe, that $D_i \neq D(y_i^1)$. We have to show a relationship of the final primal and dual solution. Assume, that the PLP algorithm terminates in iteration t. Due to the fact, that we allowed arbitrary positive entries for matrix A and vectors b and c, it might be the case that

$$\texttt{res}_t(e) \bmod b(e) \neq 0 \tag{4.75}$$

for some row e.

We perform an additional step. We divide each of the final requirements $r_t(e)$ by $(1+\epsilon)$ and similarly raise $y(e)$ in the following way

$$\begin{aligned} r_{t+1}(e) &= \frac{r_t(e)}{1+\epsilon} && (4.76)\\ y_{t+1}(e) &= y_t(e) + \frac{\epsilon}{1+\epsilon} s_{t-1} r_t(e) && (4.77)\end{aligned}$$

Furthermore, we define $P_{t+1} = P_t$. It follows, that

$$P_{t+1} \leq (1+\epsilon)^2 D_t \leq (1+\epsilon)^2 D(y_{t+1}^1) \tag{4.78}$$

Corollary 4.4.1 *Finally, $P_{t+1} \leq (1+\epsilon)^2 D(y_{t+1}^1)$.*

The primal scaling works as in the last sections. We transfer lemma (4.3.2) directly and obtain

Lemma 4.4.2 $\frac{x_{t+1}}{\epsilon \log_{1+\epsilon} 1/\delta}$ *is primal feasible.*

The dual scaling has changed slightly due to the definition of the matrix and since we added an additional step after termination of the algorithm.

Lemma 4.4.3 $\frac{y_{t+1}}{\ln \frac{(1+\epsilon)^2 d}{\delta}}$ *is a dual feasible solution, where* $d = \max_{S \in \mathcal{S}} L_S/\alpha_S$. L_S *denotes the sum of the entries of column* S *of* A *and* α_S *is the smallest element of column* S.

Proof: We define as before

$$R_i(S) = \sum_{r_i(e) > \delta} a_{eS} r_i(e) \tag{4.79}$$

$$Y_i(S) = \sum_{e \in U} a_{eS} y_i(e) \tag{4.80}$$

Observe, that y_{t+1} constitutes a dual feasible solution, if

$$Y_{t+1}(S) \leq c(S) \tag{4.81}$$

Note also, that $R_0(S) = L_S$ and $Y_0(S) = 0$. Finally, we have

$$R_0(S) \geq \frac{\delta}{(1+\epsilon)^2} \alpha_S \tag{4.82}$$

Assume, that we are in iteration i and $R(S)$ decreases by β. Suppose, that the decrease is caused by the rows from Δ, i.e.

$$\beta = \sum_{e \in \Delta} a_{eS} \frac{\epsilon}{1+\epsilon} r_i(e) \tag{4.83}$$

Similarly, Y_S increases at the same time by

$$\sum_{e \in \Delta} a_{eS} \frac{\epsilon}{1+\epsilon} r_i(e) s_i = s_i \beta \tag{4.84}$$

Thus, by the same argument as used in the last sections, $Y_{t+1}(S)$ is bounded by the area under the curve $c(S)/x$, where x goes from $\frac{\delta \alpha_S}{(1+\epsilon)^2}$ to L_S. Thus, we obtain that

$$Y_{t+1}(S) \leq c(S) \ln \frac{(1+\epsilon)^2 L_S}{\delta \alpha_S} \tag{4.85}$$

Substituting L_S/α_S by d yields the lemma. ■

We obtain, that the approximation guarantee is bounded by

$$\frac{(1+\epsilon)^2 \ln(1+\epsilon) \ln \frac{(1+\epsilon)^2 d}{\delta}}{\epsilon \ln 1/\delta} \tag{4.86}$$

Applying $\ln(1+\epsilon)/\epsilon \leq 1$ and setting choosing

$$\delta = ((1+\epsilon)^2 d)^{-1/\epsilon} \tag{4.87}$$

shows that the last term is bounded by $(1+\epsilon)^3$. We choose ϵ such that the last term is at most $(1+\omega)$ and obtain the final theorem.

Theorem 4.4.1 *Algorithm (4.6) for the FMC problem is a $1+\omega$ approximation by setting $\delta = ((1+\epsilon)^2 d)^{-1/\epsilon}$ and $\epsilon \leq \sqrt[3]{1+\omega} - 1$.*

At the beginning of this section, we assumed, that $a_{eS} \leq b(e)$ for all e, S. Observe, that the described procedure can also be applied if $a_{eS} > b(e)$ for some e, S. In this case let

$$z = \max\{\max_{eS} a_{eS}, max_e 1/b_e\} \tag{4.88}$$

and define variables

$$x' = zx \tag{4.89}$$

Now, we solve the problem

$$\begin{aligned} \textbf{min} \quad & c'^T x' \\ \textbf{s.t.} \quad & A'x \geq b \end{aligned} \tag{4.90}$$

where A' and c' arise from A and b, respectively, by dividing each entry by z. Observe, that there is a one-to-one correspondence between the latter and the original problem. If we have an optimal solution x' to the latter problem, we obtain an optimal solution to the original problem by dividing each entry of x' by z.

Observe, that the primal entries are now scaled by $z\epsilon k$ instead of only ϵk. Thus, the produced solution has a bad integrality. Actually, there are cases of large entries in A and relatively small entries in b in which no good integrality can be achieved. Suppose, b and c consist of 1-entries only and $A = \texttt{diag}(10^{-6}, \ldots, 10^{-6})$. In this case, the optimal solution is $x_i = 10^{-6}$ for all i. Hence, the integrality depends on the entries of the system matrix that the presented transformation suggests.

