Lagrangian Relaxation Based Algorithms for Convex Programming Problems

by
Rohit Khandekar
Department of Computer Science and Engineering

Submitted
in fulfillment of the requirements of the degree of
Doctor of Philosophy

to the

Indian Institute of Technology Delhi
March 2004

Certificate

This is to certify that the thesis titled "Lagrangian Relaxation Based Algorithms for Convex Programming Problems" being submitted by Rohit Khandekar to the Indian Institute of Technology Delhi, for the award of the degree of Doctor of Philosophy, is a record of bona-fide research work carried out by him under my supervision. In my opinion, the thesis has reached the standards fulfilling the requirements of the regulations relating to the degree.

The results contained in this thesis have not been submitted to any other university or institute for the award of any degree or diploma.

Naveen Garg
Associate Professor
Department of Computer Science and Engineering
Indian Institute of Technology Delhi
New Delhi 110 016

To my parents
for their unwavering love and support

Acknowledgments

A million thanks to Naveen Garg, my supervisor, for his invaluable guidance. Naveen influenced me right from my undergraduate days when he taught a course on linear programming and approximation algorithms. His attention to detail, highly intuitive approach to concepts, and ability to explain ideas in a very lucid manner have set an example I hope to match some day.

I am very grateful to Infosys Technologies Limited, Bangalore for funding my doctoral research. I would like to thank R. Ravi and Kurt Mehlhorn for hosting my research visits to CMU and Max-Planck-Institut. These visits gave me a nice exposure and opportunities to interact with other researchers working in the field.

Special thanks to Vinayaka Pandit for giving me a great company while doing research. Thanks to Anup, Basant, Parag, and Subhajit for those numerous sweets and pizzas and for providing a lively work environment in the department. Thanks to Kapil and Bharat for the great times that we spent together.

I am sincerely grateful to my extended family. Thanks, Aradhana, for your help, care, and understanding throughout the development and writing of this thesis. Many thanks to my in-laws for their great support. A great debt of gratitude to my parents, grand mother, and sister Bhagyashree whose love and encouragement dates back to times when I hadn't even dreamt about writing a thesis.

October 2004 Rohit Khandekar

Abstract

This thesis deals with a class of Lagrangian relaxation based algorithms developed in the computer science community in last couple of decades. We present a unified framework for designing such algorithms for a large family of convex programming problems. Our algorithms are based on exponential potential functions and given any $\epsilon \in (0, 1)$, compute $(1 + \epsilon)$-approximate solutions in number of iterations proportional to ϵ^{-2}.

We present an algorithm for the *general resource sharing problem* which may have both positive and negative entries in the constraints. We improve and generalize the results of Plotkin-Shmoys-Tardos and Young to convex and concave functions. Using this, we obtain faster algorithms for computing fractional solution to the Steiner forest problem and Held-Karp bounds for TSP. We formulate a new problem called the *provisioning problem* and obtain an approximation scheme for it. Using this framework, we obtain width-independent running times for solving the *generalized hitting set problems* fractionally.

We extend this framework to the mixed packing and covering problems with convex and concave functions over arbitrary convex sets, thereby solving an open problem of Young. We also obtain approximation schemes for computing primal and dual solutions for min-max and max-min resource sharing problems. Our algorithms use "variable-size increments" and have width-independent running times.

We formulate a new problem called the *substitute mixed problem*. We show that the "round-robin increments" can be used to design efficient algorithms for this problem. We also present algorithms for the special cases of the substitute packing and covering problems. Using this, we design faster approximation algorithms for computing the size of the maximum cuts in weakly bipartite graphs and the maximum independent sets in t-perfect graphs. We also show that the substitute mixed problem generalizes the block-angular min-max and max-min resource sharing problems and obtain better running times than those given by Grigoriadis-Khachiyan.

Our framework can also be used to solve the linear versions of the above abstract problems including packing, covering, mixed packing and covering. Under some conditions, we extend our techniques to linear programs which may have negative entries. Using these ideas, we improve the recent results of Fleischer for covering with upper bounds on the variables.

Contents

List of Figures

List of Algorithms

Chapter 1

Introduction

This thesis deals with designing efficient approximation algorithms for various classes of *linear* and *convex programming problems*. Convex programming — minimizing a convex function over a convex set — is one of the central problems in mathematical programming. Convex programming with its special cases like linear programming and semi-definite programming arises in numerous applications.

Since the 1970s, there has been an extensive research in convex optimization as well as in studies on the computational complexity of convex programming problems. The *ellipsoid method* was developed out of the work of Shor [76, 75, 77] and Yudin & Nemirovskii [86, 87]. In a path-breaking work, Khachiyan [53] adapted this method to prove the polynomial-time solvability of linear programming, in which the objective is to minimize a linear function of several variables over a polytope. His paper, in turn, stimulated a flood of research aimed at accelerating this method and making it more stable for numerical purposes. The applicability of the ellipsoid method to combinatorial optimization was discovered independently by Karp & Papadimitriou [52], Padberg & Rao [68], and Grötschel et al. [36].

For linear programming, however, long before the advent of the ellipsoid method, the *simplex method* was known and used extensively in practice. The simplex algorithm was discovered by Dantzig [15] and was shown to be very efficient in practice. However, in the worst case, it can be shown to have an exponential running time [54]. Another break-through was achieved by Karmarkar [50] who gave a polynomial time algorithm for linear programming which was fast in both theory and practice. Karmarkar's algorithm requires — like simplex method — the complete knowledge of the constraint system for the linear programming problem.

The methods mentioned above are general and often cannot give running times comparable with special-purpose algorithms. The main focus in this thesis is to present a framework that enables us to exploit the specific combinatorial structure of the problems as much as possible to provide near-optimum solutions very quickly. Here we would like to *trade off accuracy for speed*, explicitly and rigorously. One motivating factor is that in many practical applications, quickly obtaining a partially accurate solution is much preferable to obtaining a very accurate solution after a lot of computation. A secondary (and independent) consideration is that the input data in many practical applications has limited accuracy to begin with.

1.1 Lagrangian relaxation

Lagrangian relaxation was one of the first methods proposed for solving linear programs. As early as the 1950's, John von Neumann proposed and analyzed an $O(m^2 n \log(mn)/\epsilon^2)$-time Lagrangian-relaxation algorithm for solving two-person zero-sum matrix games [59]. The algorithm returned a solution with additive error ϵ assuming the matrix was scaled to lie between 0 and 1. In 1950, Brown & von Neumann also proposed a system of differential equations that converged to an optimal solution, with the suggestion that the equations could form the basis of an algorithm [11].

Our algorithms are based on Lagrangian relaxation. Here we describe the basic idea behind this technique. Suppose that we are interested in a convex optimization problem of the form:

$$\text{minimize } f(x) \text{ over } x \in P, \tag{1.1}$$

where $P \subseteq \mathbb{R}^n$ is a *convex* set in the n-dimensional Euclidean space and f is a convex function defined on P. A set P is said to be convex if for any two vectors $u, v \in P$, any *convex combination* $\lambda_1 u + \lambda_2 v$ (where $\lambda_1, \lambda_2 \geq 0$ and $\lambda_1 + \lambda_2 = 1$) also lies in P. For instance, any polytope which is described as an intersection of finitely many half spaces is a convex set. We assume some "nice" properties about the function f, i.e., f is *convex* and continuous. A function f is said to be convex if for any u, v in its convex domain, we have $f(\lambda_1 u + \lambda_2 v) \leq \lambda_1 f(u) + \lambda_2 f(v)$ for any $\lambda_1, \lambda_2 \geq 0$ such that $\lambda_1 + \lambda_2 = 1$. A general objective in this thesis is to obtain $x \in P$ such that $f(x) \leq \mathsf{OPT} + \epsilon$ where OPT is the optimum value of the instance and ϵ is any given error-parameter. We only deal with problems in which OPT is finite. Sometimes we are able to find $x \in P$ such that $f(x) \leq (1+\epsilon) \cdot \mathsf{OPT}$, yielding a multiplicative approximation guarantee.

Now assume that the convex set P is of the form $P = P' \cap \{x \in \mathbb{R}^n \mid h_i(x) \leq 0, 1 \leq i \leq m\}$ where $P' \subseteq \mathbb{R}^n$ is a convex set and h_i are convex functions. Suppose that P' is "easy" in terms of minimizing f over it while the constraints $h_1, \ldots, h_m$ are "hard" to satisfy. The idea behind Lagrangian relaxation is to "relax" the hard constraints and solve the problem after incorporating only the easy constraints. However we add a penalty in the objective function for violating the "hard" constraints. The relaxed problem now has the form:

$$\begin{array}{lll} \text{minimize} & f(x) + \sum_{i=1}^m y_i h_i(x) & \\ \text{s.t.} & & \\ & x \in P' & \\ & y_i \geq 0 & i = 1, \ldots, m \end{array} \tag{1.2}$$

We have associated non-negative "Lagrangian multipliers" y_i with the functions $h_i(x)$. Note that for fixed values of $y_i \geq 0$, the objective function $f(x) + \sum_{i=1}^m y_i h_i(x)$ is convex. If $h_i(x) > 0$, then the term $y_i h_i(x)$ makes a positive contribution to the objective function. Thus it acts as a penalty, since this is a minimization problem. The idea is to capture as much combinatorial structure of the problem as possible in the description of P' and using this structure, design a fast subroutine to minimize a convex function over P'.

Note that for any fixed values of $\{y_i\}$, the optimum value of the relaxed program (1.2) is a lower bound on OPT, where OPT is the optimum value of the original program (1.1). This is true since any feasible solution $x \in P$ for the original program (1.1) is feasible for the relaxed program (1.2) and objective value of this solution in the relaxed program (1.2) is at most the objective value in the original program (1.1). In order to get the best lower bound, we find the values of y_i for which this lower bound is maximized.

$$\max_{y_1,\ldots,y_m \geq 0} \min_{x \in P'} \left(f(x) + \sum_{i=1}^m y_i h_i(x) \right).$$

This program is referred to as the *Lagrangian dual* of the original program. From the above argument it is clear that the optimum value of this dual is at most OPT. Note that for any $\tilde{x} \in P$ and for any $y_i \geq 0$ for $i = 1, \ldots, m$ we have

$$\min_{x \in P'} \left(f(x) + \sum_{i=1}^m y_i h_i(x) \right) \leq \mathsf{OPT} \leq f(x).$$

The difference in the optimum values of the program (1.1) and the dual program is called the *duality gap*. All problems that we deal with have duality gap zero. In this case, one way to show the near-optimality of $\tilde{x} \in P$ for the program (1.1) is to find y_i for $i = 1 \ldots, m$ such that

$$f(\tilde{x}) \leq \epsilon + \min_{x \in P'} \left(f(x) + \sum_{i=1}^{m} y_i h_i(x) \right).$$

This in fact also proves the near-optimality of $\{y_i\}$ for the dual program. This is how we prove the near-optimality of the solutions that our algorithms generate.

1.1.1 Lagrangian multipliers as exponential weights

Almost all the problems that we consider have the form

$$\min_{x \in P} \max_{1 \leq i \leq m} f_i(x),$$

where $f_i : P \to \mathbb{R}$ are m convex functions defined on a non-empty convex set P. This problem is a special case of the problem (1.1) since the function $f(x) = \max_{1 \leq i \leq m} f_i(x)$ that we are trying to minimize is convex. The above problem can be formulated as

$$\begin{array}{lrcll} \text{min} & \lambda & & & \\ \text{s.t.} & f_i(x) - \lambda & \leq & 0 & \text{for } 1 \leq i \leq m \\ & x & \in & P & \end{array}$$

The Lagrangian dual of the above formulation is

$$\max_{y_1, \ldots, y_m \geq 0} \min_{\lambda \in \mathbb{R}, x \in P} \left(\lambda + \sum_{i=1}^{m} y_i (f_i(x) - \lambda) \right).$$

The coefficient of λ in the above expression is $1 - \sum_i y_i$. If $\sum_i y_i \neq 1$, then by choosing λ appropriately (either very large positive or very large negative number), the above expression can be made arbitrary small. Since we are trying to maximize the above expression over the choices of $\{y_i\}$, we can safely assume that $\sum_i y_i = 1$. Thus the dual problem takes the form

$$\max_{\substack{y_i \geq 0 \\ \sum_i y_i = 1}} \min_{x \in P} \sum_{i=1}^{m} y_i f_i(x).$$

Instead of minimizing $\max_i f_i(x)$, we minimize a "smooth" approximation $\ln \sum_i \exp(f_i(x))$. Here $\ln(z) = \log_e z$ denotes the natural logarithm and $\exp(z) = e^z$ denotes the exponential function. For any real numbers $a = (a_1, \ldots, a_m)$, define $\text{lmax } a = \ln \sum_i \exp(a_i)$. Observe that $\text{lmax } a$ approximates $\max_i a_i$ well since

$$\max_i a_i \leq \text{lmax } a \leq \max_i a_i + \ln m.$$

It is easy to work with lmax since it is a differentiable function of a_i. The partial derivative of $\text{lmax } a$ with respect to a_i is

$$\frac{\partial \text{ lmax } a}{\partial a_i} = \frac{\exp(a_i)}{\sum_j \exp(a_j)}.$$

Our algorithms compute a sequence of iterates $\{x_n\}$ that converges to a near-optimum solution. If $x_n \in P$ is the current iterate, the direction of the *steepest descent* of the function $\mathrm{lmax}\ f(x)$ is given by these partial derivatives at the point x_n. We then compute $x \in P$ that minimizes

$$\sum_{i=1}^{m} \frac{\exp(f_i(x_n))}{\sum_j \exp(f_j(x_n))} \cdot f_i(x)$$

and let our next iterate be $x_{n+1} = (1-\tau)x_n + \tau x$ for an appropriate step-size $\tau \in (0, 1]$.

Thus in the algorithm we let the Lagrangian multipliers to be *normalized exponential weights*

$$y_i = \frac{\exp(f_i(x))}{\sum_j \exp(f_j(x))}.$$

We repeatedly compute $x \in P$ that minimizes $\sum_i y_i f_i(x)$ and update the current solution and the Lagrangian multipliers. We prove that after making enough number of such iterations, the difference between the primal objective function and the dual objective function can be made arbitrarily small. Thus as a by-product, we also compute a near-optimum dual solution and prove that the duality gap of the convex program in question is zero. We refer to Bienstock [7] for more discussion on the exponential potential function methods for approximately solving linear programming problems.

1.2 Lagrangian relaxation based algorithms: previous work

Lagrangian relaxation has been used widely in designing approximation algorithms for linear programming problems. In early days, the algorithms for solving linear programs approximately were mainly motivated by multicommodity problems. As early as 1973, Fratta et al. [24] developed an iterative approximation algorithm for various routing problems in telecommunications. The so-called "flow deviation method" introduced in this paper forms a basis of many multicommodity flow algorithms that followed. Although, they do not prove any theoretical guarantees of their algorithm, Bienstock [7] recently pointed out that their algorithm is, in fact, provably good. In the late 1980s, Shahrokhi & Matula [74] presented the first combinatorial fully polynomial time approximation scheme (FPTAS) for a special case of the maximum concurrent flow problem that has arbitrary demands and uniform capacities. Their algorithm introduces the use of exponential length function to model the flow congestion on an edge and computes a $(1+\epsilon)$ approximation in time $O^*(nm^7\epsilon^{-5})$. This work caught the attention of the computer science community, largely due to applications of multicommodity flow to balanced separators [60]. Klein et al. [55] reduced the running time of this algorithm by using randomization. Leighton et al. [61], in turn, extended [55] to the case of arbitrary capacities and gave better running times when the capacities are uniform.

In a seminal paper, Plotkin et al. [69, 70] presented a general framework for linear packing and covering problems. Their algorithms use exponential potential functions to penalize infeasibilities and find approximately feasible solutions to systems of inequalities. They considered an abstract packing (resp. covering) problem of the form: find a vector $x \in \mathbb{R}^n$ satisfying $Ax \leq b$ (resp. $Ax \geq b$) and $x \in P$ where $A \in \mathbb{R}^{m\times n}, b \in \mathbb{R}^m_+$ and $P \subseteq \mathbb{R}^n$ is a convex set over which it is easy to minimize (resp. maximize) a linear function. Assuming that $Ax \geq 0$ for all $x \in P$, they presented an "ϵ-relaxed" decision procedure that given an error parameter $\epsilon \in (0,1)$, either computes an $x \in P$ such that $Ax \leq (1+\epsilon)b$ (resp. $Ax \geq (1-\epsilon)b$) or concludes that there is no $x \in P$ that satisfies $Ax \leq b$ (resp. $Ax \geq b$). The running time of their algorithm depends on the *width* of P relative to $Ax \leq b$, which is defined by $\rho = \max_{1\leq i\leq m} \max_{x\in P} |A_i x/b_i|$ where A_i denotes the ith row of A. The worst-case number of iterations in their algorithm depends polynomially on m and is also proportional to $\rho\epsilon^{-2}$. If Ax has negative entries for $x \in P$, their algorithm takes time proportional to $\rho^2\epsilon^{-2}$. However, for this algorithm to work, they need to know the value of ρ. Interestingly, Klein & Young [56] constructed examples of the abstract packing problems for which they proved that any Dantzig-Wolfe algorithm must take $\Omega(\rho\epsilon^{-2})$ iterations. However, their examples are limited to the range $\rho\epsilon^{-2} = O(m^{1/2})$. Furthermore, their result applies only to algorithms that access P solely through an oracle that solves linear programs over P. Young [83] presented similar algorithms for linear packing and covering problems from a perspective of oblivious randomized rounding where one does not actually solve the linear programs. Grigoriadis & Khachiyan [33] also provided an approximation algorithm for a generalization of the packing problem, again by following the generic exponential potential function framework. Luby & Nisan [64] also gave a parallel algorithm for positive linear programs. Their algorithm runs in time proportional to $\epsilon^{-4}\log(mn)$ and uses a linear number of processors.

A few years later, Garg & Könemann [31] used an idea of "variable-size increments" to design width-independent algorithms for multicommodity flow and other packing problems. They also use exponential length function to capture congestion on an edge. Their running time for the maximum multicommodity flow problem was later improved by Fleischer [20] using an idea of "round-robin increments". Recently, Karakostas [49] also used a similar technique for maximum concurrent flow problems. Most of this work was concentrated on *pure* packing or *pure* covering problems. Young [84] presented algorithms for mixed covering and packing problems, i.e., the problems with both packing and covering constraints. His algorithms also use Fleischer's round-robin increments and compute approximately feasible solutions to such problems. Recently, Fleischer [21] presented an algorithm for linear fractional covering problems with upper bounds on the variables. This problem can be thought of as a special case of the mixed packing and covering problem. However, Fleischer's algorithm for this special case computes a strictly feasible near-optimum solution. Young [85] is currently writing a tutorial on Lagrangian relaxation via randomized rounding. He uses a randomized rounding approach to the design of Lagrangian relaxation based algorithms.

In a separate line of work, Grigoriadis & Khachiyan [32, 33] presented algorithms for min-max resource sharing problems based on logarithmic potential (or barrier) functions. This was in fact the first width-independent Lagrangian relaxation iteration bound for the min-max resource sharing problems. Later Grigoriadis et al. [35] modified their algorithm to solve max-min resource sharing problems. These papers also consider the special case of block-angular min-max and max-min resource sharing problems respectively. The min-max and max-min resource sharing algorithms were extended by Jansen & Zhang [48] and Jansen [46, 45] to the case of weak approximate block-solvers. Recently Bienstock & Iyengar [9] gave an algorithm for computing a $(1+\epsilon)$ approximation for packing problems by solving $O^*(\epsilon^{-1})$ separable convex quadratic programs. Their algorithm can be thought of as a second-order procedure and is the first one to achieve a dependence of ϵ^{-1} on the error parameter. They also show how to use their algorithm to compute a $(1+\epsilon)$ approximation to the maximum concurrent flow problem. The number of shortest-path computations in this algorithm is proportional to $O^*(\epsilon^{-1})$.

1.3 Contributions and outline of the thesis

This thesis deals with a class of Lagrangian relaxation based algorithms mentioned above. One of our main objectives is to remove the "artistry" in designing such algorithms as far as possible by distilling-out the techniques that keep appearing again and again. This, we hope, will help us understand the existing body of work better and to design and analyze new algorithms. The various abstract optimization and feasibility problems that we consider are shown in Figure 1.1. The problems are arranged in form of a tree. A problem defined in a node can be thought of as a special case of the problem defined in the parent node. As we go down the tree, we are able to exploit special structure of the problems and prove stronger results.

- The main contribution of this thesis is in providing a framework that unifies and generalizes the existing results in Lagrangian relaxation based approximation algorithms for various classes of convex programming problems. We present essentially a *single* exponential-potential-based algorithm which can be modified slightly for different problems to yield state-of-the-art results for a large family of linear and convex programming problems.

  We measure the complexity of our algorithms in terms of the number of basic arithmetic operations and the number of calls to the oracles that solve convex optimization problems over the underlying convex sets.

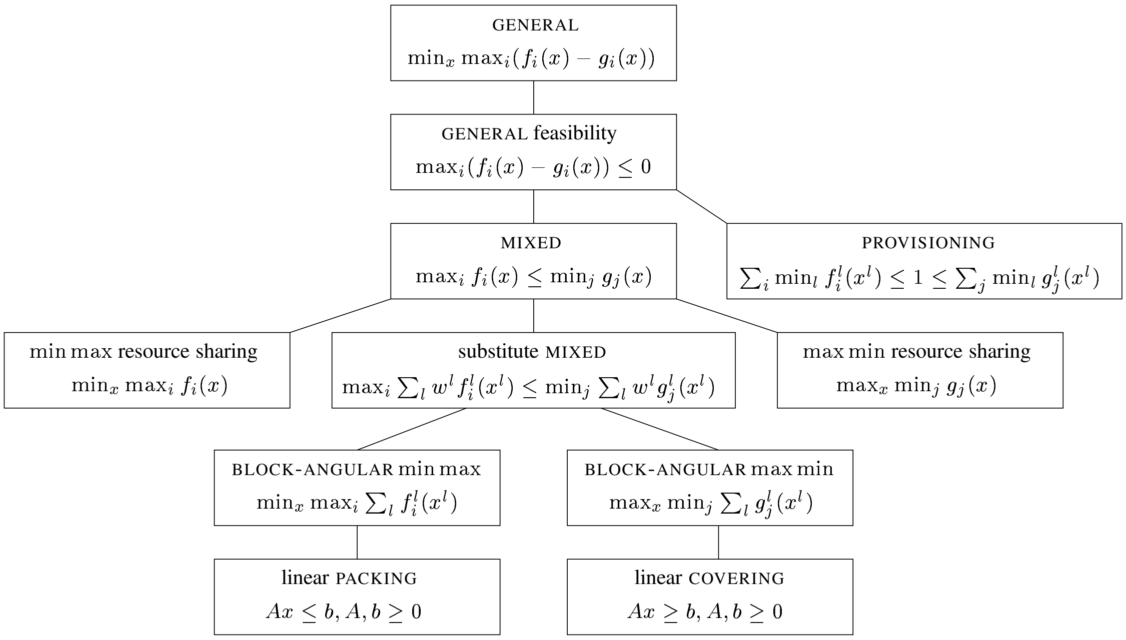

Figure 1.1: Various problems considered in the thesis. A problem can thought of as a special case of the "parent" problem. Here $x \in P$ is a solution variable where $P \subseteq \mathbb{R}^n$ is a convex set. The functions $\{f_i\}$ are convex on P while functions $\{g_j\}$ are concave on P.

Our framework captures *all* existing results in this domain with the exception of the following two results: (a) Grigoriadis et al. [35] and Jansen & Zhang [48] give algorithms for computing $(1+\epsilon)$ approximation to max-min and min-max resource sharing respectively that take $O(m(\epsilon^{-2} + \log m))$ iterations while our algorithm takes $O(m\epsilon^{-2} \log m)$ iterations; (b) Bienstock & Iyengar [9] present a $(1+\epsilon)$ approximation algorithm for computing maximum concurrent flows in $O(\epsilon^{-1})$ iterations while our algorithm takes $O(\epsilon^{-2})$ iterations.

- In Chapter 2, we prove some technical tools that are useful in the rest of the thesis. We present these tools in the context of an on-line prediction game. Similar techniques are also used extensively by the on-line learning community.

- In Chapter 3, we consider an abstract convex optimization problem called the GENERAL problem which allows both positive and negative entries in the constraints. Linear special cases of this problem were considered by Plotkin et al. [70] and Young [83]. Their approach, however, assumes the knowledge of the *width* of the input instance, which may be difficult to estimate.

  We present an algorithm for the GENERAL problem that does not need the value of the width. Unlike their approach of working with just an additive error, we separate out the error as two components — an additive term and a multiplicative term. This gives us an additional flexibility in reducing running times for some problems. For example, when we formulate the fractional Steiner forest problem as a GENERAL problem, using our framework, we can obtain a $(1+\epsilon)$-approximate solution by doing $O^*(mt\epsilon^{-2})$ shortest-path computations, while the approach of Plotkin et al. [70] needs $O^*(m^2t\epsilon^{-2})$ shortest-path computations. Here m and t denote the number of edges and terminal-pairs respectively. We also show how to use our algorithm to compute $(1+\epsilon)$ approximation to the Held-Karp [42, 43] bound for TSP in time $O(m^2\epsilon^{-2}\log^2 m)$. For sparse graphs, this improves the randomized algorithm of Plotkin et al. [70] which has an expected running time of $O(n^4\epsilon^{-2}\log^6 n)$. Here m and n denote the number of edges and vertices in the graph.