4.5 Fractional Covering

We now turn to the general case of covering. This problem was recently treated by Plotkin, Shmoys and Tardos [19] and independently by Young [18]. They consider the following general problem

$$\exists ? x \in P : Ax \geq b \tag{4.91}$$

where $b > 0$ and $Ax \geq 0$ for all $x \in P$. It is assumed to have a fast optimization subroutine that finds $M(y) = \max\{cx, x \in P\}$, where $c = y^T A$. In this section we will give an algorithm which either finds a σ- good[1] solution x, i.e. $Ax \geq (1-\sigma)b$ or claims that no exact solution exists, i.e. there is no $x \in P$ that complies with (4.91). To achieve this goal the following optimization version of the presented problem will be used.

$$\begin{aligned} \textbf{max} \quad & \lambda \\ \textbf{s.t.} \quad & Ax \geq \lambda b \\ & x \in P \end{aligned} \tag{4.92}$$

[1] A solution of this kind is called σ-approximate in [19, 18]. We use σ-good since we defined σ-approximate in a different manner

Observe, that a $(1+\frac{\sigma}{1-\sigma})$-approximate solution (x,λ) to (4.92) is sufficient to either find a σ-good solution or decide that no exact solution to (4.91) exists. Let in the following be (x^*,λ^*) an optimal solution to (4.92), then (x,λ) complies with $\lambda \geq \frac{\lambda^*}{1+\frac{\sigma}{1-\sigma}} = (1-\sigma)\lambda^*$. If $\lambda \geq (1-\sigma)$ then clearly, (x,λ) constitutes a σ-good solution. Otherwise, assume $\lambda < (1-\sigma)$. It follows that $\lambda^* < 1$ and hence no exact solution exists. Such a procedure that either finds a σ-good solution or decides that no exact solution exists is called a *relaxed decision procedure*.

We handle a more general problem, which can be solved using the methods from the last sections. We try to find a maximal point x from the *cone* defined by P, which we will denote by $\mathcal{C}(P)$.

$$\begin{aligned} \textbf{min} \quad & \sum_{j=1}^{P} \nu_j \\ \textbf{s.t.} \quad & Ax \geq b \\ & x \in \mathcal{C}(P) \end{aligned} \tag{4.93}$$

Here we assume, that P is the convex hull of the vertices $\{V_1,\ldots,V_p\}$. In the latter program, ν denotes the vector of parameters, such that $x = \nu_1 V_1 + \cdots \nu_p V_p$. The following LP is equivalent to (4.93):

$$\begin{aligned} \textbf{min} \quad & \sum_{j=1}^{P} \nu_j \\ \textbf{s.t.} \quad & (AV)\nu \geq b \\ & \nu \geq 0 \end{aligned} \tag{4.94}$$

Observe, that we can use the positive linear programming approach from the last section to solve the latter problem. The reason for this fact is, that by definition AV consists of positive entries only (remember that $Ax \geq 0$ for all $x \in P$). We shall later use algorithm (4.6) to obtain a $\left(1+\frac{\sigma}{1-\sigma}\right)$-approximate solution to problem (4.94).

Suppose, we are given an optimal solution ν^* to (4.94). Dividing the obtained variables by $z^* = \sum_j \nu_j^*$ we get a vector of parameters $\nu = \nu^*/z^*$ which in turn specifies a vector $x = \sum_j \nu_j V_j \in P$ (ν denotes a convex combination). Hence, $(1/z^*, x)$ is a solution to (4.92).

Lemma 4.5.1 *If ν^* is an optimal solution to (4.94) of value z^* then $(1/z^*, x)$ is an optimal solution for (4.92), where $x = 1/z \sum_j \nu_j^* V_j$*

Proof: We argued that $(1/z^*, x)$ is feasible and now show that it is also optimal for (4.92). Assume towards a contradiction, that there is a solution $(1/z', x')$, such that $1/z' > 1/z^*$, i.e. $z' < z^*$. Let ν' be the vector of parameters, such that $x' = \sum_j \nu_j' V_j$. Thus, $z'Ax' \geq b$ and hence $(AV)(z'\nu') \geq b$. It follows, that $z'\nu'$ is a solution of (4.94). The value of this solution is $z' \sum_j \nu_j' = z' < z^* = \sum_j \nu_j^*$. Hence ν^* could not have been optimal, contradicting the assumption. ■

Let in the following be ν^* be an optimal solution to (4.94) and let $z^* = \sum_j \nu_j^*$. Let ν be a $(1+\omega)$-approximate solution to the latter problem of value $z = \sum_j \nu_j$. Thus, $z \leq (1+\omega)z^*$.

Define in the following

$$x = 1/z \sum_j \nu_j V_j \tag{4.95}$$

$$x^* = 1/z^* \sum_j \nu_j^* V_j \tag{4.96}$$

We know from the last lemma, that $(1/z^*, x^*)$, is an optimal solution to (4.92). We now claim, that $(1/z, x)$ is a $(1+\omega)$-approximate solution of the same problem. This is an immediate consequence, since from $z \leq (1+\omega)z^*$ it follows that $(1+\omega)\frac{1}{z} \geq \frac{1}{z^*}$.

Theorem 4.5.1 *The developed greedy procedure for covering problems can also be applied to the general setting of a fractional covering problem and constitutes a $(1+\omega)$ approximation algorithm for problem (4.92)*

Thus, with the argument from above, we obtain a relaxed decision procedure for problem (4.91), by setting $\omega = \frac{\sigma}{1-\sigma}$.

4.6 Generalized Covering

In [19, 18] the latter fractional covering problem is generalized to the case of arbitrary vectors b. This version of the problem is called the *generalized covering problem.* A point x from polytope P is called σ-good with respect to a tolerance vector d, when $Ax \geq b - \sigma d$. Plotkin, Shmoys and Tardos show in [19], that this problem can be reduced to fractional covering.

We define

$$\tilde{P} = \{(x, s) : x \in P, s = 1\} \tag{4.97}$$

and reformulate the latter problem as finding a point $(x, s) \in \tilde{P}$, such that

$$Ax + (\rho d - b)s \geq \tilde{b} = (\rho d, \ldots, \rho d)^T \tag{4.98}$$

where $\rho = \max_i \max_{x \in P} \frac{|a_i x - b_i|}{d}$ and a_i denotes the ith row of A. This is a fractional covering problem since $\tilde{b} > 0$ and $Ax + (\rho d - b)s \geq 0$ for all $(x, s) \in \tilde{P}$ as $\rho d \geq |a_i x - b|$ for all rows i.

We get a σ-good solution to the generalized covering problem by computing a σ/d-good solution to the fractional covering problem given by (4.98)

4.7 Performance and Integrality

In this section we will elaborate on the running time of the presented algorithms. Besides that, an important issue will be the *integrality* of an obtained solution to one of the fractional problems. This means, we want to prove a worst case granularity of the solutions.

In the following we will state the running times of our algorithms in terms of used iterations. Within these iterations a certain minimization subroutine has to be called. For example, in the fractional set-cover case, in iteration i, we compute ratio s_i, i.e. we compute the set S_i that minimizes the ratio of the cost of the set to the total requirement of its elements. Hence, the running time in this chapter will be given modulo a factor corresponding to the costs for calling these optimization subroutines.

In the integral set-cover case, it is easy to give examples on which the algorithm uses a linear number of iterations.This means, in each step a single element is covered.

The analysis of the running times for the FSC, FMC, PLP and fractional covering algorithms arises from the fact that in each iteration we lower the requirement of at least one active element or row, respectively. We ensured this fact by defining the extent to which a set or a column is picked such that at least one reservoir gets saturated. Let the size of the universe U be m. Since an element gets inactive after at most $\log_{1+\epsilon} 1/\delta$ decreases in the requirement, we are able to bound the worst case number of iterations needed by

$$m \log_{1+\epsilon} 1/\delta \tag{4.99}$$

The running time of the different algorithms depends on the choice of δ.

In all presented algorithms, we have that the number of used iterations is at most

$$O\left(\frac{m \ln d}{\epsilon \ln(1+\epsilon)}\right) \tag{4.100}$$

The constant d is defined as the cardinality of the largest set in the set-cover and multi-cover case. In the positive linear programming setting d is defined as the maximum ratio L_S/α_S, where L_S denotes the sum of the entries of column S of A and α_S is defined as the smallest entry of this column. Since we use the PLP algorithm to solve fractional and general covering, the running time is the same as in the PLP case, but L_S and α_S are defined with respect to AV or $(AV|\rho d - b)$, respectively.

The integrality of the primal solutions can be deduced in a similar manner. In each of the sections in this chapter, we scaled the primal solution by ϵk, where we proved k to be of value

$$\log_{1+\epsilon} 1/\delta \tag{4.101}$$

Since we pick sets or rows, respectively, to an extent of ϵ, we obtain that the integrality of the solutions for all of the presented algorithms is

$$O\left(\frac{\ln d}{\epsilon \ln(1+\epsilon)}\right) \tag{4.102}$$

where d varies in the different settings.

Note, that the last integrality bound does not apply for the PLP setting in which there are e, S, such that $a_{eS} \geq b(e)$. We showed that in this case we can only guarantee a worst case granularity of

$$O\left(\frac{z \ln d}{\epsilon \ln(1+\epsilon)}\right) \tag{4.103}$$

where z is defined as $\max\{\max_{e,S} a_{eS}, \max_e 1/b(e)\}$.

Observe, that the presented algorithms also construct dual feasible solutions while computing the primal ones. In the section of positive linear programming we mentioned that the integrality of the dual solutions is rather bad. This can be easily seen, since we update the dual variables associated with the rows of the primal programs by adding $sr(i)\epsilon/(1+\epsilon)$, every time we decrease the associated requirement. The latter term involves the fraction $s = c(S)/R(S)$, whose integrality varies throughout the execution of the algorithm. The same argument applies to $r(e)$ which is some fraction between δ and 1.

Chapter 5

Maximum Concurrent Flow

In this chapter we address another version of the multicommodity flow problem, the *maximum concurrent flow* problem (MCF). In this problem, we are given a network $G = (V, E)$ as before and a set of terminal pairs $(s_1, t_1), \ldots, (s_k, t_k)$. The difference to the maximum fractional multicommodity setting is that each terminal pair comes with an associated demand d_j, which specifies the flow that we need to route between s_j and t_j. Now, we want to know the maximum value z, such that at least z percent of each demand can be shipped without violating the capacity constraints. We shall refer to z as the *throughput*.

This problem was introduced by Matula [6]. Subsequently, Matula and Shahrokhi [7] presented a combinatorial approximation algorithm for the *uniform maximum concurrent flow* problem, in which each edge has capacity 1. A faster randomized algorithm for this special case was later given by Klein et al [12]. Leighton et al [20] were the first who considered the capacitated case of the latter problem. They presented a randomized approximation scheme which was later simplified and improved by Goldberg [2]. Radzik [21] recently derandomized the former algorithms.

The mentioned approximation algorithms [7, 12, 20, 21] all use flow rerouting strategies. They start with a flow that routes the demands but is infeasible in terms of the capacity constraints. During the algorithm, parts of the flow is rerouted to satisfy the capacity constraints. The algorithms in [20] and [21] use single commodity minimum cost flow computations and Lagrangian-relaxation techniques to improve the current solution. We shall use a variant of the algorithm presented in the last chapter to solve this problem.

We will also employ single commodity minimum cost flow computations but our approach is inherently different from the above mentioned papers in that no flow is rerouted.