- In Chapter 4, we formulate a new problem called the PROVISIONING problem. The fractional Steiner forest problem can be naturally thought of as an instance of the PROVISIONING problem. In this chapter, we also formulate a family of problems called the *hitting-set problem on clutters* [30]. This family includes a variety of combinatorial optimization problems like facility location, generalized Steiner networks, multi-cut, feedback arc-set, k-spanner, etc. We show how to solve the fractional version of the hitting-set problem using the PROVISIONING framework. Thus our algorithm can be used to find near-optimum solutions of the linear programming relaxations of the problems mentioned above.

- In Chapter 5, we consider a MIXED PACKING and COVERING problem that generalizes the linear fractional MIXED problem defined by Young [84]. We solve an open problem posed by him by generalizing his results to convex/ concave functions over arbitrary convex sets. We first obtain a width-dependent running time for the MIXED problem by reducing it to the GENERAL problem. We then use an idea proposed in Garg & Könemann [31] for exploiting "variable-size" increments to obtain a width-independent running time.

  As a special case of the MIXED problem, we also present algorithms for the min-max and max-min resource sharing problems. Our algorithms can work with approximate oracles and compute near-optimum dual solutions for these problems.

- In Chapter 6, we formulate a new problem called the SUBSTITUTE MIXED problem. This abstract problem is motivated by the maximum multicommodity flow problem and is, in fact, not even a convex optimization problem. We show that the algorithm for the MIXED problem presented in Chapter 5 can be extended to solve this problem. Then we generalize Fleischer [20]'s method of "round-robin increments" to design a faster approximation algorithm for this problem.

  The SUBSTITUTE PACKING and COVERING problems are two special cases of the SUBSTITUTE MIXED problem. As applications of these problems, we show how to compute a $(1+\epsilon)$ approximation to the values of the maximum cut in weakly bipartite graphs and the maximum independent set in t-perfect graphs. Our result for the latter problem improves the running time in Eisenbrand et al. [18] by a factor equal to the number of vertices in the graph.

  We show that the block-angular min-max and max-min resource sharing problems can be thought of as special cases of the SUBSTITUTE MIXED problem. This reduction enables us to design $(1+\epsilon)$ approximation algorithms for these problems that need $O((m+t)\epsilon^{-2}\log(m+t))$ calls to individual block-solvers. Here m and t denote the numbers of coupling constraints and blocks respectively. This improves the previous best result of $O(mt\epsilon^{-2}\log m)$ given by Grigoriadis & Khachiyan [32, 33] and Grigoriadis et al. [35].

- In Chapter 7, we show how the algorithms developed in previous chapters can be used for solving families of implicitly or explicitly given linear programming problems. We also present a method to solve a class of linear programs which may have negative entries in the constraint matrix. As an example, we obtain an approximation algorithm for the linear fractional COVERING problem with upper bounds on the variables. Our algorithm makes $O(m\epsilon^{-2}\log m + \log\log C)$ calls to the *most-violated-constraint* oracle improving on the bound of $O(m\epsilon^{-2}\log(mC))$ given by Fleischer [21]; here C is a quantity that depends on the size of numbers in the input.

- In Chapter 8, we conclude with some open problems and directions for future research.

1.4 Basic notation used in the thesis

We use the following standard notations. The term "vector" refers to the column-vector, unless specified otherwise. For vectors u and v, we use $\langle u, v\rangle$ and $u^\top v$ to denote the inner product of the two vectors interchangeably. We use $\mathbf{1}$ to denote a vector of all ones. The dimension of $\mathbf{1}$ is left unspecified and will be clear from the context. For a non-zero vector

$v > 0$, we denote by $\underline{v}$, the "normalized" vector v, i.e., $\underline{v} = v / \sum_i v_i$. Thus $\underline{v}$ has 1-norm one. For a positive integer n, let $[n] = \{1, \ldots, n\}$. We use $x = (x_e : e \in X)$ to denote a vector whose co-ordinates are indexed by the elements in a finite set X. Let $\mathbb{R}$ denote the set of real numbers and $\mathbb{R}_+$ denote the set of non-negative real numbers. Thus $\mathbb{R}_+^n$ denotes the set of n-dimensional real vectors with non-negative entries. We usually use $\epsilon, \delta, \omega \in (0, 1)$ to denote the error parameters. We use $\exp(z) = e^z$ to denote the exponential function. The expression $\ln x$ (resp. $\log x$) denotes the logarithm of $x > 0$ to the base $e = 2.71828...$ (resp. 2). Along with the standard order notations O and Ω, we also use O^* and Ω^*: $O^*(f)$ (resp. $\Omega^*(f)$) denotes a function which is $O(f \log^c f)$ (resp. $\Omega(f \log^{-c} f)$) for a constant $c \geq 0$.

Chapter 2

An on-line prediction problem

In this chapter, we consider a problem called the *on-line prediction problem* which generalizes a problem considered by Auer et al. [4], Vovk [81], Cesa-Bianchi et al. [12] and Freund & Schapire [26, 27] in the context of learning theory. Although this problem may appear unrelated to the topic of this thesis, the main theorem in this chapter will be used at several places in the thesis.

2.1 The problem

In the on-line prediction problem, there is an agent who has m different strategies to choose from. The agent is involved in a "prediction" that goes over many rounds. In round r, the agent decides a distribution $p^r = (p_1^r, \ldots, p_m^r)$ (satisfying $p_i^r \geq 0$ and $\sum_i p_i^r = 1$) on the strategies. In round r, each strategy i earns a profit $\mathcal{P}_i^r$ and incurs a loss $\mathcal{L}_i^r$ that is determined by the (possibly adversarial) "environment". The income made by the ith strategy is then $\mathcal{I}_i^r = \mathcal{P}_i^r - \mathcal{L}_i^r$. The income earned by the agent in round r is the weighted sum $\langle p^r, \mathcal{I}^r \rangle = \sum_i p_i^r \mathcal{I}_i^r = \sum_i p_i^r (\mathcal{P}_i^r - \mathcal{L}_i^r)$. We assume that $-1 \leq \mathcal{I}_i \leq 1$. The goal of the agent is to maximize her cumulative income relative to the income of the best strategy. We can describe the scenario formally as given in Figure 2.1. Let N denote the total number of rounds. The agent chooses a distribution p^r in round r based on the knowledge of the vectors $\mathcal{P}^s, \mathcal{L}^s$ in all previous rounds $1 \leq s < r$. We first observe that no matter how the agent chooses the distributions, the environment can always force her to earn a cumulative income of at most the cumulative income of the best strategy. For instance, if the environment sets $\mathcal{P}_i^r = 1, \mathcal{L}_i^r = 0$ for some $i \in [m]$ and $\mathcal{P}_j^r = \mathcal{L}_j^r = 0$ for $j \neq i$ in each round r, the cumulative income of the agent is at most that of the ith strategy. Thus the agent's goal is to earn a cumulative income that is as close to the best strategy as possible. Let $\mathcal{P} = \sum_{r=1}^N \sum_{i=1}^m p_i^r \mathcal{P}_i^r$ and $\mathcal{L} = \sum_{r=1}^N \sum_{i=1}^m p_i^r \mathcal{L}_i^r$ be the cumulative profit and loss made by the agent respectively. Let $\mathcal{I} = \mathcal{P} - \mathcal{L}$ denote the net cumulative income of the agent. Let $\mathcal{I}_i = \sum_{r=1}^N (\mathcal{P}_i^r - \mathcal{L}_i^r)$ be the cumulative income of the ith strategy in N rounds. We prove the following theorem in Section 2.2.

```
For r = 1, ..., N
    Agent chooses a distribution p^r
    Environment chooses the profit and loss vectors P^r, L^r ∈ R^m_+
    Agent earns a net income of ⟨p^r, P^r − L^r⟩
endFor
```

Figure 2.1: The on-line prediction model

Theorem 2.1.1 *Given any $\epsilon \in (0,1)$, there exists a deterministic algorithm for choosing the distributions such that*

$$\max_{i\in[m]} \mathcal{I}_i \leq \Big(\exp(\epsilon)\cdot\mathcal{P} - \exp(-\epsilon)\cdot\mathcal{L}\Big) + \left(\frac{\ln m}{\epsilon}\right).$$

Here $\exp(\epsilon) = e^{\epsilon}$. We can rewrite the above inequality as

$$\max_{i\in[m]} \mathcal{I}_i \leq \mathcal{I} + \Big((\exp(\epsilon)-1)\mathcal{P} + (1-\exp(-\epsilon))\mathcal{L}\Big) + \left(\frac{\ln m}{\epsilon}\right). \tag{2.1}$$

Note that both $(\exp(\epsilon)-1) = O(\epsilon)$ and $(1-\exp(-\epsilon)) = O(\epsilon)$ for $\epsilon \in (0,1)$. Thus the cumulative income $\mathcal{I}$ of the agent falls short of the cumulative income of the best strategy, $\max_{i\in[m]} \mathcal{I}_i$, by an $O(\epsilon)$ fraction of her cumulative profit and loss and a constant term $\ln m/\epsilon$.

The special cases when either the loss is zero ($\mathcal{L}_i^r = 0$ for all i, r) or the profit is zero ($\mathcal{P}_i^r = 0$ for all i, r) were considered by Auer et al. [4], Vovk [81], Cesa-Bianchi et al. [12] and Freund & Schapire [26, 27]. For these special cases, Theorem 2.1.1 has the following corollaries which match the results in the mentioned work.

Corollary 2.1.2 *If $\mathcal{L}_i^r = 0$ for all $i \in [m], r \in [N]$, then*

$$\max_{i\in[m]} \mathcal{P}_i \leq \exp(\epsilon)\cdot\mathcal{P} + \left(\frac{\ln m}{\epsilon}\right).$$

Proof. Follows directly from Theorem 2.1.1, since in this case, we have $\mathcal{I}_i = \mathcal{P}_i$ and $\mathcal{L} = 0$. ∎

Corollary 2.1.3 *If $\mathcal{P}_i^r = 0$ for all $i \in [m], r \in [N]$, then*

$$\min_{i\in[m]} \mathcal{L}_i \geq \exp(-\epsilon)\cdot\mathcal{L} - \left(\frac{\ln m}{\epsilon}\right).$$

Proof. In this case, we have $\mathcal{I}_i = -\mathcal{L}_i$ and $\mathcal{P} = 0$. ∎

These corollaries imply that within a multiplicative factor of $(1 + O(\epsilon))$ and an additive term of $\ln m/\epsilon$, the agent can earn the maximum profit earned by any strategy and incur the minimum loss incurred by any strategy.

Interestingly, Vovk [81] and Freund & Schapire [26] consider the on-line prediction algorithms with performance bounds of the above form and prove that the above results are almost tight.

Theorem 2.1.4 ([81, 26]) *Suppose there exists a deterministic algorithm for the on-line prediction problem with the following property. There exist real numbers a and c such that for any number of strategies m and any number of rounds N, we have*

$$\mathcal{L} \leq c \cdot \min_{i\in[m]} \mathcal{L}_i + a\cdot \ln m,$$

then for any $\epsilon \in (0,1)$, either

$$c \geq \frac{\ln\left(\frac{1}{1-\epsilon}\right)}{\epsilon} \quad \textit{or} \quad a \geq \frac{1}{\epsilon}.$$

Comparing this with $\mathcal{L} \leq \exp(\epsilon)\cdot\min_{i\in[m]}\mathcal{L}_i + (\exp(\epsilon)/\epsilon)\cdot\ln m$ obtained from Corollary 2.1.3 and noting that

$$\frac{\ln\left(\frac{1}{1-\epsilon}\right)}{\epsilon} \geq \exp(\epsilon),$$

we get that the constants in Corollary 2.1.3 are almost tight.

2.2 The prediction rule

Now we describe the algorithm to choose the distributions that satisfies Theorem 2.1.1. Let $\epsilon \in (0,1)$ be a constant. In round r, let $x_i^r = \sum_{s=1}^{r-1} \mathcal{I}_i^s$ be the cumulative income of the ith strategy over the previous rounds. Then, in this round, the agent chooses the distribution $p_1^r, \ldots, p_m^r$ where

$$p_i^r = \frac{\exp(\epsilon x_i^r)}{\exp(\epsilon x_1^r) + \cdots + \exp(\epsilon x_m^r)}. \tag{2.2}$$

The denominator is a normalization factor so that $\sum_i p_i^r = 1$. Note that if for some $i \in [m]$, the cumulative income x_i^r of the ith strategy is large compared to $x_j^r, j \neq i$, the probability p_i^r is very close to one and the agent assigns a high weight to the ith strategy. On the other hand, if all strategies have approximately the same cumulative income, the agent assigns roughly equal weight to each strategy.

To convey the intuition behind why the prediction rule (2.2) gives Theorem 2.1.1, we first consider the *continuous* version of the problem. We shall prove Theorem 2.1.1 for the discrete version later. In the continuous version, instead of rounds, we have a continuous notion of time. Suppose $x_1, \ldots, x_m$ are the cumulative income of the m strategies till some time t. If $\mathrm{d}x_1, \ldots, \mathrm{d}x_m$ are the incremental incomes of the strategies at time t, the incremental income of the agent at time t is then

$$\sum_{i=1}^{m} p_i \mathrm{d}x_i = \frac{\sum_{i=1}^m \exp(\epsilon x_i)\mathrm{d}x_i}{\sum_{i=1}^m \exp(\epsilon x_i)}.$$

Therefore the cumulative income of the agent is

$$\mathcal{I} = \int_{\mathcal{C}} \frac{\sum_{i=1}^m \exp(\epsilon x_i)\mathrm{d}x_i}{\sum_{i=1}^m \exp(\epsilon x_i)},$$

where $\mathcal{C} \subset \mathbb{R}_+^m$ is a curve from $(0, \ldots, 0)$ to $(\mathcal{I}_1, \ldots, \mathcal{I}_m)$ along which the cumulative incomes of the strategies vary. Let $\Phi(x_1, \ldots, x_m) = \ln \sum_{i=1}^m \exp(\epsilon x_i)$. Then

$$\mathrm{d}\Phi = \frac{\epsilon \sum_{i=1}^m \exp(\epsilon x_i)\mathrm{d}x_i}{\sum_{i=1}^m \exp(\epsilon x_i)}.$$

Therefore,

$$\begin{aligned}
\mathcal{I} &= \frac{1}{\epsilon}\int_{\mathcal{C}} \mathrm{d}\Phi \\
&= \frac{1}{\epsilon}\Big(\Phi(\mathcal{I}_1, \ldots, \mathcal{I}_m) - \Phi(0, \ldots, 0)\Big) \\
&\qquad\qquad \text{(fundamental theorem of calculus along curves [58])} \\
&= \frac{1}{\epsilon}\left(\ln \sum_{i=1}^m \exp(\epsilon \mathcal{I}_i) - \ln m\right) \\
&\geq \frac{1}{\epsilon}\left(\ln \max_{i \in [m]} \exp(\epsilon \mathcal{I}_i) - \ln m\right) \\
&= \frac{1}{\epsilon}\left(\max_{i \in [m]} \ln \exp(\epsilon \mathcal{I}_i) - \ln m\right) \\
&= \max_{i \in [m]} \mathcal{I}_i - \frac{\ln m}{\epsilon}.
\end{aligned}$$

Therefore,

$$\max_{i \in [m]} \mathcal{I}_i \leq \mathcal{I} + \frac{\ln m}{\epsilon}.$$

Thus, in the continuous version of the problem, the agent is able to earn a cumulative income which is at most $\ln m/\epsilon$ less than that of the best strategy. The proof of the discrete version, which is given below, is very similar to that of the continuous version. It replaces the integral by an approximating summation. The factors $\exp(\epsilon)$ and $\exp(-\epsilon)$ in Theorem 2.1.1 come essentially due to the discretization. We first prove some useful identities.

Lemma 2.2.1 *For any $z \in \mathbb{R}$, we have $\exp(z) - 1 \leq z\exp(z)$ and for any $z \in \mathbb{R}_+$ such that $z \neq 0$, we have $z - 1 \geq \ln z$.*

Proof. For any $x \in \mathbb{R}$, we have $\exp(x) \geq 1 + x$. Putting $x = -z$ and multiplying by $\exp(z)$, we get the first inequality. Putting $x = \ln z$, we get the second inequality. ∎

Proof of Theorem 2.1.1. Let $y_i^r = \exp(\epsilon x_i^r)$. Observe that in the continuous version, we had $\Phi = \ln \sum_i y_i^r$. Note that the value of p_i^r is

$$\frac{y_i^{r-1}}{y_1^{r-1} + \cdots + y_m^{r-1}}.$$

We have

$$\begin{aligned}
y_i^r - y_i^{r-1} &= (\exp(\epsilon \mathcal{I}_i^r) - 1) \cdot y_i^{r-1} \\
&\leq \epsilon \mathcal{I}_i^r \exp(\epsilon \mathcal{I}_i^r) \cdot y_i^{r-1} \\
&= \epsilon \exp(\epsilon \mathcal{I}_i^r)(\mathcal{P}_i^r - \mathcal{L}_i^r) \cdot y_i^{r-1} \\
&\leq \epsilon \Big(\exp(\epsilon) \cdot \mathcal{P}_i^r - \exp(-\epsilon) \cdot \mathcal{L}_i^r \Big) \cdot y_i^{r-1}.
\end{aligned}$$

The first inequality holds since $\exp(z) - 1 \leq z\exp(z)$ for $z \in \mathbb{R}$ and the second inequality holds since $-1 \leq \mathcal{I}_i^r \leq 1$. Thus,

$$\Big(\exp(\epsilon)\mathcal{P}_i^r - \exp(-\epsilon)\mathcal{L}_i^r \Big) \cdot y_i^{r-1} \geq \frac{y_i^r - y_i^{r-1}}{\epsilon}.$$

Therefore,

$$\begin{aligned}
\exp(\epsilon) \cdot \mathcal{P} - \exp(-\epsilon) \cdot \mathcal{L} &= \sum_{r=1}^{N} \left(\frac{\sum_{i=1}^m \Big(\exp(\epsilon)\mathcal{P}_i^r - \exp(-\epsilon)\mathcal{L}_i^r \Big) \cdot y_i^{r-1}}{\sum_{i=1}^m y_i^{r-1}} \right) \\
&\geq \frac{1}{\epsilon} \sum_{r=1}^{N} \left(\frac{\sum_{i=1}^m y_i^r - \sum_{i=1}^m y_i^{r-1}}{\sum_{i=1}^m y_i^{r-1}} \right) \\
&\geq \frac{1}{\epsilon} \sum_{r=1}^{N} \left(\ln \frac{\sum_{i=1}^m y_i^r}{\sum_{i=1}^m y_i^{r-1}} \right) \\
&\qquad \text{(since } z - 1 \geq \ln z \text{ with } z = \frac{\sum_{i=1}^m y_i^r}{\sum_{i=1}^m y_i^{r-1}} > 0) \\
&= \frac{1}{\epsilon} \ln \frac{\sum_{i=1}^m y_i^N}{\sum_{i=1}^m y_i^0} \\
&= \frac{1}{\epsilon} \ln \frac{\sum_{i=1}^m \exp(\epsilon \mathcal{I}_i)}{m} \\
&\geq \frac{1}{\epsilon} \ln \frac{\max_{i \in [m]} \exp(\epsilon \mathcal{I}_i)}{m}
\end{aligned}$$

$$\geq \quad \max_{i\in[m]} \mathcal{I}_i - \left(\frac{\ln m}{\epsilon}\right).$$

Thus the proof is complete. ∎

The multiplicative weights prediction rule (2.2) has long been used in diverse areas including on-line learning, playing a repeated game, on-line auctions, portfolio management etc. It was originally suggested by Littlestone & Warmuth [63] and (in a somewhat more sophisticated form) by Vovk [80]. It was also discovered independently by Fudenberg & Levine [28]. The on-line prediction framework is also closely related to the work by Dawid [16] and Foster [23]. The exponential weights were used by Auer et al. [4], Cesa-Bianchi et al. [12] and Freund & Schapire [25] in the context of learning to use experts' advice. It was also used by Freund & Schapire [27] for playing a repeated matrix game and by Blum et al. [10] for learning in on-line auctions. The exponential weights were also used in on-line investment strategies by Cover [14], Ordentlich & Cover [66] and Helmbold et al. [44].

2.3 Solving a matrix game

We now briefly describe an application of the on-line prediction framework and Theorem 2.1.1 to solving a matrix game. This discussion will give a flavor of how we are going to use Theorem 2.1.1 in the rest of this thesis.

Let $M \in \mathbb{R}^{m\times n}$ be a matrix. Suppose there are two players: a row player R and a column player C. The set of strategies of the player R is the set of distributions $p \in [0,1]^m$ on the rows of M, while that of player C is the set of distributions $q \in [0,1]^n$ on the columns of M. If R "plays" a distribution p and C "plays" a distribution q, then C "pays" $p^\top Mq$ to R. Since R tries to maximize the payment he receives, he tries to compute p such that $\min_q p^\top Mq$ is maximized, C tries to minimize the payment he makes and so he tries to compute q such that $\max_p p^\top Mq$ is minimized. A celebrated min-max theorem of John von Neumann [65] says that

$$v = \max_p \min_q p^\top Mq = \min_q \max_p p^\top Mq.$$

The above expression v is called the *value* of the game and p and q that achieve this are called optimum strategies for R and C respectively. We are interested in computing the value and the optimum strategies of the given game.

Assume that the entries in M lie in the range $[0,1]$. We now describe a very simple algorithm to compute the near-optimum strategies by mapping this game to the on-line prediction framework. We think of the column player as the environment and the row player as the agent with as many strategies as the number of rows in M. The algorithm goes in rounds $r = 1,\ldots,N$ for some integer N. In a round r, suppose R plays a distribution p_r on his strategies. Given p_r, the column player C plays a strategy q_r on columns such that $p_r^\top Mq_r$ is minimized. Then the ith strategy earns a profit of $(Mq_r)_i$ and R earns a profit of $(p_r^\top Mq_r$. Now if R chooses his distributions according to the rule (2.2), then from Corollary 2.1.2, we have

$$\max_{i\in[m]} \sum_{r=1}^{N} (Mq_r)_i \leq \exp(\epsilon) \sum_{r=1}^{N} p_r^\top Mq_r + \frac{\ln m}{\epsilon}. \tag{2.3}$$

Now let

$$\bar{p} = \frac{\sum_{r=1}^N p_r}{N} \quad \text{and} \quad \bar{q} = \frac{\sum_{r=1}^N q_r}{N}$$

be the average strategies of R and C respectively. We prove that $\bar{p}$ and $\bar{q}$ are near-optimum strategies for R and C respectively.

If R plays $\bar{p}$, the payment he receives from the adversarial environment is $\min_q \bar{p}^\top Mq$. We show that

$$v \leq \exp(\epsilon) \cdot \min_q \bar{p}^\top Mq + \frac{\ln m}{\epsilon N}.$$

Thus if the number of rounds N is large, then up to a small multiplicative factor and a small additive term, the row player can get maximum profit possible. Similarly, we prove that

$$\max_{i \in [m]} (M\bar{q})_i \leq \exp(\epsilon) \cdot v + \frac{\ln m}{\epsilon N}.$$

Thus by playing $\bar{q}$, the column player C can minimize the worst-case payment that he has to make up to a small multiplicative factor and a small additive term. To prove this, we observe that

$$\begin{aligned}
v = \min_q \max_p p^\top Mq &= \min_q \max_{i \in [m]} (Mq)_i \\
&\leq \max_{i \in [m]} (M\bar{q})_i \\
&= \frac{1}{N} \left(\max_{i \in [m]} \sum_{r=1}^{N} (Mq_r)_i \right) \\
&\leq \frac{1}{N} \left(\exp(\epsilon) \cdot \sum_{r=1}^{N} p_r^\top Mq_r + \frac{\ln m}{\epsilon} \right) \qquad \text{(from (2.3))}.
\end{aligned}$$

Now since in each round r, the column player chose $q_r = \arg\min_q p_r^\top Mq$, the above chain of inequalities gives

$$\begin{aligned}
v \leq \max_{i \in [m]} (M\bar{q})_i &\leq \frac{1}{N} \left(\exp(\epsilon) \cdot \sum_{r=1}^{N} \min_q p_r^\top Mq + \frac{\ln m}{\epsilon} \right) \\
&\leq \frac{1}{N} \left(\exp(\epsilon) \cdot \min_q \sum_{r=1}^{N} p_r^\top Mq + \frac{\ln m}{\epsilon} \right) \\
&= \exp(\epsilon) \cdot \min_q \bar{p}^\top Mq + \frac{\ln m}{\epsilon N} \\
&\leq \exp(\epsilon) \cdot \max_p \min_q p^\top Mq + \frac{\ln m}{\epsilon N} \\
&\leq \exp(\epsilon) \cdot v + \frac{\ln m}{\epsilon N}.
\end{aligned}$$

Thus we have

$$v \leq \max_{i \in [m]} (M\bar{q})_i \leq \exp(\epsilon) \cdot \min_q \bar{p}^\top Mq + \frac{\ln m}{\epsilon N} \leq \exp(\epsilon) \cdot v + \frac{\ln m}{\epsilon N}$$

as claimed.

In the 1950's, John von Neumann proposed and analyzed a similar algorithm for solving two-player zero-sum matrix games in time $O(m^2 n \log(mn)/\epsilon^2)$ where ϵ is the given error parameter [59]. Solving matrix games in a similar manner was also considered by Althöfer [3], Grigoriadis & Khachiyan [34], Lipton & Young [62], Young [82], and Freund & Schapire [27].

Chapter 3

The GENERAL problem

In this chapter, we formulate an abstract convex optimization problem called the GENERAL problem, called so because it generalizes a large family of linear and convex optimization problems. We first consider a feasibility version and later an optimization version of this problem.