5.1 Problem Specification

We start the formal description of the algorithm with the exact formulation of the primal and dual linear programs which specify these problems. The given graph G comes with a function c on the edges which specifies their capacity. Let in the following P_j denote the set of all possible flows P that ship d_j units of flow from s_j to t_j without violating any capacity constraint. The variable x_P indicates the usage of flow P. $P(e)$ denotes the amount of flow, which is sent along edge e in flow P. As mentioned in the introduction of this chapter we try to maximize the throughput z.

$$\begin{aligned} \textbf{max} \quad & & z \\ \textbf{s.t.} \quad & \forall e \in E: & \sum_{P \ni e} P(e) x_P \leq c(e) \\ & \forall 1 \leq j \leq k: & \sum_{P \in P_j} x_P = z \end{aligned} \tag{5.1}$$

The corresponding dual is an assignment of lengths $l : E \rightarrow \mathbb{R}_+$ to the edges of G and of variables $M(j)$ to the commodities.

$$\begin{aligned} \textbf{min} \quad & & \sum_{e \in E} l(e) c(e) \\ \textbf{s.t.} \quad & \forall 1 \leq j \leq k : \forall P \in P_j : & \sum_{e \in P} P(e) l(e) \geq M(j) \\ & & \sum_{1 \leq j \leq k} M(j) \geq 1 \end{aligned} \tag{5.2}$$

The first constraint of the latter program suggests, that we view $M(j)$ as the single commodity min-cost flow for commodity i with respect to lengths l. Thus, the dual LP can be viewed as finding a length function l, such that

$$\frac{\sum_{e \in E} l(e) c(e)}{\sum_{1 \leq j \leq k} M(j)} \tag{5.3}$$

is minimized.

5.2 The Algorithm

The maximum concurrent flow algorithm proceeds in iterations. In each iteration, we will ship d_j units of flow from source s_j to sink t_j for each of the commodities. We proceed in a round-robin manner, i.e. we consider k steps in each iteration. In step j, we perform the shipment of commodity j.

As in the previous sections, we will construct primal and dual solutions at the same time. We keep a length $l(e)$ with each edge E. During the execution of our procedure, we will update these labels. Therefore, we let $l_{i,j}(e)$ be the length of edge e after having finished the jth step of iteration i. Initially choose $l_{0,0}(e) = \delta / c(e)$. As in the previous chapter, we define

$$D(l) = \sum_{e \in E} l(e) c(e) \tag{5.4}$$

to be the dual objective function value induced by length l.

In step j of iteration i, we compute a single commodity min-cost flow for commodity j with respect to the edge costs $l_{i,j-1}$. We denote the min-cost flow by

$$P_{i,j} = \texttt{mincost-flow}_j(l_{i,j-1}) \tag{5.5}$$

In the following, we will use the terminology from the linear programming definition, i.e. we will denote the flow on edge e by $P_{i,j}(e)$ and associate variable x_P with flow P. We denote the value of variable x_P at the beginning of iteration i by $x_{i,P}$.

After we have routed the flow in step j of iteration i, we update the lengths of the used edges

$$l_{i,j}(e) = \left(1 + \epsilon \frac{P_{ij}(e)}{c(e)}\right) l_{i,j-1}(e) \tag{5.6}$$

Furthermore, let $\alpha(l)$ be defined as

$$\alpha(l) = \sum_{1 \le j \le k} M_j(l) \tag{5.7}$$

where $M_j(l)$ is the value of a min-cost flow for commodity j with respect to the costs l. Observe, that by the linear program and the definition of D and α it follows immediately, that $D(l)/\alpha(l)$ is only larger than the value of a dual optimal solution.

Assume,that we are in iteration i and step j. We route flow along the min-cost flow $P_{i,j}$. Observe, that $D(l_{i,j})$ can be expressed as

$$\begin{aligned} D(l_{i,j}) &= \sum_{e \in E} l_{i,j}(e)c(e) && (5.8)\\ &= \sum_{e \in E} l_{i,j-1}(e)c(e) + \epsilon \sum_{e \in E} \frac{P_{i,j}(e)}{c(e)} l_{i,j-1}(e)c(e) && (5.9)\\ &= D(l_{i,j-1}) + \epsilon \texttt{cost}_{l_{i,j-1}}(P_{i,j}) && (5.10) \end{aligned}$$

For brevity, define $D_i = D(l_{i,0})$. Define

$$\alpha_i = \sum_{1 \le j \le k} \texttt{cost}_{l_{i-1,j-1}}(P_{i-1,j}) \tag{5.11}$$

as the sum of the costs of the min-cost flows of iteration $i-1$. It follows by (5.10) that

$$D_i = D_{i-1} + \epsilon \alpha_i \tag{5.12}$$

We terminate the procedure if $D_t \ge 1$.

Observe, that $\alpha_i \le \alpha(l_{i,0})$ since the lengths of the edges are non decreasing. Therefore, we have

$$\beta_i = \frac{D_i}{\alpha_i} \ge \frac{D(l_{i,0})}{\alpha(l_{i,0})} \tag{5.13}$$

and thus, β_i is larger than the value of an optimal dual solution, that we denote by β.

Initialize:	$\forall e \in E : l_{0,0}(e) = \delta / c(e)$
	$D_0 = m\delta$
	$\alpha_0 = \sum_{1 \le j \le k} \texttt{mincost-flow}_j(l_{0,0})$
Iteration i:	$\beta_i = \frac{D_i}{\alpha_i}$
	if $D_i \ge 1$ **then stop**
	for $1 \le j \le k$ **do**
	$P_{i,j} = \texttt{mincost-flow}_j(l_{i,j-1})$
	$\forall e \in E : l_{i,j}(e) = \left(1 + \epsilon \frac{P_{ij}(e)}{c(e)}\right) l_{i,j-1}(e)$
	$x_{i+1,P_{ij}} = x_{i,P_{ij}} + 1$
	od
	$\alpha_{i+1} = \sum_{1 \le j \le k} \texttt{cost}_{l_{i,j-1}}(P_{i,j})$
	$D_{i+1} = D_i + \epsilon \alpha_{i+1}$
Solution:	$\left(\frac{t-1}{\log_{1+\epsilon} 1/\delta}, \frac{x_{t-1}}{\log_{1+\epsilon} 1/\delta}\right)$

Figure 5.1: The simple, iteration based algorithm for the maximum concurrent flow problem

5.2.1 Analysis

We now show, that the presented procedure is a $(1+\omega)$-approximation for the maximum concurrent flow problem. In this section, we will proceed as in the previous chapters. At first, we derive a closed formula to obtain an upper bound on D_i. We will use this formula to obtain an upper bound on the approximation guarantee. Finally, we prove a worst case congestion of the edges of G and choose δ and ϵ appropriately.

As we have seen in algorithm (5.1), we can write D_i in terms of the min-cost flows we routed in iterations $0, \ldots, i-1$.

$$D_i = \epsilon \sum_{r=1}^{i} \alpha_r + D_0 \tag{5.14}$$

By the definition of β as the value of an optimal dual solution, we know that

$$\alpha_i \le \frac{D_i}{\beta} \tag{5.15}$$

for all iterations i. Thus, we obtain a recursion for D_i.

$$D_i = \frac{\epsilon}{\beta} \sum_{r=1}^{i} D_r + D_0 \tag{5.16}$$

or equivalently

$$D_i = \frac{\epsilon}{\beta - \epsilon} \sum_{r=1}^{i-1} D_r + D_0 \frac{\beta}{\beta - \epsilon} \tag{5.17}$$

Since $D_0 \leq D_0 \frac{\beta}{\beta-\epsilon}$, we can directly apply corollary (2.4.4) and obtain the following closed formula for D_i.

Lemma 5.2.1 *For all i, $D_i \leq \frac{D_0}{1-\epsilon} e^{\frac{\epsilon(i-1)}{(1-\epsilon)\beta}}$*

Proof: By applying the corollary, we get

$$D_i \leq D_0 \frac{\beta}{\beta - \epsilon} e^{\frac{\epsilon(i-1)}{\beta-\epsilon}} \tag{5.18}$$

We assume $\beta \geq 1$ and hence obtain that $\beta - \epsilon \geq \beta(1-\epsilon)$. By the last observation the lemma follows. ■

We now bound the approximation ratio. At termination time of the presented procedure, we know that $D_t \geq 1$ and thus, we have that

$$\frac{D_0}{1-\epsilon} e^{\frac{\epsilon(t-1)}{(1+\epsilon)\beta}} \geq 1 \tag{5.19}$$

and thus, we have

$$\frac{\beta}{t-1} \leq \frac{\epsilon}{(1-\epsilon)\ln\frac{1-\epsilon}{D_0}} \tag{5.20}$$

In the following, we consider the flow routed in the first $t-1$ iterations, i.e. in iterations $0, \ldots, t-2$.

Lemma 5.2.2 *The multicommodity flow instance has a throughput of at least $\frac{t-1}{\log_{1+\epsilon} 1/\delta}$*

Proof: The algorithm terminated in iteration t and thus it must have been the case that $D_{t-1} < 1$. Hence, by the form of D_{t-1} it follows that

$$l_{t-1,0}(e) < \frac{1}{c(e)} \tag{5.21}$$

Initially, we start with $l_{0,0}(e) = \delta / c(e)$ for all edges e. Every time, we route $c(e)$ units of flow along edge e, we increase the length of this edge by at least a factor of $(1+\epsilon)$ and by this it follows, that we route at most $Mc(e)$ units of flow along e, where M is defined as

$$\frac{\delta}{c(e)}(1+\epsilon)^M \leq \frac{1}{c(e)} \tag{5.22}$$

This means, $M \leq \log_{1+\epsilon} 1/\delta$ and since this applies for all edges e, the lemma follows. ■

By the last lemma, we obtain an upper bound on the approximation ratio by transforming (5.20).

$$\frac{\epsilon \log_{1+\epsilon} 1/\delta}{(1-\epsilon)\ln\frac{1-\epsilon}{D_0}} = \frac{\epsilon \ln 1/\delta}{(1-\epsilon)\ln(1+\epsilon)\ln\frac{1-\epsilon}{D_0}} \tag{5.23}$$

We know that $D_0 = m\delta$. Choose now $\delta = \left(\frac{1-\epsilon}{m}\right)^{1/\epsilon}$ and the last term transforms to

$$\frac{1}{(1-\epsilon)\ln(1+\epsilon)(1/\epsilon - 1)} \tag{5.24}$$

We know from the last chapter that $\ln(1+\epsilon) \geq (\epsilon - \epsilon^2/2)$ and thus, we finally obtain the following upper bound on the approximation guarantee.

$$\frac{1}{(1-\epsilon)^2(1-\epsilon/2)} \geq (1-\epsilon)^{-3} \tag{5.25}$$

We now choose ϵ, such that the last term is of value at most $(1+\omega)$ and obtain the final theorem.

Theorem 5.2.1 *The presented algorithm for the maximum concurrent flow problem is a* $(1+\omega)$ *approximation by setting* $\epsilon \leq 1 - \sqrt[3]{\frac{1}{1+\omega}}$ *and choosing* $\delta = \left(\frac{1-\epsilon}{m}\right)^{1/\epsilon}$

5.3 Performance Issues

The running time in terms of the number of iterations of the procedure is t. Now observe, that we were able to bound the performance guarantee of the algorithm by

$$\frac{\beta \log_{1+\epsilon} 1/\delta}{t-1} \tag{5.26}$$

This latter term is of value at least 1 and therefore

$$t \leq \beta \log_{1+\epsilon} 1/\delta + 1 \tag{5.27}$$

This running time is not polynomial in the size of the input, since β might even be exponential. In this section we describe how to use algorithm (5.1) to design another procedure whose running time does not depend on β.