3.1 Feasibility version of the GENERAL problem

The feasibility version of the GENERAL problem is defined as follows.

> GENERAL (feasibility): Given a non-empty convex set $P \subseteq \mathbb{R}^n$, non-negative continuous convex functions $f_i : P \to \mathbb{R}_+$ for $1 \le i \le m$, and non-negative continuous concave functions $g_i : P \to \mathbb{R}_+$ for $1 \le i \le m$, either find an $x \in P$ such that
> $$\max_{1 \le i \le m} (f_i(x) - g_i(x)) \le 0,$$
> or prove that no such $x \in P$ exists.

To solve the above problem, we assume that we have the following oracle.

> Given $y \in \mathbb{R}^m_+$, the oracle finds an $x \in P$ such that
> $$\langle y, f(x) - g(x) \rangle \le 0, \tag{3.1}$$
> or proves that no such $x \in P$ exists.

Here $f(x) = (f_1(x), \ldots, f_m(x))$ and $g(x) = (g_1(x), \ldots, g_m(x))$ denote the vectors of functions and $\langle \cdot, \cdot \rangle$ denotes the inner product.

The GENERAL problem (and its optimization version defined in Section 3.2) along-with the corresponding oracles form a common generalization of a large variety of convex and linear programming problems. It generalizes the min-max and max-min resource sharing problems considered by Grigoriadis & Khachiyan [33, 32], Grigoriadis et al. [35], and Jansen & Zhang [48]. When the functions f_i and g_i are *linear*, these problems are in turn identical to the linear fractional packing and covering problems considered by Plotkin et al. [70], Luby & Nisan [64], Young [83], and Bienstock & Iyengar [9, 8]. The GENERAL problem defined above also generalizes *generalized linear packing* [70, 83]. As we shall see in Chapter 5, the GENERAL problem also generalizes the MIXED PACKING and COVERING programs considered by Young [84].

3.1.1 A pseudo approximation algorithm

The main theorem of this section is as follows.

Theorem 3.1.1 *There exists an algorithm for the* GENERAL *feasibility problem that given error parameters* $\epsilon, \delta \in (0, 1)$,

- *either computes an* $x \in P$ *such that*

$$\max_{i \in [m]} \Big(\exp(-\epsilon) f_i(x) - \exp(\epsilon) g_i(x) \Big) \le \delta,$$

- *or proves that there does not exist an* $x \in P$ *such that* $\max_{i \in [m]} (f_i(x) - g_i(x)) \le 0$.

The algorithm makes at most $(1 + \hat{\rho}\epsilon^{-1}\delta^{-1} \ln m)$ *calls to the oracle (3.1), where* $\hat{\rho}$ *is the width of the "perturbed instance" and is given by*

$$\hat{\rho} = \max_{x \in P} \max_{i \in [m]} |\exp(-\epsilon) f_i(x) - \exp(\epsilon) g_i(x)|. \tag{3.2}$$

The algorithm takes additional time equal to the time to compute $f(x)$ *and* $g(x)$ *for some* $x \in P$ *between successive oracle calls.*

In the second case, the algorithm *proves* the infeasibility by producing a certificate of infeasibility as follows. Note that any $x \in P$ that is feasible for the GENERAL problem satisfies $f_i(x) - g_i(x) \le 0$ for all $i \in [m]$. Therefore for any $y \in \mathbb{R}_+^m$, we have $\langle y, f(x) - g(x) \rangle \le 0$. Thus if the oracle (3.1) fails to produce the desired $x \in P$ for some $y \ge 0$, the vector y acts as the certificate of infeasibility and we declare that the given instance is infeasible. Note however that even if the input instance is infeasible, the algorithm may find a solution as in the first case.

Therefore in the rest of the discussion, we assume that the oracle always produces a desired $x \in P$. All we need is the oracle (3.1) and an oracle that computes $f_i(x)$ and $g_i(x)$ for any given $x \in P$. We also assume that the width $\hat{\rho}$ defined in (3.2) is finite.

Refer to Algorithm 3.1. We associate Lagrangian multipliers (or dual variables) $y_i \ge 0$ with the functions $f_i - g_i$. The variables y_i are initialized to 1 (line (1)). The algorithm goes in rounds r. In each round it makes a call to oracle (3.1) with the current values of y_i (line (6)). The weight $w_r = 1/\max_{i \in [m]} |\hat{h}_i(x_r)|$ defined in line (7) gives the extent to which we pick x_r in the final solution. We use a variable W to keep track of how much weight is accumulated so far. We update the dual variables y in line (9) according to an exponential rule. The algorithm comes out of the loop (line (10)) if the weight accumulated W is at least $\epsilon^{-1}\delta^{-1} \ln m$ where $\epsilon, \delta \in (0, 1)$ are the given error parameters. The algorithm outputs an $\tilde{x} \in P$ which is a weighted convex combination of the points $x_r \in P$ returned in round r by the oracle.

Remark 3.1.2 *Our algorithm does not need the convex set* P *and the functions* f *and* g *to be given explicitly. All it needs is the oracle (3.1) and an oracle that given* $x \in P$ *returns* $f(x)$ *and* $g(x)$. *The complexity of the algorithms will be measured in terms of the total number of oracle calls that the algorithm makes.*

Proof of Theorem 3.1.1

We relate Algorithm 3.1 to an instance of the on-line prediction problem described in Chapter 2 as follows. Suppose there is an agent who has m strategies corresponding to each of the m functions $f_i(x) - g_i(x), i \in [m]$. The on-line prediction goes in rounds $r = 1, \ldots, N$ where N denotes the total number of rounds in Algorithm 3.1. Consider a round r. Let x_r denote the output of the oracle in this round. Let $w_r = 1/\max_{i \in [m]} |\hat{h}_i(x_r)|$, where $\hat{h}_i(x_r) = \exp(-\epsilon) f_i(x_r) - \exp(\epsilon) g_i(x_r)$ is the

Algorithm GENERAL (feasibility)

Input : Convex set $P \subseteq \mathbb{R}^n$, functions $f, g : P \to \mathbb{R}_+^m$, and $\epsilon, \delta \in (0,1)$
Output : "infeasible" or $\tilde{x} \in P$

(1)	For $i \in [m]$ do $y_i := 1$	{initialize the dual variables}
(2)	Define $h_i(x) := f_i(x) - g_i(x)$	
	Define $\hat{h}_i(x) := \exp(-\epsilon) f_i(x) - \exp(\epsilon) g_i(x)$	
(3)	$r := 0$	{initialize the round number}
(4)	$W := 0$	{initialize the weight}
	Repeat	
(5)	$r := r + 1$	{round r begins}
(6)	Find $x_r \in P$ such that $\langle y, h(x_r) \rangle \leq 0$	{oracle call}
	If there is no such $x_r \in P$, then output "infeasible"	
(7)	$w_r := 1/\max_{i \in [m]} \lvert \hat{h}_i(x_r) \rvert$	{note that $w_r \geq 1/\hat{\rho}$}
(8)	$W := W + w_r$	{increment weight}
(9)	For $i \in [m]$ do $y_i := y_i \cdot \exp(\epsilon w_r \hat{h}_i(x_r))$	{update dual variables}
(10)	Until $W \geq \epsilon^{-1} \delta^{-1} \ln m$	{stop if enough weight gathered}
(11)	Output $\tilde{x} = (\sum_{s=1}^{r} w_s x_s)/(\sum_{s=1}^{r} w_s)$	

Algorithm 3.1: An algorithm for the GENERAL feasibility problem

perturbed function. Imagine that the ith strategy earns a profit

$$\mathcal{P}_i^r = w_r \exp(-\epsilon) f_i(x_r) \tag{3.3}$$

and incurs a loss

$$\mathcal{L}_i^r = w_r \exp(\epsilon) g_i(x_r). \tag{3.4}$$

Thus the net income of the ith strategy in round r is $\mathcal{I}_i^r = w_r \hat{h}_i(x_r)$. Note that from the way w_r is defined, we have $-1 \leq \mathcal{I}_i^r \leq 1$. Let y_i^{r-1} denote the value of the dual variable y_i at the end of round $r-1$. Note that, from the way we update the duals in line (9), we have $y_i^{r-1} = \exp(\epsilon \sum_{s=1}^{r-1} \mathcal{I}_i^s)$. Now suppose that in this round, the agent chooses a distribution p^{r-1} defined by

$$p_i^{r-1} = \frac{y_i^{r-1}}{y_1^{r-1} + \cdots + y_m^{r-1}}, \tag{3.5}$$

which is similar to (2.2). Let the cumulative income of the ith strategy at the end of N rounds be denoted by $\mathcal{I}_i$ and the cumulative profit and loss of the agent be denoted by $\mathcal{P}$ and $\mathcal{L}$ respectively. Then from Theorem 2.1.1, we have

$$\max_{i \in [m]} \mathcal{I}_i \leq \Big(\exp(\epsilon)\mathcal{P} - \exp(-\epsilon)\mathcal{L} \Big) + \left(\frac{\ln m}{\epsilon} \right). \tag{3.6}$$

Note that $\mathcal{I}_i = \sum_{r=1}^{N} w_r \hat{h}_i(x_r)$. Thus

$$\max_{i \in [m]} \mathcal{I}_i = \max_{i \in [m]} \sum_{r=1}^{N} w_r \hat{h}_i(x_r). \tag{3.7}$$

Further

$$\exp(\epsilon)\mathcal{P} - \exp(-\epsilon)\mathcal{L} \quad = \quad \sum_{r=1}^{N} \frac{\sum_{i=1}^{m} y_i^{r-1} \Big(\exp(\epsilon)\mathcal{P}_i^r - \exp(-\epsilon)\mathcal{L}_i^r \Big)}{\sum_{i=1}^{m} y_i^{r-1}}$$

$$\begin{aligned}
&= \sum_{r=1}^{N} \frac{\sum_{i=1}^{m} y_i^{r-1} w_r \Big(f_i(x_r) - g_i(x_r) \Big)}{\sum_{i=1}^{m} y_i^{r-1}} \\
&= \sum_{r=1}^{N} w_r \cdot \frac{\langle y^{r-1}, h_i(x_r) \rangle}{\sum_{i=1}^{m} y_i^{r-1}} \\
&\leq 0
\end{aligned}$$

where the last inequality follows from the fact that $\langle y^{r-1}, h_i(x_r) \rangle \leq 0$ for each round r. Now from (3.6) and (3.7), we have,

$$\max_{i \in [m]} \sum_{r=1}^{N} w_r \hat{h}_i(x_r) \leq \left(\frac{\ln m}{\epsilon} \right). \tag{3.8}$$

Let $\tilde{x}$ be the output of the algorithm given by

$$\tilde{x} = \frac{\sum_{r=1}^{N} w_r x_r}{\sum_{r=1}^{N} w_r}.$$

Since P is a convex set, we have $\tilde{x} \in P$. Now from the fact that the functions $\hat{h}_i$ are convex, we get,

$$\begin{aligned}
\max_{i \in [m]} \hat{h}_i(\tilde{x}) &\leq \max_{i \in [m]} \frac{\sum_{r=1}^{N} w_r \hat{h}_i(x_r)}{\sum_{r=1}^{N} w_r} \\
&\leq \left(\frac{\ln m}{\epsilon \sum_{r=1}^{N} w_r} \right) \\
&\leq \delta.
\end{aligned}$$

The last inequality follows from the stopping condition (line (10)) that $\sum_{r=1}^{N} w_r \geq \epsilon^{-1}\delta^{-1} \ln m$. Thus the output $\tilde{x}$ satisfies the condition in Theorem 3.1.1. Now we prove a bound on the total number of oracle calls. Since we make exactly one oracle call in each round, it is enough to bound the number of rounds N. From the stopping condition (line (10)) of the algorithm, we have

$$\sum_{r=1}^{N-1} w_r < \epsilon^{-1}\delta^{-1} \ln m \leq \sum_{r=1}^{N} w_r.$$

From the definition of $w_r = 1/\max_{i \in [m]} |\hat{h}_i(x_r)|$, we have $w_r \geq 1/\hat{\rho}$ where $\hat{\rho}$ is as defined in (3.2). Therefore, $N \leq 1 + \hat{\rho}\epsilon^{-1}\delta^{-1} \ln m$. This completes the proof of Theorem 3.1.1.

Remark 3.1.3 *The fact that the functions f_i (resp. g_i) are convex (resp. concave) was used only in the last step. The inequality (3.8) holds even if f_i are not convex and g_i are not concave.*

Using approximate oracles

In some cases, the exact oracle (3.1) may be either computationally very expensive or not available at all. It is easy to show that Algorithm 3.1 also works with approximate oracles. Recall that $\underline{y} = y / \sum_i y_i$ is the vector y normalized.

Theorem 3.1.4 *If we are given an approximate version of the oracle (3.1), which given any $y \geq 0$, computes an $x \in P$ such that $\langle \underline{y}, f(x) - g(x) \rangle \leq \gamma$ for some $\gamma \in \mathbb{R}$, then Algorithm 3.1 computes an $\tilde{x} \in P$ such that*

$$\max_{i \in [m]} \Big(\exp(-\epsilon) f_i(\tilde{x}) - \exp(\epsilon) g_i(\tilde{x}) \Big) \leq \gamma + \delta.$$

The algorithm makes at most $(1 + \hat{\rho}\epsilon^{-1}\delta^{-1} \ln m)$ *oracle calls, where* $\hat{\rho}$ *is given by (3.2).*

Proof. The proof is similar to the one for Theorem 3.1.1 except that now we have $\exp(\epsilon)\mathcal{P} - \exp(-\epsilon)\mathcal{L} \leq \gamma \sum_{r=1}^{N} w_r$. ∎

Note that the algorithm that works with an approximate oracle does not need to know the value of γ and that the number of oracle calls is also independent of γ.

3.1.2 An illustration: the fractional Steiner forest problem

In this section, we show how the framework of the GENERAL problem can be used to find a $(1 + \omega)$ approximation to the fractional Steiner forest problem by making $O^*(mt\omega^{-2})$ calls to the single-source shortest-path oracle in a graph with m edges with non-negative lengths. Here t denotes the number of terminal-pairs in the input. We proved the same running time bound for this problem by using a slightly different, but equivalent, approach in Garg & Khandekar [30].

The *Steiner forest* problem is defined as follows. Given a (directed or undirected) graph $G = (V, E)$ with costs $c_e \geq 0$ on the edges $e \in E$ and t pairs of vertices (u_l, v_l) for $l \in [t]$, find a minimum-cost sub-graph of G that has a path between u_l and v_l for each $l \in [t]$.[1] Suppose that we are interested in solving the *fractional* version of this problem, i.e., we would like to install *capacities* $x_e \geq 0$ on the edges $e \in E$ such that one is able to send a unit flow between pair (u_l, v_l) for $l \in [t]$ and the total cost $\sum_e c_e x_e$ is minimized. Note that we do not require that flows between different pairs be routed concurrently. We formulate this problem as a linear program. With each path p between some pair (u_l, v_l), we associate a variable $x_p \geq 0$ which denotes the amount of flow on the path p. The fractional Steiner forest problem is then equivalent to solving the following linear program.

$$\begin{array}{rcll} \min & & \sum_{e \in E} c_e x_e & \\ \text{s.t.} & & & \\ \sum_{p \sim l} x_p & = & 1 & \forall l \in [t] \\ x_e & \geq & \sum_{p: e \in p, p \sim l} x_p & \forall l \in [t], \forall e \in E \\ x_e, x_p & \geq & 0 & \forall e \in E, \forall p \end{array} \tag{3.9}$$

Here $p \sim l$ denotes a path p between the pair (u_l, v_l). The first set of constraints enforce that we send a unit flow between each pair (u_l, v_l); and the second set of constraints ensure that we install sufficient capacities x_e to support these flows individually. There are alternate linear programming formulations for this problem; in particular, there is one with flow-variables for each edge and (u_l, v_l) pair. These formulations are, in fact, equivalent. We work with the formulation (3.9) since it simplifies the description of our algorithm. We assume, without loss of generality, that $c_e > 0$ for all $e \in E$. Any edge with $c_e = 0$ can be picked to an extent one and be shrunk to form a new instance.

Feasibility version of the fractional Steiner forest problem

Consider a feasibility version of (3.9) defined as: given $\alpha > 0$, determine whether there exists a feasible solution $\{x_e\}$ and $\{x_p\}$ for (3.9) such that the total cost $\sum_{e \in E} c_e x_e \leq \alpha$. We first present an algorithm for the feasibility version and later modify it to solve the optimization version.

Theorem 3.1.5 *There exists an algorithm that given any* $\alpha > 0$ *and an error parameter* $\omega \in (0, 1)$*, either finds a feasible solution to (3.9) with* $\sum_{e \in E} c_e x_e \leq \alpha \cdot \exp(\omega)$*, or proves that there is no feasible solution with* $\sum_{e \in E} c_e x_e \leq \alpha$*. The algorithm makes* $O^*(mt\omega^{-2})$ *single-source shortest-path computations on a graph with* m *edges having non-negative lengths.*

[1] Note that if a graph has a path from u to v and a path from v to w, then it has a path from u to w. Therefore in the undirected setting we can, without loss of generality, assume that $t \leq |V| - 1$.

We prove this theorem in the rest of the section. Define a polytope P^l to be the set of all unit flows between (u_l, v_l), i.e., $P^l = \{(x_p : p \sim l) \geq 0 \mid \sum_{p\sim l} x_p = 1\}$. Let $X = \{(x_e : e \in E) \geq 0 \mid \sum_{e\in E} c_e x_e \leq \alpha\}$ be the polytope of edge-capacity vectors with total cost at most α. Define $P = P^1 \times \cdots \times P^t \times X$ to be the Cartesian product of these polytopes. For $x = (x^1, \ldots, x^t, x_E) \in P$, the point $x^l \in P^l$ denotes a unit flow between (u_l, v_l) and the point $x_E = (x_e : e \in E) \in \mathbb{R}_+^{|E|}$ denotes the edge-capacities. Now for $x \in P$, $e \in E$, and $l \in [t]$ define

$$f_e^l(x) = \left(\frac{c_e}{\alpha}\right) \sum_{p: e\in p, p\sim l} x_p \quad \text{and} \quad g_e(x) = \left(\frac{c_e}{\alpha}\right) x_e. \tag{3.10}$$

The feasibility version of the Steiner forest problem can now be formulated as finding an $x \in P$ such that

$$f_e^l(x) - g_e(x) \leq 0 \qquad \forall e \in E, l \in [t].$$

The above constraints ensure that the capacities $x_E = (x_e : e \in E)$ are enough to route unit flows and since $x_E \in X$, the total cost $\sum_{e\in E} c_e x_e$ is at most α.

Oracle

To solve this problem using the GENERAL algorithm, we need the following oracle. Given dual variables $\{y_e^l \geq 0 \mid e \in E, l \in [t]\}$, find an $x \in P$ such that

$$\sum_{l\in[t]} \sum_{e\in E} y_e^l \left(f_e^l(x) - g_e(x)\right) \leq 0. \tag{3.11}$$

Clearly any feasible point $x \in P$ satisfies $f_e^l(x) - g_e(x) \leq 0$ for all $e \in E$ and $l \in [t]$ and therefore for this $x \in P$, the above inequality is satisfied.

To find such an $x \in P$, we rewrite the left-hand-side of (3.11) as

$$\sum_{l\in[t]} \sum_{p\sim l} \left(x_p \sum_{e\in p} \frac{y_e^l c_e}{\alpha}\right) - \sum_{e\in E} \left(\frac{c_e x_e}{\alpha} \sum_{l\in[t]} y_e^l\right). \tag{3.12}$$

Note that any $x \in P$ satisfies that $\sum_{p\sim l} x_p = 1$ for all $l \in [t]$. Therefore to minimize the first term in (3.12) for each $l \in [t]$, we should set $x_p = 1$ for $p \sim l$ such that $\sum_{e\in p} y_e^l c_e/\alpha$ is minimized. Such a path $p \sim l$ can be found by doing a shortest-path computation between u_l and v_l with (non-negative) edge-lengths $y_e^l c_e$. Note that these edge-lengths are different for different values of $l \in [t]$ and therefore we cannot combine these shortest-path computations. For any $x \in P$, we have $\sum_{e\in E} c_e x_e \leq \alpha$. Therefore the second term in (3.12) would be largest when we set $x_e = \alpha/c_e$ for the edge $e \in E$ that maximizes $\sum_{l\in[t]} y_e^l$ and set $x_{e'} = 0$ for the edges $e' \neq e$. Thus we compute $x \in P$ that minimizes (3.12) by doing t single-source shortest-path computations. If the expression (3.12) turns out to be greater than zero, we output that the given instance is infeasible.

Since we can implement the desired oracle, from Theorem 3.1.1, Algorithm 3.1 computes an $x \in P$ such that

$$\exp(-\epsilon) f_e^l(x) - \exp(\epsilon) g_e(x) \leq \delta \qquad \forall e \in E, l \in [t].$$

If we set $\delta = \epsilon/m$, we get

$$f_e^l(x) \leq \exp(\epsilon) \left(\exp(\epsilon) g_e(x) + \frac{\epsilon}{m}\right) \qquad \forall e \in E, l \in [t]$$

which implies

$$\sum_{p:e\in p, p\sim l} x_p \leq \frac{\alpha}{c_e}\left(\exp(2\epsilon)g_e(x) + \frac{\epsilon\exp(\epsilon)}{m}\right).$$

Hence we can set $\tilde{x}_e$ to be the expression on the right-hand-side of the above inequality. Then

$$\begin{aligned}\sum_{e\in E} c_e\tilde{x}_e &\leq \sum_{e\in E}\alpha\left(\exp(2\epsilon)g_e(x) + \frac{\epsilon\exp(\epsilon)}{m}\right)\\ &\leq \alpha\Big(\exp(2\epsilon) + \epsilon\exp(\epsilon)\Big) \qquad\qquad (\text{since } \sum_{e\in E} g_e(x) \leq 1)\\ &\leq \alpha\cdot\exp(3\epsilon).\end{aligned}$$

Thus the variables $\{\tilde{x}_e\}$ and $\{x_p\}$ form a feasible solution to (3.9) with total cost $\sum_{e\in E} c_e\tilde{x}_e \leq \alpha\cdot\exp(3\epsilon)$. By setting $\epsilon = \omega/3$, we get the desired inequality.

Reducing the width

Now we analyze the running time of the algorithm. From Theorem 3.1.1, the algorithm makes $O(\hat{\rho}\epsilon^{-1}\delta^{-1}\ln(mt))$ oracle calls. Note that the number of functions $f_e^l(x) - g_e(x)$ is mt. The width $\hat{\rho}$ is given by

$$\begin{aligned}\hat{\rho} &= \max_{e\in E}\max_{l\in[t]}\max_{x\in P}|\exp(-\epsilon)f_e^l(x) - \exp(\epsilon)g_e(x)|\\ &\leq \max\{\max_{e\in E}\frac{c_e}{\alpha}, \exp(\epsilon)\}.\end{aligned}$$

The inequality holds since we have $f_e^l(x) \leq c_e/\alpha$ and $g_e(x) \leq 1$ for any $x \in P$. Note that $\hat{\rho}$ can be very large if there are edges with cost c_e much larger than α. Observe, however, that such edges will not get used in any optimum solution. Thus we can remove these edges without affecting the optimum solution and make the width constant. The above observation is formally described as follows.

Lemma 3.1.6 *Let* $\mathsf{OPT} = \sum_e c_e x_e^*$ *be the value of an optimum solution* $\{x_e^*\}$ *to the linear program (3.9). Then for any edge* $e \in E$ *with* $c_e > \mathsf{OPT}$*, we have,* $x_e^* = 0$.

Proof. Suppose there is an edge $e \in E$ with $c_e > \mathsf{OPT}$ and $x_e^* = \gamma > 0$. The graph with edges $E\setminus\{e\}$ and capacities x^* supports at least $1-\gamma$ units of flow of between each pair. Therefore, if we scale-up these capacities by a factor $1/(1-\gamma)$, we get a feasible solution to the linear program. This solution has cost

$$\frac{1}{1-\gamma}(\mathsf{OPT} - c_e\cdot\gamma) < \frac{1}{1-\gamma}(\mathsf{OPT} - \mathsf{OPT}\cdot\gamma) = \mathsf{OPT}.$$

This is a contradiction since OPT is the cost of the optimum solution. Thus the lemma is proved. ∎

Since we are checking if $\mathsf{OPT} \leq \alpha$, we can remove edges with cost more than α. Since now $\hat{\rho} = O(1)$ and $\delta = \epsilon/m$ and $\epsilon = \omega/3$, the total number of oracle calls made by the algorithm is $O(m\epsilon^{-2}\ln(mt)) = O(m\omega^{-2}\ln(mt))$. Since in each oracle call, we have to perform t shortest-path computations, the total number of shortest-path computations is $O(mt\omega^{-2}\ln(mt))$. This proves Theorem 3.1.5.

Optimization version of the fractional Steiner forest problem

Let OPT denote the optimum value of the linear program (3.9). We would like to find a feasible solution $\{x_e\}$ and $\{x_p\}$ for (3.9) such that the total cost $\sum_{e\in E} c_e x_e \le \exp(\omega)\cdot \mathsf{OPT}$. To get an estimate of OPT, we use the following lemma.

Lemma 3.1.7 *For $l \in [t]$, let Λ_l denote the cost of the shortest path between u_l and v_l under the cost function $\{c_e\}$. Then, $\max_{l\in[t]} \Lambda_l \le \mathsf{OPT} \le \sum_{l=1}^{t} \Lambda_l \le t\cdot \max_{l\in[t]} \Lambda_l$.*

Proof. For any $l \in [t]$, the cost of routing unit flow between u_l and v_l in any fractional solution is at least Λ_l. On the other hand, union of the shortest u_l-v_l paths for all $l \in [t]$ forms a feasible fractional solution and has cost at most $\sum_{l\in[t]} \Lambda_l$. ∎

Thus we have lower and upper bounds on OPT that are within a factor t of each other. Young [84] presents a general method of reducing optimization to feasibility. We outline this method here. Let $\omega \in (0, 1)$ be the given error parameter and let L and U be lower and upper bounds on OPT such that $U = \beta L$. Initially we have $\beta \le t$. In the first phase, we make $O(\log\log t)$ runs of the feasibility algorithm to reduce β to a constant as follows. We run the feasibility algorithm with estimate $\alpha = \sqrt{\beta}L$ and error parameter $\omega' = 1/2$. The algorithm either computes a feasible solution with cost at most $\alpha\cdot\exp(\omega')$ or proves that there is no feasible solution with cost at most α. In the former case, we update the upper bound to $U' = \sqrt{\beta}L\exp(1/2)$ while in the latter case, we update the lower bound to $L' = \sqrt{\beta}L$. Thus we can reduce the ratio of upper and lower bounds from β to at most $\sqrt{\beta}\exp(1/2)$. By repeating this $O(\log\log t)$ times, we can reduce the ratio β to a constant.

Now in the second phase, we progressively increase the precision and reduce β geometrically as follows. We first run the feasibility algorithm with estimate $\alpha = \beta^{1/3}L$ and error parameter $\omega' = \ln(\beta^{1/3})$. If the algorithm computes a feasible solution with cost at most $\alpha\cdot\exp(\omega')$, we update the upper bound to $U' = \beta^{2/3}L$, otherwise we update the lower bound to $L' = \beta^{1/3}L$. Thus in one run, we reduce β to $\beta^{2/3}$. By repeating this $O(\log(1/\omega))$ times, we make $\beta \le \exp(\omega)$.

Thus we can reduce the optimization problem to a sequence of $O(\log\log t)$ feasibility problems with $\omega' = 1/2$ and $O(\log(1/\omega))$ feasibility problems where ith-to-last subproblem has $\omega' = \Omega(\omega(\frac{3}{2})^i)$. Since ω' decreases geometrically in the second phase, the total time taken for the second phase is dominated by that for the last feasibility problem. Thus for the fractional Steiner forest problem, to compute an $\exp(\omega)$ approximation to OPT, we have to do $O(mt\log(mt)(\log\log t + \omega^{-2}))$ shortest-path computations. Such a method of reducing optimization to feasibility was also used by Hassin [41], Raz & Lorenz [71], and Ergun et al. [19] in the context of finding restricted shortest paths.

Comparison with other algorithms for the fractional Steiner forest problem

The algorithm described above was given by Garg & Khandekar [30]. As we shall see in Chapter 4, this algorithm generalizes to a wide variety of fractional problems which we call *generalized hitting set problems*. Recently, Fleischer [21] presented an algorithm for the Steiner network design in which one may want to route more than unit flow between the source-sink pairs. Her algorithm specialized to the fractional Steiner forest problem has a running time of $O^*(m(m + n\min\{n, t\})\omega^{-2})$ which is better especially when t is large. In the next section, we shall prove that if we take a GENERAL-LP approach similar to Plotkin et al. [70] or Young [83], one needs to do $O^*(m^2t\omega^{-2})$ shortest-path computations. This is a factor m more than that for the algorithm presented above. The optimum fractional solution can also be computed using Vaidya's algorithm [79] for linear programming which runs in time $O(c^{1.5}vL)$ where c, v are the number of constraints and variables and L is the number of bits of precision required. The only "compact" linear program equivalent to (3.9) that we know of is obtained by introducing flow-variables for each edge and (u_l, v_l) pair. This formulation has $\Omega(mt)$ variables and constraints. Further, since in an optimum solution, the variables can take values that are exponentially small in n, we have $L = \Omega(n)$ and the running time of Vaidya's algorithm is $O(nm^{2.5}t^{2.5})$.

3.1.3 A true approximation algorithm

We now show that Algorithm 3.1 can also be used to find a *true* approximation to the GENERAL feasibility problem. Define the width ρ of an instance of the GENERAL feasibility problem as

$$\rho = \max_{x \in P} \max_{i \in [m]} |f_i(x) - g_i(x)|. \tag{3.13}$$

Assume that we know the value of ρ. We prove the following theorem.

Theorem 3.1.8 *There exists an algorithm for the* GENERAL *feasibility problem that given an error parameter $\omega \in (0,1)$,*

- *either computes an $x \in P$ such that*
$$\max_{i \in [m]} \Big(f_i(x) - g_i(x) \Big) \le \omega,$$
- *or proves that there does not exist an $x \in P$ such that $\max_{i \in [m]}(f_i(x) - g_i(x)) \le 0$.*

The algorithm makes $O(1 + \rho(1+\rho)\omega^{-2} \log m)$ calls to the oracle (3.1) and takes additional time equal to the time to compute $f(x) - g(x)$ for some $x \in P$ between successive oracle calls.

Proof. Define functions $f_i'(x) = f_i(x) - g_i(x) + \rho$ and $g_i'(x) = \rho$ for $x \in P$ and $i \in [m]$. Note that $f_i'(x)$ (resp. $g_i'(x)$) are non-negative convex (resp. concave) functions. Now let us run Algorithm 3.1 for these functions with error parameter $\epsilon, \delta \in (0,1)$ to be specified later. Since $f_i(x) - g_i(x) = f_i'(x) - g_i'(x)$, we can continue to use the same oracle. Now from Theorem 3.1.1, the algorithm computes an $x \in P$ such that

$$\max_{i \in [m]} \Big(\exp(-\epsilon) f_i'(x) - \exp(\epsilon) g_i'(x) \Big) \le \delta,$$

by making at most $(1 + \hat{\rho}\epsilon^{-1}\delta^{-1} \ln m)$ oracle calls where $\hat{\rho} = \max_{x \in P} \max_{i \in [m]} |\exp(-\epsilon) f_i'(x) - \exp(\epsilon) g_i'(x)|$. Note that

$$\exp(-\epsilon) f_i'(x) - \exp(\epsilon) g_i'(x) \quad = \quad \exp(-\epsilon)\Big(f_i(x) - g_i(x) \Big) + \Big(\exp(-\epsilon) - \exp(\epsilon) \Big)\rho.$$

Since $\epsilon \in (0,1)$, we have $-3\epsilon \le \exp(-\epsilon) - \exp(\epsilon) \le 0$. Therefore, $\hat{\rho} \le \exp(-\epsilon)\rho + 3\epsilon\rho = O(\rho)$. Furthermore,

$$\max_{i \in [m]} \exp(-\epsilon)\Big(f_i(x) - g_i(x) \Big) \le \delta + 3\epsilon\rho$$

which gives

$$\max_{i \in [m]} \Big(f_i(x) - g_i(x) \Big) \le \delta \exp(\epsilon) + 3\epsilon\rho \exp(\epsilon).$$

Now to ensure that $\delta \exp(\epsilon) + 3\epsilon\rho\exp(\epsilon) \le \omega$, we choose $\epsilon = \omega/(1 + 18\rho)$ and $\delta = \omega/6$. Note that we have added 1 to 18ρ to ensure that $\epsilon \in (0,1)$. The number of oracle calls then becomes $(1 + \hat{\rho}\epsilon^{-1}\delta^{-1} \log m) = O(1 + \rho(1+\rho)\omega^{-2} \log m)$. ∎

The fractional Steiner forest problem revisited

We now reconsider the fractional Steiner forest problem but this time with Theorem 3.1.8. The linear programming formulation and the oracle is identical to the one in Section 3.1.2. From Theorem 3.1.8, we get that the new algorithm computes an $x \in P$ such that

$$f_e^l(x) - g_e(x) \le \omega \qquad \forall e \in E, l \in [t].$$

We would now have to set $\tilde{x}_e$ to $\alpha/c_e \cdot (g_e(x) + \omega)$ so that

$$\begin{aligned}\sum_{e \in E} c_e \tilde{x}_e &= \alpha \sum_{e \in E} (g_e(x) + \omega) \\ &\leq \alpha(1 + m\omega).\end{aligned}$$

Now to get a $(1 + \tilde{\omega})$ approximation, we need to set $\omega = \tilde{\omega}/m$. This gives the total number of oracle calls as $O(1 + \rho(1 + \rho)m^2\tilde{\omega}^{-2} \ln(mt))$. Again using the same trick of removing high-cost edges to make ρ constant, we can make the number of oracle calls $O(m^2\tilde{\omega}^{-2} \ln(mt))$. Since in each oracle call, we make t shortest-path computations, the total number of shortest-path computations is $O(m^2 t\tilde{\omega}^{-2} \ln(mt))$. This bound is a factor m worse than the one in Theorem 3.1.5. Thus in the case of fractional Steiner forest problem, Theorem 3.1.1 gives strictly better running time than Theorem 3.1.8. The basic problem with this approach is that it sets ϵ and δ so that δ/ϵ is approximately equal to ρ while the approach developed in Section 3.1.1 has a flexibility of setting the values of ϵ and δ differently.

3.1.4 Comparison with previous work

A notable feature that distinguishes the GENERAL problem from most of the problems that were solved earlier using Lagrangian relaxation is that it handles both *positive* and *negative* entries simultaneously. Note that the functions $f_i(x) - g_i(x)$ can take both positive and negative values. The only previous attempts to handle such problems were by Plotkin et al. [70] and Young [83]. They considered a linear version of the GENERAL problem in which the objective is to find an $x \in P$ such that $Ax \leq b$ where $P \subseteq \mathbb{R}^n$ is a convex set and the entries of $A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m$ could be positive or negative. We refer to this problem as GENERAL-LP. If the instance is feasible, given $\omega \in (0, 1)$, their algorithm computes an $x \in P$ such that $Ax \leq b + \omega\mathbf{1}$. The algorithm makes $O(\rho^{-2}\omega^{-2} \log(m\rho\omega^{-1}))$ calls to the oracle that given $y \in \mathbb{R}^m_+$, finds an $x \in P$ such that $y^\top Ax$ is minimized. Plotkin et al. [70] defined the width ρ as $\rho = 1 + \max_{x \in P} \max_{i \in [m]} |(Ax)_i - b_i|$. The number of oracle calls was later improved to $O(\rho^{-2}\omega^{-2} \log m)$ by Young [83]. Given an instance of the GENERAL-LP problem, let $f(x) - g(x) = Ax - b$. For the algorithm to find a true approximation, we do not need to know $f(x)$ and $g(x)$ separately. Given the oracle of Plotkin et al., one can easily implement the oracle (3.1) of finding an $x \in P$ (if it exists) such that $y^\top(Ax - b) \leq 0$ simply by minimizing $y^\top Ax$ over $x \in P$. It is now easy to see that Theorem 3.1.8 subsumes the results of Plotkin et al. [70] and Young [83]. While we need a feasibility oracle; Plotkin et al. [70] use an optimization oracle. One of the major disadvantages of this approach, however, is that one needs to know the value (or an upper bound) of the width ρ. It is not clear how to compute ρ using the given oracle.

In Section 3.1.1, we followed a different approach of separating out the error as two components — an additive term, δ, and a term, ϵ arising out of the perturbed problem. As demonstrated with an example of the fractional Steiner forest problem (Section 3.1.2), this approach gives us an additional flexibility in reducing the running times. We shall see in Chapter 4 that this happens, in fact, for a large class of problems which we call *generalized hitting set problems*. Further, this approach does not require knowledge of ρ.

3.2 Optimization version of the GENERAL problem

We now consider an optimization version of the GENERAL problem defined as follows.

GENERAL (optimization): Given a non-empty convex set $P \subseteq \mathbb{R}^n$, non-negative continuous convex functions $f_i : P \to \mathbb{R}_+$ for $1 \le i \le m$, and non-negative continuous concave functions $g_i : P \to \mathbb{R}_+$ for $1 \le i \le m$, compute

$$\lambda^* = \min_{x \in P} \max_{1 \le i \le m} (f_i(x) - g_i(x)).$$

In order to solve the optimization version, we assume that we have the following optimization oracle.

Given $y \in \mathbb{R}_+^m$, the oracle finds an $x \in P$ that minimizes

$$\langle y, f(x) - g(x) \rangle. \tag{3.14}$$

The linear version of this optimization problem (with the corresponding oracle) was considered by Young [83] under the name of *generalized packing problem*. His approach was again similar to Plotkin et al. [70]: to reduce the problem to packing by adding width ρ to each entry which leads to an algorithm where the number of oracle calls varies as ρ^2.

3.2.1 A pseudo approximation algorithm

The algorithm that computes an $x \in P$ satisfying (3.15) is very similar to Algorithm 3.1 except that in line (6), we make a call to oracle (3.14) instead of oracle (3.1). Thus line (6) gets replaced by

(6)	$x_r := \arg\min_x \{\langle y, h(x) \rangle \mid x \in P\}$	{oracle call}

Theorem 3.2.1 *Given any error parameters $\epsilon, \delta \in (0,1)$, Algorithm 3.1 with the above modification computes an $x \in P$ such that*

$$\max_{i \in [m]} \hat{h}_i(x) = \max_{i \in [m]} \Big(\exp(-\epsilon) f_i(x) - \exp(\epsilon) g_i(x) \Big) \le \lambda^* + \delta. \tag{3.15}$$

where λ^* is the optimum value of the given problem. The algorithm makes $O(\hat{\rho}\epsilon^{-1}\delta^{-1}\log m)$ calls to the oracle (3.14) where the width $\hat{\rho}$ is given by (3.2).

Proof. The proof follows directly from Theorem 3.1.4. The optimum solution $x^* \in P$ satisfies $\max_{i \in [m]} h_i(x^*) = \lambda^*$, and hence $\langle \underline{y}, h(x^*) \rangle \le \lambda^*$ for $y \ge 0$. Since we try to minimize $\langle y, h(x) \rangle$, the point $x_r \in P$ returned by the oracle in any round r satisfies $\langle \underline{y}, h(x_r) \rangle \le \lambda^*$ for the current dual vector $y \ge 0$. This is same as the requirement in Theorem 3.1.4 with $\gamma = \lambda^*$. Therefore Theorem 3.1.4 implies that the output satisfies (3.15) and that the number of oracle calls is $O(\hat{\rho}\epsilon^{-1}\delta^{-1}\log m)$. ∎

Thus the output $x \in P$ minimizes the perturbed functions $\hat{h}_i(x)$ to within a small additive error of the optimum of the original problem. Thus it forms a *pseudo* approximation. Let

$$\hat{\lambda}^* = \min_{x \in P} \max_{i \in [m]} \hat{h}_i(x)$$

be the optimum value of the perturbed problem. Clearly, $\hat{\lambda}^* \le \lambda^*$. The difference between λ^* and $\hat{\lambda}^*$ can be regarded as a measure of the "stability" of the optimum. We have $\lambda^* \le \hat{\lambda}^* + 2\epsilon \cdot \overline{\rho}$, where $\overline{\rho} = \max_{x \in P} \max_{i \in [m]} \max\{f_i(x), g_i(x)\}$. This follows from the fact that $\exp(\epsilon) - 1 \le 2\epsilon$ for $\epsilon \in (0,1)$.

Computing a near-optimum dual solution

Now we prove that the modified algorithm, in fact, also computes a near-optimum *dual solution* along with a near-optimum primal solution. This is sometimes useful, for instance, when we are interested in solving the dual program. Consider the following well-known (strong) duality relation:

$$(\text{PRIMAL} \;\equiv)\;\; \min_{x\in P}\max_{y\geq 0}\,\langle \underline{y}, h(x)\rangle = \max_{y\geq 0}\min_{x\in P}\,\langle \underline{y}, h(x)\rangle \;\;(\equiv\; \text{DUAL}) \tag{3.16}$$

where $h : P \to \mathbb{R}^m$ is a vector of continuous convex functions. This duality is a generalization of the min-max theorem we saw in Chapter 2 in the context of the matrix game and follows from von Neumann's saddle-point theorem (see, e.g., Rockafellar [72], p.393). This duality was used by Grigoriadis & Khachiyan [33, 32] and Grigoriadis et al. [35] for the min-max and max-min resource sharing. In fact, it also generalizes the well-known LP-duality. In our setting, we let $h(x) = f(x) - g(x)$. We call the left-hand-side as the PRIMAL program and the right-hand-side as the DUAL program. For $x \in P$, define

$$\lambda(x) = \max_{y\geq 0}\langle \underline{y}, h(x)\rangle = \max_{i\in[m]} h_i(x)$$

and for $y \geq 0$, define

$$d(y) = \min_{x\in P}\langle \underline{y}, h(x)\rangle.$$

Thus the oracle (3.14), given y computes $d(y)$. Note that for any $x \in P$ and $y \geq 0$, we have

$$d(y) \leq \langle \underline{y}, h(x)\rangle \leq \max_{i\in[m]} h_i(x) = \lambda(x).$$

This relation is also known as *weak duality*. For the perturbed functions $\hat{h}_i(x) = (\exp(-\epsilon)f_i(x) - \exp(\epsilon)g_i(x))$, we define

$$\hat{\lambda}(x) = \max_{y\geq 0}\langle \underline{y}, \hat{h}(x)\rangle = \max_{i\in[m]} \hat{h}_i(x).$$

To compute a near-optimum dual solution, we extend the algorithm as follows. Let y_i^{r-1} be the value of the dual variable y_i at the end of round $r-1$. Through out the algorithm, we keep track of the dual solution y^{r-1} with the largest $d(y^{r-1})$ and output that as the dual solution $\tilde{y}$. Note that we do not have to make additional oracle calls to compute $d(y^{r-1})$ in any round since $d(y^{r-1}) = \langle y^{r-1}, \hat{h}(x_r)\rangle / \sum_{i=1}^m y_i^{r-1}$. The main theorem of this section is as follows.

Theorem 3.2.2 *There exists an algorithm for the* GENERAL *optimization problem that given error parameters $\epsilon, \delta \in (0,1)$, computes $x \in P$ and $y \geq 0$ such that*

$$\hat{\lambda}(x) \leq d(y) + \delta.$$

The algorithm makes at most $(1 + \hat{\rho}\epsilon^{-1}\delta^{-1}\ln m)$ calls to the oracle (3.14), where $\hat{\rho}$ is given by (3.2).

Proof. The proof is very similar to that of Theorem 3.1.1 given in Section 3.1.1. The profit and loss of the ith strategy in round r is again given by (3.3) and (3.4). The equations (3.6) and (3.7) continue to hold. However, now we have

$$\begin{aligned} \exp(\epsilon)\mathcal{P} - \exp(-\epsilon)\mathcal{L} &\leq \sum_{r=1}^{N} w_r \cdot \frac{\langle y^{r-1}, h_i(x_r)\rangle}{\sum_{i=1}^m y_i^{r-1}} \\ &\leq \sum_{r=1}^{N} w_r \cdot d(y^{r-1}) \end{aligned}$$

$$\leq \quad d(\tilde{y}) \cdot \sum_{r=1}^{N} w_r.$$

The remainder of the proof is identical to that of Theorem 3.1.1. We have

$$\hat{\lambda}(\tilde{x}) = \max_{i \in [m]} \hat{h}_i(\tilde{x}) \leq d(\tilde{y}) + \delta$$

and the number of calls is at most $1 + \hat{\rho}\epsilon^{-1}\delta^{-1} \ln m$. ∎

From Theorem 3.2.2, we have

$$\hat{\lambda}^* \leq \hat{\lambda}(x) \leq d(y) + \delta \leq \lambda^* + \delta.$$

Thus Theorem 3.2.2 subsumes Theorem 3.2.1. Theorem 3.2.2, in fact, yields an alternate proof of the strong duality (3.16) as explained below. Note first that $\lambda(x) \leq \hat{\lambda}(x) + 2\epsilon \cdot \overline{\rho}$ where $\overline{\rho} = \max_{x \in P} \max_{1 \leq i \leq m} \max\{f_i(x), g_i(x)\}$. Therefore, the algorithm computes $x \in P$ and $y \geq 0$ such that $\lambda(x) \leq d(y) + 2\epsilon \cdot \overline{\rho} + \delta$. Now by setting both $\epsilon, \delta \in (0,1)$ to arbitrarily small positive numbers, we can make the difference $\lambda(x) - d(y)$ arbitrarily small. Therefore $\min_{x \in P} \lambda(x) = \max_{y \geq 0} d(y)$ must hold. This is exactly the strong duality theorem (3.16).

Using approximate oracles

Similar to the feasibility version, the algorithm for the optimization version can also work with approximate oracles.

Theorem 3.2.3 *If we are given an approximate version of the oracle (3.14), which given any $y \geq 0$, computes an $x \in P$ such that $\langle \underline{y}, f(x) - g(x) \rangle \leq \sigma \cdot \min_{x' \in P} \langle \underline{y}, f(x') - g(x') \rangle + \gamma$ for some $\sigma, \gamma \in \mathbb{R}_+$, then the algorithm computes $x \in P$ and $y \geq 0$ such that*

$$\hat{\lambda}(x) \leq \sigma \cdot d(y) + \gamma + \delta.$$

The algorithm makes at most $(1 + \hat{\rho}\epsilon^{-1}\delta^{-1} \ln m)$ oracle calls, where $\hat{\rho}$ is given by (3.2).

Proof. The proof is identical to the one for Theorem 3.2.2 except that now we have $\exp(\epsilon)\mathcal{P} - \exp(-\epsilon)\mathcal{L} \leq (\sigma \cdot d(\tilde{y}) + \gamma) \sum_{r=1}^{N} w_r$. ∎

Thus a multiplicative error in the oracle translates into a multiplicative error in the final solution and an additive error in the oracle translates into an additive error in the final solution. Note that the algorithm does not need to know the values of σ and γ and that the number of oracle calls is independent of these values.

3.2.2 A true approximation algorithm

Similar to Theorem 3.1.8, we can prove the following theorem for the optimization version.

Theorem 3.2.4 *There exists an algorithm for the* GENERAL *optimization problem that given error parameter $\omega \in (0,1)$ computes $x \in P$ and $y \geq 0$ such that*

$$\lambda(x) \leq d(y) + \omega.$$

The algorithm makes at most $O(1 + \rho(1+\rho)\omega^{-2} \log m)$ calls to the oracle (3.14), where $\rho = \max_{x \in P} \max_{i \in [m]} |f_i(x) - g_i(x)|$.

The proof is similar to that of Theorem 3.1.8 and is omitted. Young [83] obtained a similar result for the linear version.

3.3 Computing the Held-Karp bound for TSP

Held & Karp [42, 43] proposed a very useful way of getting a lower bound on the length of the optimum tour for the traveling salesman problem. The traveling salesman problem is to find a minimum-cost tour that visits all the vertices and returns to the starting vertex in the given undirected graph $G = (V, E)$ with non-negative edge-costs $c_e \geq 0$. We present an $O(m^2\omega^{-2}\log^2 m)$ time algorithm for computing a $(1+\omega)$ approximation to the Held-Karp bound. For sparse graphs, this improves the randomized algorithm of Plotkin et al. [70] which has an expected running time of $O(n^4\omega^{-2}\log^6 n)$. Here m and n denote the number of edges and vertices in G.

Held & Karp [42, 43] formulate their lower bound in terms of 1-trees. They show that this lower bound can also be obtained by optimizing on the *sub-tour elimination polytope*. In this formulation, there are variables x_e for $e \in E$ which we also think of as capacities and constraints associated with the cuts; we wish to minimize the cost $\sum_{e\in E} c_e x_e$ subject to the constraint that the sum of capacities on the edges across any cut is at least two.

Here we work with an equivalent formulation based on packing branchings. Consider an assignment of capacities $x_e \geq 0$ to the edges $e \in E$ such that for any subset $\emptyset \neq S \subset V$, the total capacity across S is at least two: $\sum_{e\in\delta(S)} x_e \geq 2$. Now consider a directed graph $H = (V, A)$ where A is obtained from E by replacing each edge $e = (u, v) \in E$ by a pair of anti-parallel directed arcs $e_1 = (u, v)$ and $e_2 = (v, u)$. Assign capacities $x_{e_1} := x_e$ and $x_{e_2} := x_e$ to these directed arcs. Fix a vertex $r \in V$ as the root. Note that any subset $S \subset V$ containing r has at least two units of capacity "directed out" of S, i.e., $\sum_{a\in\delta(S)} x_a \geq 2$, where $\delta(S) \subset A$ denotes the set of arcs $(u, v) \in A$ such that $u \in S$ and $v \in V \setminus S$. An r-branching in a directed graph H is defined as a subset of edges that has a (directed) path from r to each vertex in V.

The fractional version of Edmond's edge-disjoint branchings theorem [17] implies that there exists a packing of r-branchings of total weight two that satisfies the arc capacities x_a. Thus with each r-branching T, we can associate a real number $x_T \geq 0$ such that $\sum_T x_T = 2$ and for any arc $a \in A$, we have $\sum_{T:a\in T} x_T \leq x_a$. On the other hand, if we have an assignment $\{x_T\}$ to r-branchings T such that $\sum_T x_T = 2$, then the capacities $x_e = \max\{\sum_{T:e_1\in T} x_T, \sum_{T:e_2\in T} x_T\}$ defined on the undirected edges $e \in E$ satisfy $\sum_{e\in\delta(S)} x_e \geq 2$ for any $\emptyset \neq S \subset V$. Therefore, we can formulate the Held-Karp bound as the following linear program.

$$\begin{array}{rcll} \min & & \sum_{e\in E} c_e x_e & \\ \text{s.t.} & & & \\ \sum_T x_T & = & 2 & \\ x_e & \geq & \sum_{T:e_1\in T} x_T & \forall e \in E \\ x_e & \geq & \sum_{T:e_2\in T} x_T & \forall e \in E \\ x_e, x_T & \geq & 0 & \forall e \in E, \forall T \end{array} \tag{3.17}$$

Feasibility version

Now suppose that we have to check if the optimum of the linear program (3.17) is at most some given $\alpha > 0$. We prove the following theorem.