In the following, we prove, that an optimal β has to come from a certain interval $[\lambda, k\lambda]$. We perform a search on β on the given interval. In phase i of the procedure, we are given a lower bound λ_i on β. We either find a $(1+\omega)$-approximate throughput or show, that $\beta \geq 2\lambda_i$.

Let z_j be the maximal achievable throughput of commodity j, i.e.

$$z_j = \frac{\texttt{max-flow}_j(c)}{d_j} \tag{5.28}$$

and define

$$z = \min_{1 \leq j \leq k} z_j \tag{5.29}$$

Lemma 5.3.1 *An optimal dual solution value* β *has to be within the interval* $[z/k, z]$

Initialize: $\forall 1 \le j \le k : z_j = \frac{\texttt{max-flow}_j(c)}{d_j}$
$z = \min_{1\le j\le k} z_j$
$\lambda_0 = z/k$
Phase i: $\lambda'_i = \lambda_i / \lfloor \lambda_i \rfloor$
$\forall 1 \le j \le k : d_{i,j} = \lfloor \lambda_i \rfloor d_j$
$z^*_i = \texttt{algorithm1}(d_i, 2\lambda'_i \log_{1+\epsilon} 1/\delta)$
if `!term` **then**
  $\lambda_{i+1} = 2\lambda_i$
else
  $z^* = z^*_i \lfloor \lambda_i \rfloor$
  stop
fi

Figure 5.2: The β search variant of the maximum concurrent flow algorithm.

Proof: Since z_i is percentage of the demand d_i, which could be routed obeying the capacity constraints $c(e)$, we can clearly route $\frac{z_i}{k} d_i$ units using only a fraction $1/k$ of the edge capacities and thus the total throughput is at least z/k. Furthermore, the overall throughput is limited by the maximum single commodity throughput of a commodity, i.e. by z. ■

Assume, that we are given an instance of the maximum concurrent flow problem

$$(G_0, \{(s_j, t_j)\}_{1\le j\le k}, c_0, d_0) \tag{5.30}$$

Suppose, that we compute a range for the optimum throughput for this instance, using the last lemma.

$$z/k \le \beta_0 \le z \tag{5.31}$$

To keep the description of the algorithm simple, we would like to have a lower bound on the optimum throughput of value at least 1. Thus, in the case of $z/k < 1$, we define a new instance

$$(G, \{(s_j, t_j)\}_{1\le j\le k}, c, d) \tag{5.32}$$

where $G = G_0, d = d_0$ and c is defined as

$$\forall e \in E : c(e) = \lceil k/z \rceil c_0(e) \tag{5.33}$$

Let β be the maximum throughput of the latter instance Observe, that we have $\beta = \lceil \frac{k}{z} \rceil \beta_0$ and thus

$$1 \le \left\lceil \frac{k}{z} \right\rceil \frac{z}{k} \le \beta \le z \left\lceil \frac{k}{z} \right\rceil \tag{5.34}$$

In the following and in algorithm (5.2), we assume, that $z/k \ge 1$. Otherwise, the last modification has to be applied first. Afterwards, algorithm (5.2) can be applied to obtain a solution

z^* for the modified problem. Finally, we get a solution to the original problem by dividing z^* by $\lceil \frac{k}{z} \rceil$.

Assume, that we are in phase i. We are given λ_i, such that

$$1 \leq \lambda_i \leq \beta \tag{5.35}$$

We now multiply each given demand d_j by $\lfloor \lambda_i \rfloor$ and obtain

$$d_{i,j} = \lfloor \lambda_i \rfloor d_j \tag{5.36}$$

By this modification, we obtain a new problem instance, with optimal dual solution $\beta_i' = \frac{\beta}{\lfloor \lambda_i \rfloor}$. In other words, we know that

$$\beta_i' \geq \lambda_i' = \frac{\lambda_i}{\lfloor \lambda_i \rfloor} \tag{5.37}$$

Observe, that λ_i' is of value at least 1 and at most 2. We now perform $2\lambda_i' \log_{1+\epsilon} 1/\delta$ iterations of procedure (5.1) on this new instance. We denote this call by the following keyword

$$\texttt{algorithm1}(d_i, 2\lambda_i' \log_{1+\epsilon} 1/\delta) \tag{5.38}$$

If the procedure terminates (variable `term` is set to the value `true`) within the given number of steps, we obtain a $(1+\omega)$-approximate throughput z_i^* to the adjusted problem. If it does not terminate, we know, that the optimal throughput for this problem has to be of value at least $2\lambda_i'$, i.e.

$$\beta_i' \geq 2\lambda_i' = 2\frac{\lambda_i}{\lfloor \lambda_i \rfloor} \tag{5.39}$$

and hence, $\beta \geq 2\lambda_i$.

Now suppose, that algorithm (5.2) terminates in phase F, i.e. we get a $(1+\omega)$-approximate solution z_F^* to the modified problem.

$$(1+\omega) z_F^* \geq \beta_F' \tag{5.40}$$

Note, that the solution we get constitutes a solution to the original problem. Since we raised the demands in the modified problem by the factor $\lfloor \lambda_F \rfloor$, the throughput of the obtained solution in the original setting would be $\lfloor \lambda_F \rfloor z_F^*$. Hence, by using inequality (5.40) we obtain

$$(1+\omega) z^* \geq \lfloor \lambda_F \rfloor \beta_i' = \beta \tag{5.41}$$

and thus, we obtain a $(1+\omega)$-approximate solution to the original problem, by multiplying z_F^* by $\lfloor \lambda_F \rfloor$.