Theorem 3.3.1 *There exists an algorithm that given any $\alpha > 0$ and an error parameter $\omega \in (0, 1)$, either finds a feasible solution to (3.17) with $\sum_{e\in E} c_e x_e \leq \alpha \cdot \exp(\omega)$, or proves that there is no feasible solution with $\sum_{e\in E} c_e x_e \leq \alpha$. The algorithm takes $O(m^2\omega^{-2}\log^2 m)$ time.*

Similar to that in the fractional Steiner forest problem, we define a polytope $P = P' \times X$ where $P' = \{(x_T : T \text{ is an } r\text{-branching}) \geq 0 \mid \sum_T x_T = 2\}$ and $X = \{(x_e : e \in E) \geq 0 \mid \sum_{e \in E} c_e x_e \leq \alpha\}$. For an $x \in P$, $e \in E$, and $l \in \{1, 2\}$, define

$$f_e^l(x) = \left(\frac{c_e}{\alpha}\right) \sum_{T : e_l \in T} x_T \quad \text{and} \quad g_e(x) = \left(\frac{c_e}{\alpha}\right) x_e.$$

The feasibility version of (3.17) can now be formulated as finding an $x \in P$ such that

$$f_e^l(x) - g_e(x) \leq 0 \qquad \forall e \in E, l = 1, 2.$$

Oracle

To solve the above formulation as the GENERAL problem we need an oracle that given $\{y_e^l \geq 0 \mid e \in E, l = 1, 2\}$ computes an $x \in P$ such that

$$\sum_{l=1}^{2} \sum_{e \in E} y_e^l \Big(f_e^l(x) - g_e(x)\Big) \leq 0.$$

We rewrite the left-hand-side as

$$\sum_T x_T \Big(\sum_{e_1 \in T} \frac{y_e^1 c_e}{\alpha} + \sum_{e_2 \in T} \frac{y_e^2 c_e}{\alpha}\Big) - \sum_{e \in E} \frac{c_e x_e}{\alpha} \cdot (y_e^1 + y_e^2)$$

and set $x_T = 2$ for a branching T that minimizes $\sum_{e_1 \in T} y_e^1 c_e/\alpha + \sum_{e_2 \in T} y_e^2 c_e/\alpha$. Such a branching, T, is the minimum cost branching with respect to costs $y_e^1 c_e/\alpha$ on the arcs $e_1 \in A$ and $y_e^2 c_e/\alpha$ on the arcs $e_2 \in A$ and can be found in $O(m \log n)$ time [78] where m and n denote the number of edges and vertices respectively. We also set $x_e = \alpha/c_e$ for an edge $e \in E$ for which $y_e^1 + y_e^2$ is maximum. Note that this assignment of variables minimizes the expression. If the value of the expression is great than zero, then, as before, the given instance is infeasible.

Now from Theorem 3.1.1, there is an algorithm that computes an $x \in P$ such that

$$\exp(-\epsilon) f_e^l(x) - \exp(\epsilon) g_e(x) \leq \delta \qquad \forall e \in E, l = 1, 2.$$

If we set $\delta = \epsilon/m$, we get

$$\max\{f_e^1(x), f_e^2(x)\} \leq \exp(\epsilon) \left(\exp(\epsilon) \cdot g_e(x) + \frac{\epsilon}{m}\right) \qquad \forall e \in E, l = 1, 2$$

which implies

$$\max\{\sum_{T : e_1 \in T} x_T, \sum_{T : e_2 \in T} x_T\} \leq \frac{\alpha}{c_e} \left(\exp(2\epsilon) g_e(x) + \frac{\epsilon \exp(\epsilon)}{m}\right).$$

Hence we can set $\tilde{x}_e$ to be the expression on the right-hand-side of the above inequality. Then

$$\begin{aligned} \sum_{e \in E} c_e \tilde{x}_e &\leq \sum_{e \in E} \alpha \left(\exp(2\epsilon) g_e(x) + \frac{\epsilon \exp(\epsilon)}{m}\right) \\ &\leq \alpha \Big(\exp(2\epsilon) + \epsilon \exp(\epsilon)\Big) && (\text{since } \sum_{e \in E} g_e(x) \leq 1) \\ &\leq \alpha \cdot \exp(3\epsilon). \end{aligned}$$

If we set $\epsilon = \omega/3$, we get a feasible solution to the linear program (3.17) with cost at most $\alpha \cdot \exp(\omega)$.

The total number of oracle calls is $O(\hat{\rho}\epsilon^{-1}\delta^{-1}\log m) = O(\hat{\rho}m\omega^{-2}\log m)$. Similar to that in Lemma 3.1.6, we can prove that any edge $e \in E$ with cost $c_e > \mathsf{OPT}$ does not have a positive x_e value in any optimum solution. Thus we can remove the edges $e \in E$ with $c_e > \alpha$ and make the width $\hat{\rho}$ constant. Thus the number of oracle calls becomes $O(m\omega^{-2}\log m)$. Since each oracle call takes $O(m\log n)$ time, we have Theorem 3.3.1.

The algorithm for the optimization version is similar to the one described for the fractional Steiner forest problem. Let OPT be the optimum value of (3.17) and C_{MST} be the cost of the minimum spanning tree in the given (undirected) graph G. It is easy to see that $C_{\mathrm{MST}} \le \mathsf{OPT} \le 2 \cdot C_{\mathrm{MST}}$. Since we have lower and upper bounds on OPT that are within a constant factor, the algorithm for the optimization version has the same asymptotic running time as that of the feasibility version.

3.4 Summary of key ideas developed in this chapter

1. To be able to ensure that the values of some m convex functions $\{f_i - g_i\}$ are below a certain threshold, zero in case of the feasibility version and λ^* in case of optimization version, it is enough to be able to minimize any given linear combination of these functions. If some function $f_i - g_i$ takes large accumulated value, the corresponding multiplier y_i will grow rapidly. Therefore, in the successive rounds, we will choose $x \in P$ with $f_i(x) - g_i(x)$ having small value. Thus these dual variables $\{y_i\}$ are able to guide us to a near-optimum solution.

2. We charged the additive error term $\ln m/\epsilon$ to the total weight accumulated W. In order to accumulate the weight rapidly, we choose each iterate x_r to the largest possible weight so that none of the dual variables changes by a factor more than $\exp(\epsilon)$. This idea of variable-size increments was first introduced by Garg & Könemann [31] in their algorithms for multicommodity flow problems.

Chapter 4

The PROVISIONING problem

In this chapter, we formulate an abstract problem called the PROVISIONING problem. This problem is, in fact, motivated by the fractional Steiner forest problem that we addressed in the last chapter. We reduce the PROVISIONING problem to the GENERAL feasibility problem. The algorithm for the PROVISIONING problem is therefore identical to the one for the GENERAL problem. Nevertheless, we abstract-out this problem since there is a natural way of reducing the width and obtaining a width-independent running time for this problem. We shall also see that our technique of separating the error term into an additive and a multiplicative components as developed in Chapter 3 gives better running times for the PROVISIONING problem as compared to those that would have been obtained by previously known methods.

We first motivate the PROVISIONING problem by considering a *group survivable network design problem*.

4.1 The group survivable network design problem

Consider the *survivable network design problem* (SNDP): given a (directed or undirected) graph $G = (V, E)$ with costs $c_e \geq 0$ and capacities $\mathcal{U}_e \geq 0$ on the edges $e \in E$ and t pairs of vertices (u_l, v_l) with demands $d_l > 0$ for $l \in [t]$, find the minimum-cost sub-graph of G that can support d_l units of flow between u_l and v_l for each $l \in [t]$ independently. We assume that the capacities are large enough to support d_l units of flow between each pair (u_l, v_l) in G one pair at a time. Note that the special case with $\mathcal{U}_e = 1$ and $d_l = 1$ corresponds to the Steiner forest problem considered in Chapter 3.

We define a generalization of SNDP which we call the *group*-SNDP. This is motivated by a problem faced by the network service provider catering to many groups of clients. In this problem, the input is the same as the SNDP except that we are also given k "groups" $\mathcal{C}_1, \ldots, \mathcal{C}_k \subseteq \{1, \ldots, t\}$ such that their union is $\{1, \ldots, t\}$. These groups may intersect each other. If the service provider serves all the demands in a group $\mathcal{C}_j$, he earns a unit profit. His objective is to maximize his total profit given a budget of $\alpha > 0$ on the total cost of installing the network.

Here we shall be interested in solving the *fractional* version of this problem. We assume that if each pair in the jth group $\mathcal{C}_j$ is able to send $\lambda_j \in [0, 1]$ fraction of its demand, then the service provider gets a profit of λ_j. Thus he would like to install *capacities* $0 \leq x_e \leq \mathcal{U}_e$ on the edges $e \in E$ such that

- the budget constraint is met: $\sum_{e \in E} c_e x_e \leq \alpha$,
- the profit $\sum_{j=1}^{k} \lambda_j$ is maximized.

Let p be a path between (u_l, v_l); we denote this by $p \sim l$. Let us define a polytope P^l to be the set of all flows between (u_l, v_l) of total value d_l that respect the edge-capacities., i.e., $P^l = \{(x_p : p \sim l) \geq 0 \mid \sum_{p \sim l} x_p = d_l, \sum_{p : e \in p, p \sim l} x_p \leq$

$\mathcal{U}_e$ for all $e \in E\}$. From our assumption, P^l is non-empty. Define $P = P^1 \times \cdots \times P^t$ to be the Cartesian product of these polytopes. Thus an $x = (x^1, \ldots, x^t) \in P$ is a vector of flows x^l between (u_l, v_l) that individually respect the edge-capacities. To route the flow given by a point $x \in P$, we need to install capacities

$$x_e = \max_{l \in [t]} \sum_{p: e \in p, p \sim l} x_p.$$

The profit λ_j of the jth group $\mathcal{C}_j$ is given by

$$\lambda_j = \min_{l \in \mathcal{C}_j} \frac{\sum_{p \sim l} x_p}{d_l}.$$

Now given an estimate $\mathcal{P} > 0$ on the total profit, we would like to check if the service provider can get a profit of $\mathcal{P}$. Thus we would like to find a solution $x \in P$ such that,

$$\sum_{e \in E} \max_{l \in [t]} \left(\left(\frac{c_e}{\alpha} \right) \sum_{p: e \in p, p \sim l} x_p \right) \leq 1 \leq \sum_{j=1}^{k} \min_{l \in \mathcal{C}_j} \left(\frac{\sum_{p \sim l} x_p}{d_l \cdot \mathcal{P}} \right). \tag{4.1}$$

In this chapter, we present an algorithm to solve exactly this kind of problems.

4.2 The PROVISIONING problem

The abstract PROVISIONING problem is given as follows.

PROVISIONING: Given non-empty convex sets $P^l \subseteq \mathbb{R}^{n_l}$ for $1 \leq l \leq t$, non-negative continuous convex functions $f_i^l : P^l \to \mathbb{R}_+$ for $1 \leq i \leq m$, and non-negative continuous concave functions $g_j^l : P^l \to \mathbb{R}_+$ for $1 \leq j \leq k$, either find $x^l \in P^l$ for $1 \leq l \leq t$ such that $$\sum_{i=1}^{m} \max_{1 \leq l \leq t} f_i^l(x^l) \leq 1 \leq \sum_{j=1}^{k} \min_{1 \leq l \leq t} g_j^l(x^l),$$ or prove that no such $x^l \in P^l, 1 \leq l \leq t$ exist.

The name PROVISIONING is motivated from the example of the survivable network design problem. Since the flows between different pairs share edge capacities, we would like to provision enough capacity to be able to route individual flows.

In order to solve the above problem, we assume that there is a subroutine to solve the following problem.

Given $y \in \mathbb{R}_+^m$, $z \in \mathbb{R}_+^k$, and $l \in \{1, \ldots, t\}$, the oracle finds an $x^l \in P^l$ that minimizes $$\langle y, f^l(x^l) \rangle - \langle z, g^l(x^l) \rangle. \tag{4.2}$$

To formulate the fractional group-SNDP (4.1) as a PROVISIONING problem, we set

$$f_e^l(x^l) = \left(\frac{c_e}{\alpha} \right) \sum_{p: e \in p, p \sim l} x_p,$$

and

$$g_j^l(x^l) = \begin{cases} \frac{\sum_{p \sim l} x_p}{d_l \cdot \mathcal{P}}, & \text{if } l \in \mathcal{C}_j; \\ 1, & \text{otherwise.} \end{cases}$$

Now let us see what form the oracle (4.2) takes for this problem. Given $\{y_e \geq 0 \mid e \in E\}$, $\{z_j \geq 0 \mid j \in [k]\}$, and $l \in [t]$, we want to find a flow-vector $x^l = (x_p : p \sim l) \in P^l$ that minimizes

$$\sum_{e \in E} y_e \left(\left(\frac{c_e}{\alpha} \right) \sum_{p: e \in p, p \sim l} x_p \right) - \sum_{j: l \in \mathcal{C}_j} z_j \left(\frac{\sum_{p \sim l} x_p}{d_l \cdot \mathcal{P}} \right) - \sum_{j: l \notin \mathcal{C}_j} z_j.$$

The last term $\sum_{j: i \notin \mathcal{C}_j} z_j$ is independent of $x^l \in P^l$. The first two terms can be rewritten as,

$$\sum_{p \sim l} \left(x_p \sum_{e \in p} \frac{y_e c_e}{\alpha} \right) - \left(\frac{\sum_{j: l \in \mathcal{C}_j} z_j}{d_l \cdot \mathcal{P}} \right) \sum_{p \sim l} x_p.$$

Observe that the above expression is same as the cost of the flow $x^l \in P^l$ under the cost function $y_e c_e / \alpha$ minus $(\sum_{j: l \in \mathcal{C}_j} z_j)/(d_l \cdot \mathcal{P})$ times the total amount of flow routed. Such a flow can be found, for instance, by doing a single min-cost flow computation in time $O(m(m + n \log n) \log n)$ [67, 1].

For the abstract PROVISIONING problem, we prove the following theorem.

Theorem 4.2.1 *There exists an algorithm for the* PROVISIONING *problem that given an error parameter* $\omega \in (0, 1)$*,*

- *either computes* $x^l \in P^l$ *for* $l \in [t]$ *such that*

$$\exp(-\omega) \cdot \sum_{i=1}^{m} \max_{1 \leq l \leq t} f_i^l(x^l) \leq 1 \leq \exp(\omega) \cdot \sum_{j=1}^{k} \max_{1 \leq l \leq t} g_j^l(x^l),$$

- *or proves that there does not exist* $x^l \in P^l, l \in [t]$ *such that* $\sum_i \max_l f_i^l(x^l) \leq 1 \leq \sum_j \min_l g_j^l(x^l)$.

The algorithm makes $O((1 + \rho)(m + k)t\omega^{-2} \log((m + k)t))$ *calls to the oracle (4.2), where*

$$\rho = \max_{l \in [t]} \max_{x^l \in P^l} \max\{\max_{i \in [m]} f_i^l(x^l), \max_{j \in [k]} g_j^l(x^l)\}$$

and takes additional time equal to the time to compute $f_i^l(x^l)$ *and* $g_j^l(x^l)$ *for some* $x^l \in P^l$ *for* $l \in [t]$ *between successive oracle calls.*

We remark again that our algorithm does not need an explicit representation of the either the convex set or the functions. All it needs is the oracle (4.2) and an oracle that computes $f^l(x^l)$ and $g^l(x^l)$ for any given $x^l \in P^l$ for $l \in [t]$.

4.2.1 Reduction to the GENERAL problem

We now show how to reduce the PROVISIONING problem to the GENERAL feasibility problem and obtain Theorem 4.2.1. We reformulate the PROVISIONING problem in the following equivalent form.

$$\begin{array}{rcll} f_i^l(x^l) - F_i & \leq & 0 & \text{for all } i \in [m], l \in [t] \\ G_j - g_j^l(x^l) & \leq & 0 & \text{for all } j \in [k], l \in [t] \\ x^l & \in & P^l & \text{for all } l \in [t] \\ \sum_{i=1}^{m} F_i = 1 & \text{and} & \sum_{j=1}^{k} G_j = 1. & \end{array} \tag{4.3}$$

Here the variables F_i are considered as upper bounds on $\max_{l\in[t]} f_i^l(x^l)$ and the variables G_j are considered as lower bounds on $\min_{l\in[t]} g_j^l(x^l)$.

Now this formulation can be thought of as an instance of the GENERAL feasibility problem if we let P to be the polytope defined by points $x^l \in P^l$ for $l \in [t]$ and real vectors F and G such that $\sum_i F_i = \sum_j G_j = 1$. More formally, we define $P = P^1 \times \cdots \times P^t \times \mathcal{F} \times \mathcal{G}$ as the Cartesian product, where $\mathcal{F} = \{(F_1, \ldots, F_m) \geq 0 \mid \sum_{i=1}^m F_i = 1\}$ and $\mathcal{G} = \{(G_1, \ldots, G_k) \geq 0 \mid \sum_{j=1}^k G_j = 1\}$ are m and k dimensional simplices respectively.

Oracle

To solve the above GENERAL problem using the algorithm described in Section 3.1, we need the following oracle. We have dual variables y_{il} and z_{jl} for $i \in [m]$, $j \in [k]$, and $l \in [t]$. Let $y_l = (y_{1l}, \ldots, y_{ml})$ and $z_l = (z_{1l}, \ldots, z_{kl})$. Then the oracle needs to find an $x = (x^1, \ldots, x^t, F, G) \in P$ such that

$$\sum_{l=1}^{t} \Big(\langle y_l, f^l(x^l) - F \rangle + \langle z_l, G - g^l(x^l) \rangle \Big) \leq 0,$$

which is equivalent to

$$\sum_{l=1}^{t} \Big(\langle y_l, f^l(x^l) \rangle - \langle z_l, g^l(x^l) \rangle \Big) - \left\langle \sum_{l=1}^{t} y_l, F \right\rangle + \left\langle \sum_{l=1}^{t} z_l, G \right\rangle \leq 0. \tag{4.4}$$

The second and third terms in the above expression are independent of x^l and so we should try to minimize the first term. We make a call to oracle (4.2) for each $l \in [t]$ with $y = y_l$ and $z = z_l$. Let $x^l \in P^l$ for $l \in [t]$ be the solutions returned by these t calls to the oracle (4.2). To minimize the expression, we need to choose F and G as

$$F_i = \begin{cases} 1, & \text{if } i = \arg\max_i \sum_l y_{il}; \\ 0, & \text{otherwise,} \end{cases}$$

and

$$G_j = \begin{cases} 1, & \text{if } j = \arg\min_j \sum_l z_{jl}; \\ 0, & \text{otherwise.} \end{cases}$$

If the left-hand-side of (4.4) with these values of $\{x^l\}$, F and G is greater than zero, we get that the given instance of the PROVISIONING problem is infeasible. Therefore, we assume that in every oracle call, we find an $x \in P$ that satisfies (4.4).

Now let $\epsilon, \delta \in (0, 1)$ be the error parameters used in the GENERAL algorithm. We set $\epsilon = \omega/3$ and $\delta = \epsilon/2(m+k)$ where ω is the given error parameter for the PROVISIONING problem. Thus from Theorem 3.1.1, the algorithm finds an $x \in P$ such that,

$$\begin{aligned}
\exp(-\epsilon) \cdot f_i^l(x^l) - \exp(\epsilon) \cdot F_i &\leq \frac{\epsilon}{2(m+k)} && \text{for } i \in [m], l \in [t], \\
\exp(-\epsilon) \cdot G_j - \exp(\epsilon) \cdot g_j^l(x^l) &\leq \frac{\epsilon}{2(m+k)} && \text{for } j \in [k], l \in [t], \\
\sum_{i=1}^{m} F_i &= 1, \\
\sum_{j=1}^{k} G_j &= 1.
\end{aligned}$$

Thus we have,

$$\begin{aligned}
\sum_{i=1}^{m} \max_{1\le l\le t} f_i^l(x^l) &\le \exp(\epsilon)\sum_{i=1}^{m}\left(\exp(\epsilon)\cdot F_i + \frac{\epsilon}{2(m+k)}\right)\\
&\le \exp(2\epsilon)\cdot(1+\epsilon/2)\\
&\le \exp(3\epsilon),
\end{aligned}$$

and

$$\begin{aligned}
\sum_{j=1}^{k} \min_{1\le l\le t} g_j^l(x^l) &\ge \exp(-\epsilon)\sum_{j=1}^{k}\left(\exp(-\epsilon)\cdot G_j - \frac{\epsilon}{2(m+k)}\right)\\
&\ge \exp(-2\epsilon)\cdot(1-\epsilon/2)\\
&\ge \exp(-3\epsilon).
\end{aligned}$$

This is similar to the analysis of the fractional Steiner forest algorithm in Section 3.1.2. Thus the solution $\{x^l\}$ satisfies the condition in Theorem 4.2.1 since we have $\epsilon = \omega/3$.

Now let us estimate the width $\hat{\rho}$ of the instance (4.3) of the GENERAL problem. Note that

$$\max_{i\in[m]}\max_{l\in[t]}\max_{x^l\in P^l} |\exp(-\epsilon)f_i^l(x^l) - \exp(\epsilon)F_i| = O(1+\rho),$$

and

$$\max_{j\in[k]}\max_{l\in[t]}\max_{x^l\in P^l} |\exp(-\epsilon)G_j - \exp(\epsilon)g_j^l(x^l)| = O(1+\rho),$$

where ρ is as given in Theorem 4.2.1. Note that one call to the oracle (3.1) is implemented by making t calls to the oracle (4.2). Since $\epsilon = \omega/3$ and $\delta = \epsilon/2(m+k)$, the total number of calls to the oracle (4.2), from Theorem 3.1.1, is

$$t(1+(1+\rho)\epsilon^{-1}\delta^{-1}\ln((m+k)t)) = O((1+\rho)(m+k)t\omega^{-2}\log((m+k)t)).$$

This proves Theorem 4.2.1.

4.2.2 A technique for reducing width

We can use ideas similar to those in Section 3.1.2, to reduce the width of the given instance. The number of oracle calls in Theorem 4.2.1 depends linearly on the width which is defined as

$$\rho = \max_{l\in[t]}\max_{x^l\in P^l}\max\{\max_{i\in[m]} f_i^l(x^l), \max_{j\in[k]} g_j^l(x^l)\}.$$

Here we demonstrate that for some problems we can make the width constant. This is done by modifying the instance so that the optimum is not affected. This, however, has to be done with caution since we also need the appropriate oracle for the new instance.

For the group-SNDP

Let us consider the group-SNDP as formulated in (4.1) again. Note that the first term in the definition of ρ can be bounded by

$$\max_{l\in[t]} \max_{x^l\in P^l} \max_{e\in E} \frac{c_e \sum_{p:e\in p, p\sim l} x_p}{\alpha} \leq \max_{l\in[t]} \max_{x^l\in P^l} \max_{e\in E} \frac{c_e \mathcal{U}_e}{\alpha}.$$

The above inequality holds since $\sum_{p:e\in p,p\sim l} x_p \leq \mathcal{U}_e$. Now since α is a budget on the total installation cost, we will not install a capacity of more than α/c_e on an edge $e \in E$. Thus we can safely set $\mathcal{U}_e = \min\{\mathcal{U}_e, \alpha/c_e\}$ and so the above term is at most one. The second term in the definition of ρ for this instance is

$$\max\left\{1, \max_{l\in[t]} \max_{x^l\in P^l} \frac{\sum_{p\sim l} x_p}{d_l \cdot \mathcal{P}}\right\}.$$

Note that by definition of P^l we have $\sum_{p\sim l} x_p \leq d_l$. Thus if $\mathcal{P} \geq 1$, the above expression is at most one. Let us now consider the case when the service provider is looking for a profit of $\mathcal{P} < 1$. In this case, there is no *need* to send a flow of more than $d_l \cdot \mathcal{P}$ between any pair. Therefore, without loss of generality, we can restrict $x^l \in P^l$ to satisfy $\sum_{p\sim l} x_p \leq d_l \cdot \mathcal{P}$.

Putting the above arguments together, we can make the width constant in case of the group-SNDP. Note that we can continue to use the same oracle for the modified instance.

For the PROVISIONING problem

We now extend the above argument to the PROVISIONING problem. Note that any feasible solution $x^l \in P^l$ for $l \in [t]$ satisfies $f_i^l(x^l) \leq 1$. Thus without loss of generality, we can "truncate" the underlying convex sets as $\hat{P}^l = \{x^l \in P^l \mid f_i^l(x^l) \leq 1, \text{ for all } i \in [m]\}$ for $l \in [t]$. Since the functions f_i^l are convex, the new sets $\hat{P}^l$ continue to be convex.

Furthermore since we are interested in $x^l \in P^l$ such that $\sum_j \min_l g_j^l(x^l) \geq 1$, there is no need to consider values of g_j^l more than one. Therefore, we can "truncate" the functions g_j^l to have values at most 1. Thus we define new functions $\hat{g}_j^l : \hat{P}^l \to \mathbb{R}_+$ as

$$\hat{g}_j^l(x^l) = \min\{g_j^l(x^l), 1\}.$$

Since the functions g_j^l are concave and we are taking minimum, the new functions $\hat{g}_j^l$ continue to be concave. Note here that unlike in the case of functions f_i^l, we cannot truncate the underlying convex sets because of two reasons:

- the sets $\{x^l \in P^l \mid g_j^l(x^l) \leq 1\}$ may not be convex,
- we may still need $x^l \in P^l$ with $g_j^l(x^l) > 1$ for satisfying $\sum_i \max_l f_i^l(x^l) \leq 1$.