The following lemma bounds the number of phases, that we need to obtain z^*.

Lemma 5.3.2 *Algorithm (5.2) needs at most* $(\log k)$ *phases to compute a* $(1+\omega)$*-approximate solution to the given instance of a maximum concurrent flow problem*

Proof: We start with $\lambda_0 \geq z/k$ and finally, we have $\lambda_F \leq z$. In each phase i, we double the value of λ_i to construct λ_{i+1} and hence

$$\frac{z}{k}2^F \leq z \tag{5.42}$$

This yields the assumption. ■

By the last lemma we obtain a bound on the overall running time of the last algorithm.

Lemma 5.3.3 *Algorithm (5.2) needs* $O\left(\frac{\log k}{\epsilon \log(1+\epsilon)} \log \frac{m}{1-\epsilon}\right)$ *iterations to compute a* $(1+\omega)$ *approximate solution to the given instance of the maximum concurrent flow problem*

Proof: In phase i of the last procedure, we perform $2\lambda_i' \log_{1+\epsilon} 1/\delta$ iterations of algorithm (5.1). We argued that $\lambda_i' \leq 2$ and thus, by applying lemma (5.3.2), we obtain an overall number of needed iterations of

$$O(\log k \log_{1+\epsilon} 1/\delta) \tag{5.43}$$

The assumption follows by substituting δ by $\left(\frac{1-\epsilon}{m}\right)^{1/\epsilon}$. ■

We can even further improve the running time. Suppose, we want to obtain a $(1+\omega)$ approximation on the maximum throughput of a distinct problem. Instead of applying algorithm (5.2) directly, we split the computation into 2 parts. In the first part, we compute a 2-approximation. This computation stays the same for every input and thus, ϵ can be viewed as a constant. By this observation the running time of the first part reduces to

$$O(\log k \log m) \tag{5.44}$$

We denote a call to algorithm (5.2) by

$$\texttt{algorithm2}((G, \{(s_i, t_i)\}_{1 \leq i \leq k}, d, c), \omega) \tag{5.45}$$

This returns a $(1+\omega)$-approximate throughput for the given problem instance.

By running the first part, we obtain a 2-approximate throughput z_1. In the case of $\omega \geq 1$, we are done at this point. Otherwise note, that algorithm (5.2) initially computes a range for an optimal dual solution value β. After having finished part 1 we are in a better position. We obtained a 2-approximate throughput z_1 and therefore know, that

$$z \leq \beta \leq 2z \tag{5.46}$$

Thus, we proceed as in a single phase of algorithm (5.2). We multiply each of the original demands d_j by $\lfloor z \rfloor$ and call algorithm (5.1). Thus, the second phase uses only

$$\left(\frac{2}{\epsilon \log(1+\epsilon)} \log \frac{m}{1-\epsilon}\right) \tag{5.47}$$

iterations of the first algorithm. Since we know that $\omega \leq 1$, we can lower bound $1-\epsilon$ by a constant and thus obtain the following running time of the second part of algorithm (5.3).

$$O\left(\frac{1}{\epsilon \log(1+\epsilon)} \log m\right) \tag{5.48}$$

Putting the running times together and noting that each iteration of algorithm (5.1) consists of k min-cost flow computations yields the final theorem.

Theorem 5.3.1 *The algorithm (5.3) needs $O(k \log m(\log k + (\epsilon \log(1+\epsilon))^{-1}))$ single commodity min-cost flow computations to obtain a $(1+\omega)$ approximate solution to the maximum concurrent flow problem*

Input:	$(G, \{(s_j, t_j)\}_{1 \le j \le k}, d, c, \omega)$
Part 1:	$z_1 = \texttt{algorithm2}((G, \{(s_j, t_j)\}_{1 \le j \le k}, d, c), 1)$
Part 2:	**if** $\omega < 1$ **then**
	$\forall 1 \le j \le k : d'_j = \lfloor z_1 \rfloor d_j$
	$z'_1 = z_1 / \lfloor z_1 \rfloor$
	$z_2 = \texttt{algorithm1}(d', 2z'_1 \log_{1+\epsilon} 1/\delta)$
	$z^* = z_2 \cdot \lfloor z_1 \rfloor$
	else
	$z^* = z_1$
	fi

Figure 5.3: Subdividing the computation of a $(1+\omega)$ approximate solution into two parts results in a better running time.

5.4 Integrality

As in the last chapters, we are again interested in the integrality of the produced solution for the MCF problem. Observe, that the notion of integrality is only reasonable, in the case of integral input, i.e. the assumption we made in the last sections, that we use integral demands and capacities is justified in that sense.

In the presented procedure, we route flow along single commodity min-cost flows. In the case of integral capacities and demands one can decompose a solution to the latter problem into integral flow paths. In the simple algorithm (5.1) we leave the input untouched, i.e. in each of the steps we work with the integral input and thus route along flow paths of integral value only. In the second algorithm (5.2) we solve auxiliary problems, that arise from the original instance by multiplying the demands or the capacities respectively by an integral amount. Hence, we again work on integral input and thus, the min-cost-flow computations give us flow paths of integral value only.

After our procedures terminate, we divide the variables x_P of the various flows by $\log_{1+\epsilon} 1/\delta$. This scaling determines the integrality of the solution.

Theorem 5.4.1 *In the case of integral demands and edge capacities the presented algorithms find a solution of integrality $O(\log_{1+\epsilon} 1/\delta) = O\left((\epsilon \log 1+\epsilon)^{-1} \log \frac{m}{1-\epsilon}\right)$. For $\omega \le 1$, we obtain the integrality $O\left((\epsilon \log 1+\epsilon)^{-1} \log m\right)$.*

5.5 Comments on the Analysis

Observe, that the presented procedures differ from the packing algorithms in that we allow arbitrary integral edge capacities but we only use a continuous length function. There is no notion of a reservoir here. We decided to use only one length because the running time analysis of the MCF algorithm does not depend on saturating a reservoir in each step. This was the case in the arbitrary capacity multicommodity setting.