Note that the original instance is feasible if and only if the modified instance is feasible. Furthermore, a feasible solution to the modified instance is also feasible for the original instance. Note also that the width of the modified instance is at most one. Thus if we have the oracle required for the modified instance, the total number of calls to this oracle made by the algorithm is $O((m+k)t\omega^{-2}\log((m+k)t))$.

The group Steiner network design problem revisited

To compute an $\exp(\epsilon)$ approximate solution to the feasibility version of the group-SNDP problem, we need to make $O^*((m+k)t\omega^{-2})$ oracle calls. Each oracle call can be implemented by doing a min-cost flow computation in time $O^*(m^2)$. Thus the overall running time is $O^*(m^2(m+k)t\omega^{-2})$. For a pair $l \in [t]$, let $\Lambda_l \in [0,1]$ be the maximum fraction of its demand d_l that can be routed with total installation cost at most the budget α. We can compute Λ_l for each $l \in [t]$ by doing a constrained maximum flow computation [2] in time $O^*(m^2 \log M)$, where M is an upper bound on the largest element in the input. Note that for any group $C_j, j \in [k]$, the maximum profit that the jth group can contribute is $\min_{l \in C_j} \Lambda_l$. Thus if OPT denotes the maximum profit, we have

$$\max_{j\in[k]} \min_{l \in C_j} \Lambda_l \leq \mathsf{OPT} \leq \sum_{j \in [k]} \min_{l \in C_j} \Lambda_l \leq k \cdot \max_{j \in [k]} \min_{l \in C_j} \Lambda_l .$$

Thus we have lower and upper bounds on the optimum profit that are within a factor of k. Therefore, in a manner similar to our procedure for the fractional Steiner forest problem in Section 3.1.2, the optimization version can be solved in time $O^*(m^2 t(\log M + (m+k)(\log\log k + \omega^{-2})))$ time.

The group-SNDP problem can also be solved by the GENERAL-LP approach of Plotkin et al. [70] and Young [83]. However again as in the fractional Steiner forest problem, this approach takes a factor $\Omega(m+k)$ more time. The group-SNDP problem can also be formulated as a linear program with $(mt+k)$ variables and constraints. Since the variables in an optimum solution can take values that are exponentially small in n, we have $L = \Omega(n)$ and hence Vaidya's algorithm [79] takes time $O((mt+k)^{2.5} n)$.

4.3 Generalized hitting set problem

We now consider a generic framework for a class of combinatorial optimization problems which we call the *generalized hitting set* problem. We are given a finite set X with non-negative costs $c : X \to \mathbb{R}_+$. We are also given t families $\mathcal{C}_1, \ldots, \mathcal{C}_t \subseteq 2^X$ of subsets of X. The generalized hitting set problem is to find a subset $S \subseteq X$ of minimum-cost such that it contains, as a subset, at least one set from each family $\mathcal{C}_l$. Formally, the objective is to find $S \subseteq X$ such that for each $l \in [t]$, there exists $S_l \in \mathcal{C}_l$ such that $S_l \subseteq S$ and the cost $\sum_{e \in S} c_e$ is minimized. This framework captures a wide variety of interesting combinatorial optimization problems including:

- Hitting set problem: given a set X with non-negative costs on the elements and a family of subsets $S_l, l \in [t]$, find a minimum-cost subset that intersects (hits) each S_l. Here we let $\mathcal{C}_l$ to be the family of singleton sets corresponding to the elements of S_l.

- Survivable network design problem: given a (directed or undirected) graph with non-negative edge-costs and t pairs of vertices with integer demands $d_l \geq 1$ for $l \in [t]$, find a minimum-cost subgraph that has at least d_l edge-disjoint paths between the lth pair of terminals for each $l \in [t]$. Here we let X be the set of edges and $\mathcal{C}_l$ be the family of the set of d_l edge-disjoint paths between the lth pair.

- Multi-cut problem: given a (directed or undirected) graph with non-negative edge-costs and t pairs of vertices, find a minimum-cost subgraph that when removed disconnects each pair of terminals. Again X is the set of edges and $\mathcal{C}_l$ is the family of cuts separating the lth pair.

- Feedback arc-set problem: given a directed graph with non-negative edge-costs, find a minimum-cost subgraph that when removed leaves no directed cycles. Here X is the set of edges and $\mathcal{C}_v$ for a vertex v is the family of edge-sets that intersect every cycle containing v.

An integer programming formulation for the generalized hitting set problem involves finding an assignment $x_e \in \{0,1\}$ for $e \in X$ such that $x_e = 1$ if and only if $e \in S$ and $x_T \in \{0,1\}$ for $T \in \mathcal{C}_l$ for each l where $x_T = 1$ implies that the set T

is a subset of S. The LP-relaxation of this integer program is given below.

$$\begin{array}{rcll} \min & & \sum_{e\in X} c_e x_e & \\ \text{s.t.} & & & \\ \sum_{T\in\mathcal{C}_l} x_T & \geq & 1 & \forall l\in[t] \\ x_e & \geq & \sum_{\substack{T:e\in T,\\ T\in\mathcal{C}_l}} x_T & \forall l\in[t], \forall e\in E \\ x_e, x_T & \geq & 0 & \forall e\in E, \forall l\in[t], \forall T\in\mathcal{C}_l \end{array} \tag{4.5}$$

Feasibility version

We now argue that the feasibility version of the above linear program can be solved as a special case of the PROVISIONING problem. Let $P^l = \{(x^l_T : T\in\mathcal{C}_l) \geq 0 \mid \sum_{T\in\mathcal{C}_l} x^l_T = 1\}$ denote the set of "fractional" feasible solutions for the lth family $\mathcal{C}_l$. Now for $e\in E$ and $l\in[t]$, define functions $f^l_e : P^l \to \mathbb{R}_+$ as $f^l_e(x^l) = c_e \sum_{T:e\in T, T\in\mathcal{C}_l} x^l_T$ where $x^l = (x^l_T : T\in\mathcal{C}_l)$. Thus setting $x_e = 1/c_e \cdot \max_{l\in[t]} f^l_e(x^l)$, we get a feasible solution $\{x_e \mid e\in E\}$ to the linear program (4.5). Therefore the linear program (4.5) is equivalent to computing:

$$\min_{x\in P} \sum_{e\in X} \max_{l\in[t]} f^l_e(x^l),$$

where $x = (x^1, \ldots, x^t)$ and $P = P^1 \times \cdots \times P^t$. In the feasibility version, given $\alpha > 0$, the objective is to check if there is a solution $x\in P$ with total cost at most α. Thus we would like to find an $x\in P$ such that

$$\sum_{e\in X} \max_{l\in[t]} \frac{f^l_e(x^l)}{\alpha} \leq 1.$$

To think of this problem as a special case of the PROVISIONING problem, we let $k = 1$ and introduce an artificial constant function $g^l(x^l) = 1$.

Oracle

The oracle needed to solve this feasibility version as the PROVISIONING problem is: given $l\in[t]$ and $y : X \to \mathbb{R}_+$, find an $x^l \in P^l$ such that

$$\sum_{e\in X} y_e \left(\frac{f^l_e(x^l)}{\alpha}\right) = \sum_{e\in X} y_e \left(\frac{c_e}{\alpha}\right) \sum_{T:e\in T, T\in\mathcal{C}_l} x^l_T = \sum_{T\in\mathcal{C}_l} x^l_T \left(\sum_{e\in T} \frac{y_e c_e}{\alpha}\right)$$

is minimized. Since $\sum_{T\in\mathcal{C}_l} x^l_T = 1$, the above expression will be minimized if we set $x^l_T = 1$ for that $T\in\mathcal{C}_l$ for which $\sum_{e\in T} y_e c_e/\alpha$ is minimum. Thus the oracle reduces to finding the minimum-cost set in $\mathcal{C}_l$ under any given non-negative cost function.

Let us see what form this oracle takes for the case of hitting set, survivable network design, multi-cut, and feedback arc-set problems. For the hitting set problem, the oracle just finds a minimum-cost element in the lth subset. In the survivable network design problem, the oracle is to find the minimum-cost collection of d_l edge-disjoint paths between the lth pair. Such a collection can be found by doing a min-cost flow computation or by doing d_l shortest-path computations. For the multi-cut problem, the oracle is to find the minimum cut separating the lth pair. For the feedback arc-set problem, the oracle is to find a minimum-cost edge-set that intersects every cycle through given vertex v. Such an edge-set can be found by doing a single minimum-cost directed s-t cut.

Given this oracle for the hitting set problem, using Theorem 4.2.1, we can find an $x \in P$ such that

$$\sum_{e \in X} \max_{l \in [t]} f_e^l(x^l) \leq \alpha \cdot \exp(\omega),$$

by making $O(\rho m t \omega^{-2} \log(mt))$ oracle calls where $m = |X|$ is the number of elements and t is the number of families of subsets.

Reducing the width

We now show that the techniques of reducing the width given in Sections 3.1.2 and 4.2.2 are fairly general and can be used for the generalized hitting set problem. It is easy to see that Lemma 3.1.6 extends to the case of the hitting set problem.

Lemma 4.3.1 *Let* $\mathsf{OPT} = \sum_e c_e x_e^*$ *be the value of an optimum solution* $\{x_e^*\}$ *to the linear program (4.5). Then for any* e *with* $c_e > \mathsf{OPT}$*, we have,* $x_e^* = 0$.

The proof is very similar to that of Lemma 3.1.6 and is omitted. Thus while checking if there exists a solution with cost at most α, we can "remove" all elements $e \in X$ with $c_e > \alpha$. Note that we can continue using the original oracle which finds the minimum cost set by assigning the elements with $c_e > \alpha$ an infinite cost to ensure that they are never included. This effectively removes the high-cost elements and so we have $\rho \leq 1$. Hence the total number of oracle calls needed to get an $\exp(\omega)$ approximation is $O(mt\omega^{-2} \log(mt))$ and we have the following theorem.

Theorem 4.3.2 *There exists an algorithm for the generalized hitting set problem that given an error parameter* $\omega \in (0, 1)$*,*

- *either computes a feasible solution to (4.5) with cost at most* $\alpha \cdot \exp(\omega)$*,*
- *or proves that there does not exist a feasible solution to (4.5) with cost at most* α*.*

The algorithm makes $O(mt\omega^{-2} \log(mt))$ *calls to the oracle that given* $l \in [t]$ *and costs* $\{y_e \geq 0 \mid e \in X\}$*, finds a set* $S \in \mathcal{C}_l$ *whose cost,* $\sum_{e \in S} y_e$*, is minimum.*

Optimization version

The reduction of the optimization version to the feasibility version is identical to the one for the fractional Steiner forest problem discussed in Section 3.1.2. Let OPT denote the optimum value of the linear program (4.5). We would like to find a feasible solution $\{x_e \mid e \in X\}$ and $\{x_T^l \mid T \in \cup_{l \in [t]} \mathcal{C}_l\}$ for (4.5) such that the total cost $\sum_{e \in X} c_e x_e \leq \exp(\omega) \cdot \mathsf{OPT}$. To get an estimate of OPT, we use the following lemma.

Lemma 4.3.3 *For* $l \in [t]$*, let* Λ_l *denote the cost of the minimum cost set in* $\mathcal{C}_l$ *under the cost function* $\{c_e\}$*. Then,* $\max_{l \in [t]} \Lambda_l \leq \mathsf{OPT} \leq \sum_{l=1}^t \Lambda_l \leq t \cdot \max_{l \in [t]} \Lambda_l$.

Proof. For any $l \in [t]$, the cost of covering the sets in $\mathcal{C}_l$ to the total extent one in any fractional solution is at least Λ_l. On the other hand, union of the minimum cost sets in $\mathcal{C}_l$ for all $l \in [t]$ forms a feasible fractional solution and has cost at most $\sum_{l \in [t]} \Lambda_l$. ∎

Thus we have lower and upper bounds on OPT that are within a factor t of each other. Therefore in a manner similar to one described in Section 3.1.2, we can compute an $\exp(\omega)$ approximation to the generalized hitting set problem by making $O(mt \log(mt)(\log\log t + \omega^{-2})$ oracle calls. Solving this problem via GENERAL-LP approach of Plotkin et al. [70] and Young [83] again takes a factor $\Omega(|X|)$ more time.

Chapter 5

The MIXED PACKING and COVERING problem

In this chapter, we consider an abstract problem called MIXED PACKING and COVERING. Like the PROVISIONING problem, even this problem can be thought of as a special case of the GENERAL problem. However, for the MIXED problem, we present an algorithm with running time independent of the width. A linear version of the MIXED problem was first considered by Young [84]. We generalize his results to arbitrary convex/concave functions over arbitrary convex sets. Later in the chapter, we show that the min-max and max-min resource sharing problems considered by Grigoriadis & Khachiyan [32, 33], Grigoriadis et al. [35] and Jansen & Zhang [48] can be thought of as special cases of the MIXED problem.

5.1 The MIXED problem

The abstract MIXED PACKING and COVERING problem is defined as follows.

> MIXED: Given a non-empty convex set $P \subseteq \mathbb{R}^n$, non-negative continuous convex functions $f_i : P \to \mathbb{R}_+$ for $1 \leq i \leq m$, and non-negative continuous concave functions $g_i : P \to \mathbb{R}_+$ for $1 \leq i \leq k$, either find an $x \in P$ such that
>
> $$\max_{1 \leq i \leq m} f_i(x) \leq \min_{1 \leq j \leq k} g_j(x),$$
>
> or prove that no such $x \in P$ exists.

This problem is called MIXED because it contains both a packing component $\max_i f_i(x)$ and a covering component $\min_j g_j(x)$. Recall that for a non-zero vector $v \geq 0$, the vector $\underline{v}$ denotes v after normalization, i.e., $\underline{v} = v / \sum_i v_i$. Thus $\underline{v}$ is a vector with 1-norm one. In order to solve the MIXED problem, we assume that there is a subroutine to solve the following problem.

> Given $y \in \mathbb{R}^m_+$ and $z \in \mathbb{R}^k_+$, the oracle either finds an $x \in P$ such that
>
> $$\langle \underline{y}, f(x) \rangle \leq \langle \underline{z}, g(x) \rangle, \tag{5.1}$$
>
> or proves that no such $x \in P$ exists.

5.1.1 Previous work

Young [84] first considered linear programming problems of finding an $x \in \mathbb{R}^n_+$ such that

$$Ax \leq b, Cx \geq d$$

where $A \in \mathbb{R}_+^{m\times n}$, $b \in \mathbb{R}_+^m$, $C \in \mathbb{R}_+^{k\times n}$, and $d \in \mathbb{R}_+^k$ are matrices and vectors with non-negative entries. The constraints $Ax \leq b$ are called *packing* constraints while the constraints $Cx \geq d$ are called *covering* constraints. We call the combined problem as MIXED-LP. As explained in Young [84], we can assume, without loss of generality, that each entry in b and d is 1. This is achieved as follows. We first remove any constraint of the form $(Ax)_i \leq 0$ since they force each x_j with $A_{ij} > 0$ to zero and do not restrict x_j with $A_{ij} = 0$ at all. We also remove any constraint of the form $(Cx)_i \geq 0$ since they do not constrain any x_j at all. After doing this, we divide the ith packing constraint by b_i, i.e., replace A_{ij} by A_{ij}/b_i and b_i by 1. We also divide the ith covering constraint by d_i, i.e., replace C_{ij} by C_{ij}/d_i and d_i by 1.

The MIXED problem generalizes the MIXED-LP problem. To see this, let $f_i(x) = (Ax)_i$ for $i \in [m]$, $g_j(x) = (Cx)_j$ for $j \in [k]$, $P = \mathbb{R}_+^n$, and introduce two constant functions $f_0(x) = g_0(x) = 1$ for $x \in P$. Now it is easy to see that $\max_{0\leq i\leq m} f_i(x) \leq \min_{0\leq j\leq k} g_j(x)$ for $x \in P$ if and only if $Ax \leq \mathbf{1} \leq Cx$ for $x \in \mathbb{R}_+^n$.

Young [84] assumes the following oracle to solve the MIXED-LP. Given $y, z \geq 0$, find $p \in [n]$ such that $(\underline{y}^\top A)_p \leq (\underline{z}^\top C)_p$ holds. He uses dual vectors y, z which are partial derivatives of an exponential penalty function. Note that this oracle is the same as the oracle (5.1) for the MIXED-LP problem. This is because for $f_i(x) = (Ax)_i$ and $g_j(x) = (Cx)_j$, we have that $\langle \underline{y}, f(x)\rangle \leq \langle \underline{z}, g(x)\rangle$ is equivalent to $\underline{y}^\top Ax \leq \underline{z}^\top Cx$. Finding such an $x \in \mathbb{R}_+^n$ is equivalent to finding $p \in [n]$ such that $(\underline{y}^\top A)_p \leq (\underline{z}^\top C)_p$. He also points out that one can in fact work with an approximate oracle that finds $p \in [n]$ such that $(y^\top A)_p \leq (1+\epsilon)(z^\top C)_p$. He used this oracle to improve the running time by employing an idea of round-robin increments first used by Fleischer [20] in the context of multicommodity flows. We generalize Young's results to arbitrary convex/concave functions over arbitrary convex sets (provided we have appropriate oracles), thereby solving one of his open questions.

5.1.2 Reduction to the GENERAL feasibility problem

The MIXED problem can be reduced to the GENERAL feasibility problem described in Section 3.1. Given an instance of the MIXED problem with m functions f_i and k functions g_j, we define an instance of the GENERAL feasibility problem with mk functions as follows. Find an $x \in P$ (if it exists) such that

$$\max_{i\in[m], j\in[k]} (f_i(x) - g_j(x)) \leq 0.$$

Clearly an $x \in P$ is a feasible solution for the MIXED problem if and only if it is a feasible solution for the above instance of the GENERAL feasibility problem.

Oracle

In Algorithm 3.1 for the GENERAL feasibility problem, we associate dual variables $y_{ij} \geq 0$ with functions $(f_i(x) - g_j(x))$. We initialize the dual variables y_{ij} to 1 and in round r, we update them as

$$y_{ij} := y_{ij} \cdot \exp(\epsilon w_r(\exp(-\epsilon) f_i(x_r) - \exp(\epsilon) g_j(x_r))).$$

We can equivalently let $y_{ij} = Y_i Z_j$ where Y_i and Z_j are initialized to 1 and updated in round r as

$$Y_i := Y_i \cdot \exp(\epsilon w_r \exp(-\epsilon) f_i(x_r)),$$
$$Z_j := Z_j \cdot \exp(-\epsilon w_r \exp(\epsilon) g_j(x_r)).$$

Algorithm MIXED (width-dependent)

Input : Convex set $P \subseteq \mathbb{R}^n$, functions $f : P \to \mathbb{R}^m_+$, $g : P \to \mathbb{R}^k_+$ and $\epsilon, \delta \in (0, 1)$
Output : "infeasible" or $\tilde{x} \in P$

(1)	For $i \in [m]$ do $y_i := 1$	{initialize the dual variables}
	For $j \in [k]$ do $z_j := 1$	
(2)	Define $\hat{h}_{ij}(x) := \exp(-\epsilon) f_i(x) - \exp(\epsilon) g_j(x)$	
(3)	$r := 0$	{initialize the round number}
(4)	$W := 0$	{initialize the weight}
	Repeat	
(5)	$r := r + 1$	{round r begins}
(6)	Find $x_r \in P$ such that $\langle \underline{y}, f(x_r) \rangle \leq \langle \underline{z}, g(x_r) \rangle$	{oracle call}
	If there is no such $x_r \in P$, then output "infeasible"	
(7)	$w_r := 1/ \max_{i \in [m], j \in [k]} \lvert \hat{h}_{ij}(x_r) \rvert$	{note that $w_r \geq 1/\hat{\rho}$}
(8)	$W := W + w_r$	{increment weight}
(9)	For $i \in [m]$ do $y_i := y_i \cdot \exp(\epsilon w_r \exp(-\epsilon) f_i(x_r))$	{update dual variables}
	For $j \in [k]$ do $z_j := z_j \cdot \exp(-\epsilon w_r \exp(\epsilon) g_j(x_r))$	
(10)	Until $W \geq \epsilon^{-1} \delta^{-1} \ln m$	{stop if enough weight gathered}
(11)	Output $\tilde{x} = (\sum_{s=1}^{r} w_s x_s)/(\sum_{s=1}^{r} w_s)$	

Algorithm 5.1: A width-dependent algorithm for the MIXED problem

Now in round r, given the new dual variables $\{Y_i\}$ and $\{Z_j\}$, the oracle (line (6) in Algorithm 3.1) needs to find an $x \in P$ such that

$$\sum_{i=1}^{m} \sum_{j=1}^{k} Y_i Z_j (f_i(x) - g_j(x)) \leq 0.$$

This is equivalent to

$$\sum_{i=1}^{m} Y_i f_i(x) \cdot \sum_{j=1}^{k} Z_j - \sum_{j=1}^{k} Z_j g_j(x) \cdot \sum_{i=1}^{m} Y_i \leq 0$$

which, in turn, is the same as

$$\frac{\sum_{i=1}^{m} Y_i f_i(x)}{\sum_{i=1}^{m} Y_i} \leq \frac{\sum_{j=1}^{k} Z_j g_j(x)}{\sum_{j=1}^{k} Z_j},$$

which is the same requirement as that of oracle (5.1). This completes the reduction. Algorithm 5.1 corresponds to Algorithm 3.1 for the instance of the GENERAL problem obtained from the given MIXED problem.

Theorem 3.1.1 yields the following corollary.

Corollary 5.1.1 *There exists an algorithm for the* MIXED *problem that given error parameters $\epsilon, \delta \in (0, 1)$,*

- *either computes an $x \in P$ such that*

$$\exp(-\epsilon) \max_{i \in [m]} f_i(x) \leq \exp(\epsilon) \min_{j \in [k]} g_i(x) + \delta,$$

- *or proves that there does not exist an $x \in P$ such that* $\max_{i \in [m]} f_i(x) \leq \min_{j \in [k]} g_j(x)$.

The algorithm makes at most $(1 + \rho \epsilon^{-1} \delta^{-1} \ln(mk))$ calls to the oracle (5.1), where

$$\rho = \max_{x \in P} \max_{i \in [m], j \in [k]} \lvert \exp(-\epsilon) f_i(x) - \exp(\epsilon) g_j(x) \rvert$$

and takes additional time equal to the time to compute $f(x)$ and $g(x)$ for some $x \in P$ between successive oracle calls.

Like in the case of the GENERAL problem, the algorithm also works with approximate oracles. The following corollary follows from Theorem 3.1.4.

Corollary 5.1.2 *If the oracle, given $y \in \mathbb{R}_+^m$ and $z \in \mathbb{R}_+^k$, returns an $x \in P$ such that $\langle \underline{y}, f(x)\rangle \leq \langle \underline{z}, g(x)\rangle + \gamma$ for some $\sigma, \gamma \in \mathbb{R}$, then the algorithm computes $x \in P$ such that*

$$\exp(-\epsilon) \max_{i \in [m]} f_i(x) \leq \exp(\epsilon) \min_{j \in [k]} g_j(x) + \gamma + \delta.$$

The algorithm makes at most $(1 + \rho\epsilon^{-1}\delta^{-1} \ln(mk))$ oracle calls.

5.1.3 A width-independent algorithm for the MIXED problem

A width-independent algorithm for the MIXED-LP problem was given by Young [84]. In this section, we generalize his algorithm to the MIXED problem. The main theorem of the section is as follows.

Theorem 5.1.3 *There exists an algorithm for the* MIXED *problem that given an error parameter $\omega \in (0, 1)$,*

- *either computes an $x \in P$ such that*
$$\max_{i \in [m]} f_i(x) \leq \exp(\omega) \cdot \min_{j \in [k]} g_j(x),$$
- *or proves that there does not exist an $x \in P$ such that $\max_{i \in [m]} f_i(x) \leq \min_{j \in [k]} g_j(x)$.*

The algorithm makes $O((m + k)\omega^{-2} \log(m + k))$ calls to the oracle (5.1) and takes additional time equal to the time to compute $f(x)$ and $g(x)$ for some $x \in P$ between successive oracle calls.

We prove that Algorithm 5.2 satisfies the claims in Theorem 5.1.3. The procedures **Initialize**, **Update**, and **Stop-condition** used in lines (1), (6), and (7) respectively are defined in Algorithm 5.3. We set $\epsilon = \omega/3$ where $\omega \in (0, 1)$ is the given error parameter. We associate dual variables y_i and z_j with functions f_i and g_j respectively; these dual variables are initialized to 1. The algorithm goes in rounds and makes one call to the oracle (5.1) (line (4)) in each round. Note that the dual variables y and z are never zero in the algorithm.

If in some round r, there is no $x_r \in P$ such that $\langle \underline{y}, f(x_r)\rangle \leq \langle \underline{z}, g(x_r)\rangle$, then the algorithm outputs that the given instance is infeasible. This is justified as follows. If $x \in P$ is a feasible solution for the given instance of the MIXED problem, then for any $y, z \geq 0$, we have $\langle \underline{y}, f(x)\rangle \leq \max_{i \in [m]} f_i(x) \leq \min_{j \in [k]} g_j(x) \leq \langle \underline{z}, g(x)\rangle$. Thus if the oracle (5.1) fails to produce a desired $x \in P$ for some $y, z \geq 0$, we can declare that the given instance is infeasible. In the rest of the section, we assume that the oracle always produces a desired $x \in P$.

Note that this algorithm differs from the one given for the GENERAL problem in Section 3.1 in three ways:

- Choosing the step-size w_r. To choose the step-size w_r (see line (5)), we consider all functions f_i but consider only those g_j for which the dual variables z_j are sufficiently large. Furthermore, we do not use the perturbed functions anymore.

- Updating the dual variables. The algorithm updates the dual variable z_j only if it is sufficiently large (see procedure **Update** in Algorithm 5.3).

- The stopping condition. The algorithm stops (see **Stop-condition** in Algorithm 5.3), as soon as *some* function f_i is "packed" to a large extent or *all* functions g_j are "covered" to a large extent, i.e., when the extent of the covering or packing exceeds $\Gamma = O(\epsilon^{-2} \log(m + k))$.

Algorithm MIXED (width-independent)

Input : Convex set $P \subseteq \mathbb{R}^n$, functions $f : P \to \mathbb{R}_+^m$, $g : P \to \mathbb{R}_+^k$, and $\omega \in (0, 1)$
Output : "infeasible" or $\tilde{x} \in P$

(1)	**Initialize**	
(2)	$r := 0$	$\{$initialize the round number$\}$
	Repeat	
(3)	$\quad r := r + 1$	$\{$round r begins$\}$
(4)	$\quad$ Find $x_r \in P$ such that $\langle \underline{y}, f(x_r) \rangle \leq \langle \underline{z}, g(x_r) \rangle$	$\{$oracle call$\}$
	$\quad$ If there is no such x_r, then output "infeasible"	
(5)	$\quad w_r := 1/\max\{\max_{i\in[m]} f_i(x_r),$ $\max_{j:z_j>\exp(-\epsilon\Gamma)} g_j(x_r)\}$	$\{$pick x_r to an extent $w_r\}$
(6)	$\quad$ **Update**(w_r, f, g, x_r)	
(7)	Until **Stop-condition** is True	$\{$round r ends$\}$
(8)	Output $\tilde{x} = (\sum_{s=1}^r w_s x_s)/(\sum_{s=1}^r w_s)$	

Algorithm 5.2: A width-independent algorithm for the MIXED problem. The procedures **Initialize**, **Update**, and **Stop-condition** are defined in Algorithm 5.3.

Initialize:	
Set $\epsilon = \omega/3$	
For $i \in [m]$ do $y_i := 1$	
For $j \in [k]$ do $z_j := 1$	$\{$initialize the dual variables$\}$
$\Gamma := 2\epsilon^{-2}(\ln m + \exp(2\epsilon)\ln k)$	$\{$see **Stop-condition**$\}$
Update(w, f, g, x)**:**	
For $i \in [m]$ do $y_i := y_i \cdot \exp(\epsilon w f_i(x))$	
For $j \in [k]$ do	
$\quad$ If $z_j > \exp(-\epsilon\Gamma)$, then $z_j := z_j \cdot \exp(-\epsilon w g_j(x))$	$\{$dual variables updated$\}$
Stop-condition:	
True, if $\max_{i\in[m]} \sum_{s=1}^r w_s f_i(x_s) \geq \Gamma$	$\{$stop if some f_i is packed "enough"$\}$
True, if $\min_{j\in[k]} \sum_{s=1}^r w_s g_j(x_s) \geq \Gamma$	$\{$stop if all g_j are covered "enough"$\}$
False, otherwise	

Algorithm 5.3: Procedures for Algorithm 5.2

These modifications are needed to make the number of oracle calls width-independent. They also ensure that we will be able to "charge" the additive error term to either the packing or the covering term, thus obtaining a multiplicative guarantee.

The algorithm outputs a convex combination of the solutions found in all the rounds. Again notice that, similar to the algorithms presented in Chapters 3 and 4, this algorithm just needs an oracle (5.1) and an oracle that computes $f(x)$ and $g(x)$ for given $x \in P$.

Analysis

Now consider the following instance of the on-line prediction problem. Suppose that there is an agent who has m strategies corresponding to each of the m functions $f_i(x)$. The on-line prediction goes in rounds $r = 1 \ldots, N$ where N denotes the total number of rounds in Algorithm 5.2. Suppose that in round r, the ith strategy earns a profit of $w_r f_i(x_r)$ and incurs a loss of zero. Assume that the agent chooses a distribution $\{\underline{y}^{r-1}\}$ in round r, where y_i^{r-1} denotes the value of y_i at the end of round $r - 1$. Note that in the procedure **Update** (Algorithm 5.3), the dual variables $\{\underline{y}_i\}$ are updated exactly according to the exponential rule given in (2.2). Since the loss is zero, from Corollary 2.1.2, we get

$$\max_{i\in[m]}\sum_{r=1}^{N} w_r f_i(x_r) \le \exp(\epsilon)\sum_{r=1}^{N} w_r\langle \underline{y}^{r-1}, f(x_r)\rangle + \left(\frac{\ln m}{\epsilon}\right). \tag{5.2}$$

Now consider another instance of the on-line prediction problem. Suppose now that the agent has k strategies corresponding to each of the k functions $g_j(x)$. The on-line prediction goes in rounds $r = 1\ldots, N$ where in round r, the jth strategy incurs a loss of $w_r g_j(x_r)$ if $z_j^{r-1} > \exp(-\epsilon\Gamma)$ and zero otherwise (see the way the dual variables $\{z_j\}$ are updated in procedure **Update** (Algorithm 5.3)). Assume also that the jth strategy earns zero profit. Now the agent chooses a distribution $\{\underline{z}^{r-1}\}$ in round r, where z_j^{r-1} denotes the value of z_j at the end of round $r-1$. From the way we have defined the loss, the dual variables $\{\underline{z}_j\}$ are updated exactly according to the exponential rule given in (2.2). Since now the profit is zero, Corollary 2.1.3 implies

$$\min_{j\in[k]}\sum_{r=1}^{N} w_r g_j(x_r) \ge \exp(-\epsilon)\sum_{r=1}^{N} w_r\langle \underline{z}^{r-1}, g(x_r)\rangle - \left(\frac{\ln k}{\epsilon}\right). \tag{5.3}$$

Since in any round r, the oracle returns $x_r \in P$ such that $\langle \underline{y}^{r-1}, f(x_r)\rangle \le \langle \underline{z}^{r-1}, g(x_r)\rangle$ we have,

$$\sum_{r=1}^{N} w_r\langle \underline{y}^{r-1}, f(x_r)\rangle \le \sum_{r=1}^{N} w_r\langle \underline{z}^{r-1}, g(x_r)\rangle. \tag{5.4}$$

Now from equations (5.2), (5.3), and (5.4), we get,

$$\max_{i\in[m]}\sum_{r=1}^{N} w_r f_i(x_r) \le \exp(2\epsilon)\cdot\min_{j\in[k]}\sum_{r=1}^{N} w_r g_j(x_r) + \left(\frac{(\ln m + \exp(2\epsilon)\ln k)}{\epsilon}\right). \tag{5.5}$$

Let $\Gamma = 2\epsilon^{-2}(\ln m + \exp(2\epsilon)\ln k)$ as defined in the procedure **Initialize** (Algorithm 5.3). Thus the second term on the right-hand-side of the inequality (5.5) is $\epsilon\Gamma/2$. Now, depending on the **Stop-condition** due to which the algorithm exits the repeat-loop at line (7), we have either

$$\max_{i\in[m]}\sum_{r=1}^{N} w_r f_i(x_r) \ge \Gamma,$$

or

$$\min_{j\in[k]}\sum_{r=1}^{N} w_r g_j(x_r) \ge \Gamma.$$

In the first case, we "charge" the error term $\epsilon\Gamma/2$ to the functions f_i while in the second case, we "charge" it to the functions g_j. Thus in these two cases, we have

$$\begin{aligned}\max_{i\in[m]}\sum_{r=1}^{N} w_r f_i(x_r) - \frac{\epsilon\Gamma}{2} &\ge (1-\epsilon/2)\max_{i\in[m]}\sum_{r=1}^{N} w_r f_i(x_r)\\ &\ge \exp(-\epsilon)\max_{i\in[m]}\sum_{r=1}^{N} w_r f_i(x_r),\end{aligned}$$

or

$$\exp(2\epsilon)\cdot\min_{j\in[k]}\sum_{r=1}^{N} w_r g_j(x_r) + \frac{\epsilon\Gamma}{2} \le (1+\epsilon)\exp(2\epsilon)\cdot\min_{j\in[k]}\sum_{r=1}^{N} w_r g_j(x_r)$$

$$\leq \quad \exp(3\epsilon) \cdot \min_{j \in [k]} \sum_{r=1}^{N} w_r g_j(x_r).$$

Therefore, from inequality (5.5), we have

$$\max_{i \in [m]} \sum_{r=1}^{N} w_r f_i(x_r) \leq \exp(3\epsilon) \cdot \min_{j \in [k]} \sum_{r=1}^{N} w_r g_j(x_r). \tag{5.6}$$

Now let $\tilde{x}$ denote the output of the algorithm given by

$$\tilde{x} = \frac{\sum_{r=1}^{N} w_r x_r}{\sum_{r=1}^{N} w_r}.$$

Since P is a convex set, we have $\tilde{x} \in P$. From the convexity of f_i and the concavity of g_j, we have

$$\begin{aligned}
\max_{i \in [m]} f_i(\tilde{x}) \quad &\leq \quad \max_{i \in [m]} \left(\frac{\sum_{r=1}^{N} w_r f_i(x_r)}{\sum_{r=1}^{N} w_r} \right) \\
&\leq \quad \exp(3\epsilon) \cdot \min_{j \in [k]} \left(\frac{\sum_{r=1}^{N} w_r g_j(x_r)}{\sum_{r=1}^{N} w_r} \right) \\
&\leq \quad \exp(3\epsilon) \cdot \min_{j \in [k]} g_j(\tilde{x}).
\end{aligned}$$

Since $\epsilon = \omega/3$, the output satisfies the condition in Theorem 5.1.3.

Remark 5.1.4 *Note that we have used the convexity of the functions f_i and the concavity of the functions g_j only in this last step. Thus the inequality (5.6) holds for arbitrary non-negative functions f_i and g_j. This observation will have a non-trivial consequence in Chapter 6.*

The following lemma bounds the number of oracle calls that the algorithm makes.

Lemma 5.1.5 *Algorithm 5.2 makes $O((m+k)\omega^{-2}\log(m+k))$ calls to the oracle (5.1).*

Proof. In any round r, let $A = \{j \in [k] \mid \sum_{s=1}^{r} w_s g_j(x_s) < \Gamma\}$ denote the set of functions g_j that are not yet "covered" to an extent Γ, let $F_i = \sum_{s=1}^{r} w_s f_i(x_s)$ and $G_j = \sum_{s=1}^{r} w_s g_j(x_s)$. Note that the values of F_i and G_j never decrease as the algorithm progresses. Now let us consider carefully the way we have defined the weight w_r in line (5) and the way we update the duals in procedure **Update** in Algorithm 5.3. If $j \in A$, we have $G_j < \Gamma$. Therefore the current value of z_j is $\exp(-\epsilon G_j) < \exp(-\epsilon\Gamma)$. Therefore, in any round r,

- either A decreases in size by at least one,
- or at least one of $\sum_{i=1}^{m} F_i$ and $\sum_{j \in A} G_j$ increases by at least 1.

We exit the repeat loop as soon as either $\max_{i \in [m]} F_i \geq \Gamma$ or A becomes empty (see **Stop-condition** in Algorithm 5.3). Therefore, the maximum value that any F_i can take is $\Gamma + 1$. We have added 1 to Γ, because each F_i increases in steps of at most 1. Thus the maximum value that $\sum_{i=1}^{m} F_i$ can take is $m(\Gamma + 1)$. Now, as soon as $G_j \geq \Gamma$, the index j is removed from A. Thus for any j, the value of G_j can rise at most up to $\Gamma + 1$ while $j \in A$. Therefore the total number of rounds is at most $(m+k)(\Gamma+1) = O((m+k)\omega^{-2}\log(m+k))$. Since the number of oracle calls is exactly equal to the number of rounds and $\epsilon = \omega/3$, the proof is complete. ∎

Thus the proof of Theorem 5.1.3 is also complete.

Using approximate oracles

Note that there is a lot of flexibility in the way one can implement oracle (5.1). We can either minimize $\langle y, f(x)\rangle - \langle z, g(x)\rangle$ or $\langle y, f(x)\rangle / \langle z, g(x)\rangle$. Furthermore we can also work with approximate oracles.

Theorem 5.1.6 *If the oracle, given $y \in \mathbb{R}^m_+$ and $z \in \mathbb{R}^k_+$, returns an $x \in P$ such that $\langle \underline{y}, f(x)\rangle \le \sigma \cdot \langle \underline{z}, g(x)\rangle$ for some $\sigma \in \mathbb{R}_+$, then the algorithm computes an $x \in P$ such that*

$$\max_{i\in[m]} f_i(x) \le \sigma \cdot \exp(\omega) \cdot \min_{j\in[k]} g_j(x).$$

The algorithm makes $O((m+k)\omega^{-2}\log(m+k))$ oracle calls.

Proof sketch. In case of approximate oracles, inequality (5.4) gets replaced by

$$\sum_{r=1}^{N} w_r \langle \underline{y}^{r-1}, f(x_r)\rangle \le \sum_{r=1}^{N} w_r \Big(\sigma \cdot \langle \underline{z}^{r-1}, g(x_r)\rangle + \gamma\Big).$$

If we were to continue the same analysis with this inequality, we would get that the number of oracle calls is proportional to σ. We can, however, use the following idea to make the running time independent of σ. In each round r, the oracle returns $x_r \in P$ such that $\langle \underline{y}, f(x_r)\rangle \le \sigma \cdot \langle \underline{z}, g(x_r)\rangle$. Let $\sigma_r = \langle \underline{y}, f(x_r)\rangle / \langle \underline{z}, g(x_r)\rangle$. Note that $\sigma_r \le \sigma$ and that we can compute the value of σ_r. Define functions $g^r_j(x) = \sigma_r \cdot g_j(x)$ for $j \in [k]$. We use the functions g^r_j to define w_r and to update the dual variables in round r. The rest of the algorithm is same as Algorithm 5.2. Now, in the analysis, inequality (5.4) gets replaced by

$$\sum_{r=1}^{N} w_r \langle \underline{y}^{r-1}, f(x_r)\rangle \le \sum_{r=1}^{N} w_r \langle \underline{z}^{r-1}, g^r(x_r)\rangle.$$

Continuing further, the inequality (5.6) gets replaced by

$$\max_{i\in[m]} \sum_{r=1}^{N} w_r f_i(x_r) \le \exp(3\epsilon) \cdot \min_{j\in[k]} \sum_{r=1}^{N} w_r g^r_j(x_r).$$

The fact that $g^r_j(x_r) \le \sigma \cdot g_j(x_r)$ implies

$$\max_{i\in[m]} \sum_{r=1}^{N} w_r f_i(x_r) \le \sigma \cdot \exp(3\epsilon) \cdot \min_{j\in[k]} \sum_{r=1}^{N} w_r g^r_j(x_r).$$

The rest of the analysis is similar to the one given for Theorem 5.1.3. ∎

Thus again, a multiplicative error in the oracle translates into a multiplicative error in the final output. Note again that the algorithm does not need to know the value of σ and that the number of oracle calls is also independent of this value.

5.2 Some special cases of the MIXED problem

There are many interesting corollaries of Theorem 5.1.3. Let $P \subseteq \mathbb{R}^n_+$ be a convex set. Let $f : P \to \mathbb{R}^m_+$ denote a vector of non-negative continuous convex functions and let $g : P \to \mathbb{R}^k_+$ denote a vector of non-negative continuous concave functions. Let $\omega \in (0,1)$ be the given error parameter.

Corollary 5.2.1 *There is an algorithm that either finds an $x \in P$ such that*

$$\exp(-\omega) \cdot \max_{i \in [m]} f_i(x) \leq 1 \leq \exp(\omega) \cdot \min_{j \in [k]} g_j(x),$$

or proves that there is no $x \in P$ such that $\max_{i \in [m]} f_i(x) \leq 1 \leq \min_{j \in [k]} g_j(x)$ holds. The algorithm makes $O((m + k)\omega^{-2} \log(m + k))$ calls to the oracle that given $y_0 \in \mathbb{R}_+, y \in \mathbb{R}_+^m$ and $z_0 \in \mathbb{R}_+, z \in \mathbb{R}_+^k$ such that $y_0 + \langle \mathbf{1}, y \rangle = z_0 + \langle \mathbf{1}, z \rangle = 1$, finds an $x \in P$ such that $y_0 + \langle y, f(x) \rangle \leq z_0 + \langle z, g(x) \rangle$.

Proof. Similar to that given in Section 5.1.1, we introduce constant functions f_0 and g_0 such that $f_0(x) = g_0(x) = 1$ for all $x \in P$. We then use the algorithm for the MIXED problem on functions $f_0, \ldots, f_m$ and $g_0, \ldots, g_k$. It is easy to see that the output $\tilde{x} \in P$ of the algorithm satisfies the desired inequality. ∎

5.2.1 Min-max and max-min resource sharing problems

Comparison with previous work

We next show that the min-max and max-min resource sharing problems considered by Grigoriadis & Khachiyan [32, 33], Grigoriadis et al. [35], Jansen & Zhang [48], Jansen & Porkolab [47], and Jansen [46] can be thought of as special cases of the MIXED problem. The algorithms we present use exactly the same oracles as used in the above papers. While their algorithms are based on a logarithmic potential function, our algorithms yield similar results with exponential potential functions, thereby solving an open question in Grigoriadis et al. [35] of obtaining a width-independent algorithm based on the exponential potential functions for the max-min resource sharing. Our algorithms are substantially simpler to describe, analyze, and implement. For example, to update the dual solution (price vector) in each round, our algorithms need $O(m)$ operations while their algorithms need to compute a root of a degree-m polynomial. However our results are slightly weaker since our algorithms use $O(m\omega^{-2} \log m)$ oracle calls while Grigoriadis et al. [35] and Jansen & Zhang [48] give algorithms for max-min and min-max resource sharing respectively that need $O(m(\omega^{-2} + \log m))$ oracle calls. Here m denotes the number of convex or concave functions involved. Our algorithms extend *without any modification* to the case of approximate oracles (block-solvers) with arbitrary large approximation ratios while Jansen & Zhang [48] and Jansen [45] respectively have to do substantial extra work to obtain min-max and max-min resource sharing algorithms that work with approximate oracles. Furthermore, their algorithms, unlike ours, do not automatically generate near-optimum dual solutions without any further increase in the running time. For instance, Jansen & Zhang [48] mention that they need $O(m\omega^{-3} \log(m\omega^{-1}))$ oracle calls to obtain near-optimum dual solutions. We solve one of their open questions by reducing this number to $O(m\omega^{-2} \log m)$.

The linear versions of the min-max and max-min resource sharing problems called packing and covering problems were considered by Plotkin et al. [70], Luby & Nisan [64], Garg & Könemann [31], and Könemann [57]. Their algorithms have running times that are similar to the ones presented in this section. Bienstock & Iyengar [9, 8] recently presented an algorithm that computes a $(1 + \omega)$ approximation to packing problems by solving $O(\omega^{-1}\sqrt{Kn \log m})$ separable convex quadratic programs, where K is the maximum number of non-zeros per row and n and m are the number of variables and constraints respectively. This is the first algorithm that needs $O^*(\omega^{-1})$ iterations.

Corollary 5.2.2 (Min-max resource sharing or PACKING) *There is an algorithm that finds an $x \in P$ such that*

$$\max_{i \in [m]} f_i(x) \leq \exp(\omega) \cdot \lambda^*$$

where $\lambda^* = \min_{x \in P} \max_{i \in [m]} f_i(x)$ is the optimum value. The algorithm makes $O(m\omega^{-2} \log m)$ calls to an oracle that given $y \in \mathbb{R}_+^m$, finds an $x \in P$ that minimizes $\langle y, f(x) \rangle$.

Proof. Define $g_0(x) = \lambda^*$ for all $x \in P$, where $\lambda^* = \min_{x \in P} \max_{i \in [m]} f_i(x)$. Consider running Algorithm 5.2 for the mixed problem on the functions $f_1, \ldots, f_m$ and g_0. Note that we do not know the value of $g_0 = \lambda^*$. We argue, however, that the algorithm does not really need to know the value of λ^* and can still be made to work. In round r of the algorithm, the value of the dual vector is y^{r-1}. Let $x = \arg\min_{x \in P} \langle \underline{y}^{r-1}, f(x) \rangle$. Clearly we have, $\langle \underline{y}^{r-1}, f(x) \rangle \le \langle \underline{y}^{r-1}, f(x^*) \rangle \le \lambda^* = g_0(x)$ where $x^* = \arg\min_{x \in P} \max_{i \in [m]} f_i(x)$ is the optimum solution. Thus the oracle which, given $y \in \mathbb{R}^m_+$, finds $x = \arg\min_{x \in P} \langle y, f(x) \rangle$ can be used in line (4) in any round r. Note that in line (5) of the algorithm, we have

$$w_r = \frac{1}{\max\{\max_{i \in [m]} f_i(x_r), \lambda^*\}} = \frac{1}{\max_{i \in [m]} f_i(x_r)}$$

since, for any $x \in P$, we have $\max_{i \in [m]} f_i(x) \ge \lambda^*$. Thus the function g_0 does not affect the extent w_r to which we pick x_r. Thus after $O(m\Gamma)$ oracle calls the **Stop-condition** corresponding to the functions f_i will be true. From Theorem 5.1.3, this algorithm finds an $x \in P$ with $\max_{i \in [m]} f_i(x) \le \exp(\omega) \cdot \lambda^*$ by making $O(m\omega^{-2} \log m)$ oracle calls. ∎

The algorithm for the min-max resource sharing or PACKING can be modified to obtain a near-optimum dual solution as well. This is similar to Theorem 3.2.2 for the optimization version of the GENERAL problem. For $x \in P$, let $\lambda(x) = \max_{i \in [m]} f_i(x)$ be the *primal* objective function and for $y \ge 0$, let $d(y) = \min_{x \in P} \langle \underline{y}, f(x) \rangle$ be the *dual* objective function. Clearly, for any $x \in P$ and $y \ge 0$, we have $d(y) \le \lambda^* \le \lambda(x)$ where λ^* is the optimum value of the primal.

Corollary 5.2.3 *There is an algorithm that finds an $x \in P$ and $y \ge 0$ such that*

$$\lambda(x) \le \exp(\omega) \cdot d(y).$$

The algorithm makes $O(m\omega^{-2} \log m)$ calls to the oracle that given $y \in \mathbb{R}^m_+$, finds an $x \in P$ that minimizes $\langle y, f(x) \rangle$.

Proof. The proof is similar to that of Corollary 5.2.2. Consider the algorithm described in the proof of Corollary 5.2.2. In this algorithm, we had set $g_0(x) = \lambda^*$ for $x \in P$. In the proof, we used only two properties of λ^*: (1) that in round r, we can compute an $x_r \in P$ such that $\langle \underline{y}^{r-1}, f(x_r) \rangle \le \lambda^*$, and (2) that $\max_{i \in [m]} f_i(x_r) \ge \lambda^*$. Now in this algorithm, let

$$y = y^{p-1} \text{ where } p = \arg \max_{1 \le r \le N} d(y^{r-1}). \tag{5.7}$$

Let us modify the algorithm to let $g_0(x) = d(y)$ for $x \in P$. Note that $d(y)$ satisfies the two desired properties: (1) we can compute $x_r = \arg\min_{x \in P} \langle \underline{y}^{r-1}, f(x) \rangle$ such that $\langle \underline{y}^{r-1}, f(x_r) \rangle \le d(y)$, and (2) $\max_{i \in [m]} f_i(x_r) = \lambda(x_r) \ge d(y)$. The remaining proof is identical to that of Corollary 5.2.2. ∎

The above corollary is useful particularly when we are also interested in solving the dual program of finding $y \ge 0$ that maximizes $d(y)$. We shall see an application of this corollary in Section 7.3.1.

Similar to the MIXED problem, we can work with approximate oracles here as well.

Corollary 5.2.4 *Given an approximate oracle that when given $y \in \mathbb{R}^m_+$, finds an $x \in P$ that minimizes $\langle y, f(x) \rangle$ within a multiplicative factor of $\sigma \ge 1$, there is an algorithm that finds an $x \in P$ and $y \ge 0$ such that*

$$\lambda(x) \le \sigma \cdot \exp(\omega) \cdot d(y).$$

The algorithm makes $O(m\omega^{-2} \log m)$ oracle calls.

The proof of the above corollary follows by combining Theorem 5.1.6 and the proof of Corollary 5.2.3.

We can prove analogous results for max-min resource sharing.

Corollary 5.2.5 (Max-min resource sharing or COVERING) *There is an algorithm that finds an $x \in P$ and $z \geq 0$ such that*

$$\lambda(x) \geq \exp(-\omega) \cdot d(z)$$

where $\lambda(x) = \min_{j \in [k]} g_j(x)$ and $d(z) = \max_{x \in P} \langle \underline{z}, g(x) \rangle$. The algorithm makes $O(k\omega^{-2} \log k)$ calls to the oracle that given $z \in \mathbb{R}_+^k$, finds an $x \in P$ that maximizes $\langle z, g(x) \rangle$.