The algorithms of the packing chapter have the advantage of computing shortest paths with respect to *nice* lengths, i.e. the length of each edge e has the form $\delta(1+\epsilon)^u$, for some u. This might be an advantage, when implementing the algorithm on the RAM. However, we can present the maximum concurrent flow algorithms in the same discrete manner. The analysis would be similar to the PLP-analysis of the last chapter. We incur a slightly worse bound on ϵ. Thus, a modification of the given techniques to the discrete model is straightforward and involves only known techniques from previous chapters.

Chapter 6

Conclusion and Open Problems

In this thesis, we presented combinatorial algorithms for the class of covering and packing problems. We have shown that our algorithms are capable of solving the same general problems as previously known combinatorial procedures (c.f. [19, 18]). In fact we showed an improvement in the running time. In the covering case we proved, that our general procedures have polynomial running time while the procedures for packing problems have strongly polynomial running time. The previously known algorithm have a running time, dependent on the width of the given problem instance.

The presented maximum concurrent flow algorithm matches the best known running time results from [21]. In the concurrent flow section we have seen how powerful and flexible the developed ideas are. By a slight modification of the packing algorithm we obtained a fast and simple procedure for this setting.

Besides that, we also considered the question of solution granularity. We proved worst case bounds which are again polynomial in the input size for the class of covering problems and which turned out to be strongly polynomial for the packing case. The latter result also extended to the concurrent flow setting.

We have shown, that the natural way of solving the mentioned problems leads to intuitive and short proofs. In the covering chapter the presented methods arise from the ideas of the set-cover greedy algorithm. In the packing chapter we show how to use shortest path and min-cost flow computations to obtain simple algorithms.

However, we do not know how the presented procedures perform in practice. However, we are convinced that fast implementations are possible.

We were not able to show a dependence of the running time of the packing procedures on $O(\log k)$ instead of $O(\log m)$. Moreover, it is quite likely, that the presented techniques allow the design of strongly polynomial algorithms for the covering case. We also expect, that the presented techniques extend to fast min-cost multicommodity flow algorithms.

Bibliography

[1] R.K. Ahuja, T.L. Magnati, and J.B. Orlin. *Network Flows*. Prentice Hall, 1993.

[2] A.V.Goldberg. A natural randomization strategy for multicommodity flow and related algorithms. Technical Report STAN-CS-91-1372, Stanford University, 1991.

[3] V. Chvátal. A greedy heuristic for the set-covering problem. *Math. of Oper. Res.*, 4(3):233–235, 1979.

[4] C.Papadimitriou and K. Steiglitz. *Combinatorial Optimization*. Prentice-Hall, 1982.

[5] D.Hochbaum, editor. *Approximation Algorithms for NP-hard Problems*. PWS Publishing Company, 1997.

[6] D.W.Matula. *Graph Theory and its Applications to Algorithms and Computer Science*. Wiley, New York, 1985. Concurrent Flow and Concurrent Connectivity in Graphs.

[7] D.W.Matula and F.Shahrokhi. The maximum concurrent flow problem. *J. of the ACM*, 37(2):318–334, 1990.

[8] R.L. Graham, D.E. Knuth, and O. Patashnik. *Concrete Mathematics*. Addision Wesley, 1988.

[9] G.Ziegler. *Lectures on Polytopes*. Springer, 1994.

[10] J.Kleinberg and E.Tardos. Approximations for the disjoint paths problem in high diameter planar networks. In *27th Annual ACM Symposium on Theory of Computing*, pages 26–35, 1995.

[11] D.S. Johnson. Approximation algorithms for combinatorial problems. *J. of Computer and System Sciences*, 9:256–278, 1974.

[12] P. Klein, S.Plotkin, C.Stein, and E.Tardos. Faster approximation algorithms for the unit capacity concurrent flow problem with applications to routing and finding sparse cuts. Technical Report 961, School of Operations Research, Cornell, 1990.

[13] L.Lovász. On the ratio of optimal integral and fractional covers. *J. of Discrete Mathematics*, 13:383–390, 1975.

[14] M.Luby and N.Nisan. A parallel approximation algorithm for positive linear programming. In *25th Annual ACM Symposium on the Theory of Computing*, pages 448–457, 1993.

[15] N.Garg. Approximating sparsest cut. Technical report, Max-Planck-Institute for Computer Science, 1995.

[16] N.Garg. Lecture notes on approximation algorithms. 1995.

[17] N.Garg, V.Vazirani, and N.Yannakakis. Approximate max-flow min-(multi)cut theorems and their applications. *Siam J. of Computing*, 25(2):235–251, 1996.

[18] N.Young. Randomized rounding without solving the linear program. In *6th Annual Symposium on Discrete Algorithms*, 170-178, 1995.

[19] S.Plotkin, D.Shmoys, and E.Tardos. Fast approximation algorithms for fractional packing and covering. *Methods of Operations Research*, 20:257–301, 1995.

[20] T.Leighton, F.Makedon, S.Plotkin, C.Stein, E.Tardos, and S.Tragoudas. Fast approximation algorithms for multicommodity flow problems. In *25th Annual ACM Symposium on the Theory of Computing*, pages 101–111, 1991.

[21] T.Radzik. Fast deterministic approximation for the multicommodity flow problem. In *6th Annual Symposium on Discrete Algorithms*, pages 486–492, 1995.

[22] P.M. Vaidya. A new algorithm for minimizing convex functions over convex sets. In *30th Annual IEEE Symposium on Foundations of Computer Science*, pages 332–337, 1989.