Corollary 5.2.6 *Given an approximate oracle that when given $z \in \mathbb{R}_+^k$, finds an $x \in P$ that maximizes $\langle z, g(x) \rangle$ within a multiplicative factor of $\sigma \geq 1$, there is an algorithm that finds an $x \in P$ and $z \geq 0$ such that*

$$\lambda(x) \geq 1/\sigma \cdot \exp(-\omega) \cdot d(z).$$

The algorithm makes $O(k\omega^{-2} \log k)$ oracle calls.

The proofs of both these corollaries are similar to their packing counterparts and are omitted.

5.3 Summary of key ideas developed in this chapter

1. In the MIXED problem, the objective is to find an $x \in P$ such that $\max_{i \in [m]} f_i(x)$ is "small" and $\min_{j \in [k]} g_j(x)$ is "large". Therefore we introduce dual variables y_i that increase rapidly with increase in f_i and dual variables z_j that decrease rapidly with increase in g_j. The algorithm stops when *some* y_i is large enough or *all* z_j are small enough. Doing this, we are able to charge the additive error term to either $\max_{i \in [m]} f_i(x)$ or $\min_{j \in [k]} g_j(x)$.

2. To reach the stopping condition as soon as possible, we choose the step-size w_r in each round so that either at least one y_i increases by a multiplicative factor of $\exp(\epsilon)$ or at least one z_j that is "large enough" decreases by a factor of $\exp(-\epsilon)$. This is achieved by using variable-size increments of Garg and Könemann [31]. With this, we are able to argue that after a width-independent number of rounds, the stopping condition is achieved.

Chapter 6

The SUBSTITUTE MIXED problem

In this chapter, we formulate a new problem called the SUBSTITUTE MIXED problem. This problem can be thought of as a special case of the MIXED problem in which we allow the functions f_i to be non-convex and the functions g_j to be non-concave. We motivate this problem by first discussing the maximum multicommodity flow problem.

6.1 The maximum multicommodity flow problem

Consider the *maximum multicommodity flow* problem defined as follows. We are given a (directed or undirected) graph $G = (V, E)$ with capacities $c_e \geq 0$ on the edges $e \in E$. We are also given t pairs of vertices (u_l, v_l) for $l \in [t]$. The problem is to find the maximum flow between these pairs that can be routed simultaneously so that the capacity constraints are satisfied. We formulate this problem as a linear program. Let us associate variables $x_p \geq 0$ indicating the flow on a path p between u_l and v_l for some $l \in [t]$. Let $\mathcal{P}$ denote the set of all paths between all t pairs. We would like to find $x = (x_p : p \in \mathcal{P})$ such that $\sum_{p \in \mathcal{P}} x_p$ is maximized subject to the constraint $\sum_{p: e \in p} x_p \leq c_e$ for each edge $e \in E$.

$$\begin{array}{lrcll} \max & \sum_{p \in \mathcal{P}} x_p & & & \\ \text{s.t.} & \sum_{p: e \in p} x_p & \leq & c_e & \text{for } e \in E \\ & x_p & \geq & 0 & \text{for } p \in \mathcal{P} \end{array} \tag{6.1}$$

The maximum multicommodity flow problem can be "equivalently" formulated as follows. We send a total of unit flow between all pairs put together so that the maximum *relative congestion* of any edge is minimized. The relative congestion of an edge is defined as the ratio of the total flow through the edge to the capacity of that edge. Thus we would like to minimize $\max_{e \in E} \frac{\sum_{p: e \in p} x_p}{c_e}$ subject to the constraint that $\sum_{p \in \mathcal{P}} x_p = 1$.

$$\begin{array}{lrcll} \min & \max_{e \in E} \frac{\sum_{p: e \in p} x_p}{c_e} & & & \\ \text{s.t.} & \sum_{p \in \mathcal{P}} x_p & = & 1 & \\ & x_p & \geq & 0 & \text{for } p \in \mathcal{P} \end{array} \tag{6.2}$$

Observe that if $x = (x_p : p \in \mathcal{P})$ is a feasible solution with value $\lambda = \sum_{p \in \mathcal{P}} x_p$ to the linear program (6.1), then x/λ is a feasible solution to the program (6.2) with value $\max_{e \in E} \frac{\sum_{p: e \in p} x_p}{c_e} \leq 1/\lambda$. On the other hand, if $x = (x_p : p \in \mathcal{P})$ is a feasible solution with value $\mu = \max_{e \in E} \frac{\sum_{p: e \in p} x_p}{c_e}$ to the program (6.2), then x/μ is a feasible solution to the linear program (6.1) with value $\sum_{e \in E} x_p/\mu = 1/\mu$. It is now easy to see that λ^* is the optimum value of the linear program (6.1) if and only if $1/\lambda^*$ is the optimum value of the program (6.2). Thus the two formulations are equivalent up to scaling of the variables.

The latter formulation (6.2) can be seen as an instance the PACKING problem considered in Corollary 5.2.2. For this, let $P = \{(x_p : p \in \mathcal{P}) \geq 0 \mid \sum_{p \in \mathcal{P}} x_p = 1\}$ be the set of all flow vectors sending a total of unit flow. Define

$$f_e(x) = \frac{\sum_{p: e \in p} x_p}{c_e} \tag{6.3}$$

to be the relative congestion of the edge $e \in E$. Thus using Corollary 5.2.2, one can find $x \in P$ such that

$$\max_{e \in E} \frac{\sum_{p: e \in p} x_p}{c_e} \leq \exp(\omega) \cdot \lambda^*$$

where $\lambda^* = \min_{x \in P} \max_{e \in E} f_e(x)$ is the optimum. Thus after scaling, we get an $\exp(\omega)$ approximation for the linear program (6.1). The algorithm makes $O(m\omega^{-2} \log m)$ oracle calls where m is the number of edges. An oracle call, given $\{y_e \geq 0 \mid e \in E\}$, minimizes $\langle y, f(x) \rangle$ over $x \in P$. Since

$$\langle y, f(x) \rangle = \sum_{e \in E} y_e \cdot \frac{\sum_{p: e \in p} x_p}{c_e} = \sum_{p \in \mathcal{P}} x_p \left(\sum_{e \in p} \frac{y_e}{c_e} \right)$$

and $\sum_{p \in \mathcal{P}} x_p = 1$, the minimum will be attained when $x_p = 1$ for a path $p \in \mathcal{P}$ which minimizes $\sum_{e \in p} y_e/c_e$. In other words the oracle needs to find a path in $\mathcal{P}$ that is shortest under the length function $y_e/c_e \geq 0$. This can be done by doing t shortest path computations, one for each pair (u_l, v_l), $l \in [t]$. Thus with this approach, to compute the near-optimum maximum multicommodity flow, one needs to make $O(mt\omega^{-2} \log m)$ shortest path computations. This algorithm was given by Garg & Könemann [31].

Note that in the above algorithm, we make t shortest-path computations to augment flow once. Thus there are $t - 1$ "unfruitful" shortest-path computations in each round. Fleischer [20] made the following interesting observation to reduce the number of unfruitful shortest-path computations. Suppose that in round 1, we computed the shortest-path $p \in \mathcal{P}$, of length, say α, under the length function y_e/c_e. Suppose the path p is between the lth pair (u_l, v_l). Now after we augment some flow along p, the dual variables y_e get multiplied by $\exp(\epsilon w_1 f_e(x_1))$ where $\epsilon = \omega/3$ (see the procedure **Update** in Algorithm 5.3). Here the subscript 1 stands for the round 1. Since $w_1 f_e(x_1) \geq 0$, the variables y_e (and therefore the lengths y_e/c_e) do not decrease. Thus in the second round, the length of the shortest path in $\mathcal{P}$ would be at least α. Suppose now that the shortest path p' between the lth pair under the new length function is of length at most $\alpha \exp(\epsilon)$. Note that p' forms an $\exp(\epsilon)$ approximation to the shortest path in $\mathcal{P}$. In Corollary 5.2.4, we observed that it is enough to work with an approximate oracle. For instance, if the oracle routes the flow along a path in $\mathcal{P}$ of length at most $\sigma = \exp(\epsilon)$ times the length of the shortest path, the algorithm would compute a $\exp(\epsilon + \omega)$-approximate solution. What is important is that, by doing this, we have avoided the "unfruitful" shortest-path computations in round 2. Fleischer [20] used this observation systematically to reduce the total number of shortest-path computations to $O((m + t)\omega^{-2} \log m)$. Furthermore, in the case of maximum multicommodity flow problem, we can group the commodities with a common source to make t at most the number of vertices.

In the multicommodity flow problem, let π^l denote the total flow sent between the lth pair. Since we route a unit flow, we have $\sum_{l=1}^{t} \pi^l = 1$. We can divide the variable x_p where $p \sim l$ is a path between the lth pair by π^l so that the total flow between the lth pair becomes unity. Now the functions f_e defined in (6.3) can be equivalently defined as

$$\sum_{l=1}^{t} \pi^l \left(\frac{\sum_{p:e \in p, p \sim l} x_p}{c_e} \right),$$

where $\sum_{p \sim l} x_p = 1$ for each $l \in [t]$ and $\sum_{l=1}^{t} \pi^l = 1$. In this chapter, we generalize the technique of Fleischer [20] to MIXED problems in this form.

6.2 The SUBSTITUTE MIXED problem

We define the SUBSTITUTE MIXED problem as follows.

SUBSTITUTE MIXED: Given non-empty convex sets $P^l \subseteq \mathbb{R}^{n_l}$ for $1 \leq l \leq t$, non-negative continuous convex functions $f_i^l : P^l \to \mathbb{R}_+$ for $1 \leq i \leq m$, and non-negative continuous concave functions $g_j^l : P^l \to \mathbb{R}_+$ for $1 \leq j \leq k$, either find $x^l \in P^l$ and $\pi^l \in \mathbb{R}_+$ for $1 \leq l \leq t$ such that $\sum_{l=1}^{t} \pi^l = 1$ and

$$\max_{1 \leq i \leq m} \sum_{l=1}^{t} \pi^l f_i^l(x^l) \leq \min_{1 \leq j \leq k} \sum_{l=1}^{t} \pi^l g_j^l(x^l),$$

or prove that no such $x^l \in P^l$ and $\pi^l \in \mathbb{R}_+$ for $1 \leq l \leq t$ exist.

We assume that we have the following oracle to solve the above problem.

Given $y \in \mathbb{R}_+^m$, $z \in \mathbb{R}_+^k$, $l \in \{1, \ldots, t\}$, and a real number $\alpha > 0$, the oracle either finds an $x^l \in P^l$ such that,

$$\langle \underline{y}, f^l(x^l) \rangle \leq \alpha \cdot \langle \underline{z}, g^l(x^l) \rangle, \tag{6.4}$$

or proves that no such $x^l \in P^l$ exists.

Note that this oracle deals with the individual sets P^l. To implement this, we can minimize either the ratio $\langle y, f^l(x^l) \rangle / \langle z, g^l(x^l) \rangle$ or the difference $\langle y, f^l(x^l) \rangle - \langle \alpha z, g^l(x^l) \rangle$ over $x^l \in P^l$.

We call this problem SUBSTITUTE MIXED because we have t "substitutes" $P^1, \ldots, P^t$ to choose the solution from.

6.2.1 Reduction to the MIXED problem

Here we show that the SUBSTITUTE MIXED problem can be thought of as a special case of the MIXED problem in which we allow the functions f_i and g_j to be non-convex and non-concave respectively. Given an instance of the SUBSTITUTE MIXED problem, we define a convex set $P \subseteq \mathbb{R}^{n_1 + \cdots + n_t + t}$ and functions $f : P \to \mathbb{R}_+^m$ and $g : P \to \mathbb{R}_+^k$ as follows. Let $\Pi = \{\pi = (\pi^1, \ldots, \pi^t) \in \mathbb{R}_+^t \mid \sum_{l=1}^{t} \pi^l = 1\}$ be the t-dimensional simplex and let

$$P = P^1 \times \cdots \times P^t \times \Pi$$

be the Cartesian product of the convex sets P^l and Π. For a point $x = (x^1, \ldots, x^t, \pi) \in P$, let

$$f_i(x) = \sum_{l=1}^{t} \pi^l f_i^l(x^l) \quad \text{for } i \in [m], \qquad g_j(x) = \sum_{l=1}^{t} \pi^l g_j^l(x^l) \quad \text{for } j \in [k]. \tag{6.5}$$

It is now easy to see that the SUBSTITUTE MIXED problem is equivalent to finding an $x \in P$ such that

$$\max_{i\in[m]} f_i(x) \leq \min_{j\in[k]} g_j(x).$$

This is similar to the MIXED problem defined in Chapter 5 except that the function f_i and g_j defined in (6.5) may not be convex and concave respectively. However, as shown below, we can still use Algorithm 5.2 given for the MIXED problem.

Oracle

The oracle (5.1) for the instance of the MIXED problem defined in (6.5) can be implemented by making t calls to the oracle (6.4) with $\alpha = 1$. This is because finding an $x \in P$ such that $\langle y, f(x)\rangle \leq \langle z, g(x)\rangle$ is the same as finding $x^l \in P^l, l \in [t]$ and $\pi \in \Pi$ such that

$$\left\langle y, \sum_{l=1}^{t} \pi^l f^l(x^l)\right\rangle \leq \left\langle z, \sum_{l=1}^{t} \pi^l g^l(x^l)\right\rangle.$$

Using linearity, we can rewrite the above inequality as

$$\sum_{l=1}^{t} \pi^l \langle y, f^l(x^l)\rangle \leq \sum_{l=1}^{t} \pi^l \langle z, g^l(x^l)\rangle.$$

For this to hold, there must exist an $l \in [t]$ such that

$$\langle y, f^l(x^l)\rangle \leq \langle z, g^l(x^l)\rangle. \tag{6.6}$$

If the given instance of the SUBSTITUTE MIXED problem is feasible, such an $l \in [t]$ can be easily identified by making a call to oracle (6.4) with $\alpha = 1$ for each $l \in [t]$.

Algorithm

We modify Algorithm 5.2 given for the MIXED problem, to get Algorithm 6.1 for the SUBSTITUTE MIXED problem. The procedures **Initialize**, **Update**, and **Stop-condition** are given in Algorithm 6.2. In line (4), we make t calls to the oracle (6.4) as described above. The output $\tilde{x}^l \in P^l$ for $l \in [t]$ and $\tilde{\pi} \in \Pi$ is given as follows. Note that in each round r, we find $l(r) \in [t]$ for which there exists $x^{l(r)} \in P^{l(r)}$ that satisfies (6.6). Let $W = \sum_{r=1}^{N} w_r$ and $W^l = \sum_{r:l(r)=l} w_r$ for $l \in [t]$. Note that $W = \sum_{l=1}^{t} W^l$. Now define

$$\tilde{x}^l = \frac{\sum_{r:l(r)=l} w_r x_r^{l(r)}}{W^l} \quad \text{and} \quad \tilde{\pi}^l = \frac{W^l}{W} \quad \text{for} \quad l \in [t]. \tag{6.7}$$

We remarked in Section 5.1.3 that the inequality (5.6) holds even if the functions f_i are not convex and the functions g_j are not concave (see Remark 5.1.4). This implies

$$\max_{i\in[m]} \sum_{r=1}^{N} w_r f_i^{l(r)}(x_r^{l(r)}) \leq \exp(3\epsilon) \cdot \min_{j\in[k]} \sum_{r=1}^{N} w_r g_j^{l(r)}(x_r^{l(r)}). \tag{6.8}$$

Now, we have

$$\max_{i\in[m]} \frac{\sum_{r=1}^{N} w_r f_i^{l(r)}(x_r^{l(r)})}{W} = \max_{i\in[m]} \sum_{l=1}^{t} \left(\frac{\sum_{r:l(r)=l} w_r f_i^{l(r)}(x_r^{l(r)})}{W}\right)$$

Algorithm SUBSTITUTE MIXED

Input : Convex set $P \subseteq \mathbb{R}^n$, functions $f^l : P \to \mathbb{R}_+^m$, $g^l : P \to \mathbb{R}_+^k$ for $l \in [t]$,
and $\omega \in (0, 1)$
Output : "infeasible" or $\tilde{x}^l \in P^l, \pi^l \in \mathbb{R}_+$ for $l \in [t]$

(1) **Initialize**
(2) $r := 0$ {initialize the round number}
Repeat
(3) $r := r + 1$ {round r begins}
(4) Find $l(r) \in [t]$ such that $\exists\, x_r^{l(r)} \in P^{l(r)}$
such that $\langle \underline{y}, f^{l(r)}(x_r^{l(r)}) \rangle \leq \langle \underline{z}, g^{l(r)}(x_r^{l(r)}) \rangle$ {t oracle calls}
If there is no such $l(r)$, then output "infeasible"
(5) $w_r := 1/\max\{\max_{i\in[m]} f_i^{l(r)}(x_r^{l(r)}),$
$\max_{j:z_j>\exp(-\epsilon\Gamma)} g_j^{l(r)}(x_r^{l(r)})\}$ {pick x_r to an extent w_r}
(6) **Update**$(w_r, f^{l(r)}, g^{l(r)}, x_r^{l(r)})$
(7) Until **Stop-condition** is True {round r ends}
(8) Output $\tilde{x}^l, \tilde{\pi}^l$ for $l \in [t]$ as defined in (6.7)

Algorithm 6.1: An algorithm for the SUBSTITUTE MIXED problem. The procedures **Initialize**, **Update**, and **Stop-condition** are defined in Algorithm 6.2.

Initialize:
Set $\epsilon = \omega/3$
For $i \in [m]$ do $y_i := 1$
For $j \in [k]$ do $z_j := 1$ {initialize the dual variables}
$\Gamma := 2\epsilon^{-2}(\ln m + \exp(2\epsilon)\ln k)$ {see **Stop-condition**}

Update(w, f, g, x)**:**
For $i \in [m]$ do $y_i := y_i \cdot \exp(\epsilon w f_i(x))$
For $j \in [k]$ do
If $z_j > \exp(-\epsilon\Gamma)$, then $z_j := z_j \cdot \exp(-\epsilon w g_j(x))$ {dual variables updated}

Stop-condition:
True, if $\max_{i\in[m]} \sum_{s=1}^{r} w_s f_i^{l(s)}(x_s^{l(s)}) \geq \Gamma$ {stop if some f_i is packed "enough"}
True, if $\min_{j\in[k]} \sum_{s=1}^{r} w_s g_j^{l(s)}(x_s^{l(s)}) \geq \Gamma$ {stop if all g_j are covered "enough"}
False, otherwise

Algorithm 6.2: Procedures for Algorithm 6.1

$$
\begin{aligned}
&= \max_{i\in[m]} \sum_{l=1}^{t} \left(\frac{W^l}{W} \cdot \frac{\sum_{r:l(r)=l} w_r f_i^{l(r)}(x_r^{l(r)})}{W^l} \right) \\
&\geq \max_{i\in[m]} \sum_{l=1}^{t} \tilde{\pi}^l \cdot f_i^l \left(\frac{\sum_{r:l(r)=l} w_r x_r^{l(r)}}{W^l} \right) &(6.9)\\
&= \max_{i\in[m]} \sum_{l=1}^{t} \tilde{\pi}^l \cdot f_i^l(\tilde{x}^l). &(6.10)
\end{aligned}
$$

The inequality (6.9) holds since the functions f_i^l are convex. Similarly using the concavity of the functions g_j^l, we have

$$\min_{j\in[k]} \frac{\sum_{r=1}^{N} w_r g_j^{l(r)}(x_r^{l(r)})}{W} \leq \min_{j\in[k]} \sum_{l=1}^{t} \tilde{\pi}^l \cdot g_j^l(\tilde{x}^l).$$

Thus from (6.8), we have

$$\max_{i\in[m]} \sum_{l=1}^{t} \tilde{\pi}^l \cdot f_i^l(\tilde{x}^l) \leq \exp(3\epsilon) \cdot \min_{j\in[k]} \sum_{l=1}^{t} \tilde{\pi}^l \cdot g_j^l(\tilde{x}^l).$$

Now set $\epsilon = \omega/3$ where $\omega \in (0,1)$ is the given error parameter. Thus $\tilde{x}^l$ for $l \in [t]$ and $\tilde{\pi}$ defined in (6.7) form an approximate solution to the SUBSTITUTE MIXED problem. Since we make t calls to the oracle (6.4) in each round, and the total number of rounds in the algorithm is $O((m+k)\omega^{-2}\log(m+k))$, the total number of calls to the oracle (6.4) is $O((m+k)t\omega^{-2}\log(m+k))$. Note that if we use the oracle (6.4) with the value $\alpha = \sigma$, the inequality (6.8) gets replaced by

$$\max_{i\in[m]} \sum_{r=1}^{N} w_r f_i^{l(r)}(x_r^{l(r)}) \leq \sigma \cdot \exp(3\epsilon) \cdot \min_{j\in[k]} \sum_{r=1}^{N} w_r g_j^{l(r)}(x_r^{l(r)}).$$

This follows from the proof of Theorem 5.1.6. This, in turn, implies that the algorithm computes $\tilde{x}^l$ and $\tilde{\pi}$ such that

$$\max_{i\in[m]} \sum_{l=1}^{t} \tilde{\pi}^l \cdot f_i^l(\tilde{x}^l) \leq \sigma \cdot \exp(3\epsilon) \cdot \min_{j\in[k]} \sum_{l=1}^{t} \tilde{\pi}^l \cdot g_j^l(\tilde{x}^l).$$

We use the above result regarding approximate oracles in the next section to improve the overall running time.

6.2.2 A faster algorithm for the SUBSTITUTE MIXED problem

In this section, using Fleischer's [20] idea of "round-robin increments", we reduce the number of oracle calls from $O((m+k)t\omega^{-2}\log(m+k))$ to $O((m+k+t)\omega^{-2}\log(m+k))$ and prove the following theorem.

Theorem 6.2.1 *There exists an algorithm for the* SUBSTITUTE MIXED *problem that given an error parameter* $\omega \in (0,1)$*,*

- *either computes $x^l \in P^l$ for $l \in [t]$ and $\pi \in \Pi$ such that*

$$\max_{i\in[m]} \sum_{l=1}^{t} \pi^l f_i^l(x^l) \leq \exp(\omega) \cdot \min_{j\in[k]} \sum_{l=1}^{t} \pi^l g_j^l(x^l),$$

(4a) While $x^l \in P^l$ for some $l \in [t]$ is not found in this round do
(4b) If l is undefined, let $l := 1$
(4c) Find $x^l \in P^l$ such that $\langle \underline{y}, f^l(x^l) \rangle \leq \exp(\epsilon) \cdot \langle \underline{z}, g^l(x^l) \rangle$ {oracle call}
(4d) If there is no such $x^l \in \overline{P}^l$,
then
(4e) If we have cycled through all $l \in [t]$ without finding x^l,
(4f) then output "infeasible"
(4g) else $l := (l \bmod t) + 1$ {go to the next convex set}
(4h) else $l(r) := l$ and $x_r^{l(r)} := x^l$ {solution in round r}
end While

Algorithm 6.3: An implementation of the oracle call in line (4) of Algorithm 6.1

- *or proves that there do not exist $x^l \in P^l$ for $l \in [t]$ and $\pi \in \Pi$ such that*

$$\max_{i \in [m]} \sum_{l=1}^{t} \pi^l f_i^l(x^l) \leq \min_{j \in [k]} \sum_{l=1}^{t} \pi^l g_j^l(x^l).$$

The algorithm makes $O((m + k + t)\omega^{-2} \log(m + k))$ calls to the oracle (6.4) with $\alpha = \exp(\omega/4)$. The algorithm takes additional time equal to the time to compute $f^l(x^l)$ and $g^l(x^l)$ for some $x^l \in P^l$ for some $l \in [t]$ between successive oracle calls.

The key idea here is Theorem 5.1.6 which states that one can work with an approximate oracle which given any non-zero $y, z \geq 0$, finds an $x \in P$ such that

$$\langle \underline{y}, f(x) \rangle \leq \alpha \cdot \langle \underline{z}, g(x) \rangle. \tag{6.11}$$

We let $\alpha = \exp(\epsilon)$ in each oracle call.

The algorithm is identical to Algorithm 6.1, except that

- in the procedure **Initialize**, we now set $\epsilon = \omega/4$ where $\omega \in (0, 1)$ is the given error parameter,
- the oracle call in line (4) is implemented as given in Algorithm 6.3.

In the first execution of lines (4a)–(4h), we initialize $l = 1$. We make a call to the oracle (6.4) in line (4c) with the current value of l. If in round r, we find an $x^l \in P^l$ with the desired property, we let $l(r)$ to be the current value of $l \in [t]$ and $x_r^{l(r)} = x^l \in P^l$ to be the output of the oracle call. If we cannot find $x^l \in P^l$ with the desired property, we move on to the next convex set in the cyclic order by setting $l := (l \bmod t) + 1$. If, at this point, we observe that we have cycled through all t values of l without finding an appropriate $x^l \in P^l$, we declare that the given instance of the SUBSTITUTE MIXED problem is infeasible. We justify this step in Lemma 6.2.2. In the execution of the lines (4a)–(4h) in the next round, we start with the current value of l; thus l is considered as a "global" variable.

Analysis

Lemma 6.2.2 *If the input instance of the* SUBSTITUTE MIXED *problem is feasible, there exists an $x^l \in P^l$ with $\langle \underline{y}, f^l(x^l) \rangle \leq \langle \underline{z}, g^l(x^l) \rangle$ for some $l \in [t]$ and in any execution of the while-loop in the lines (4a)–(4h), we find an $x^l \in P^l$ with $\langle \underline{y}, f^l(x^l) \rangle \leq \exp(\epsilon) \cdot \langle \underline{z}, g^l(x^l) \rangle$ for some $l \in [t]$.*

Proof. The argument is similar to the one in Section 6.2.1. If the input instance is feasible, there exists an $x = (x^1, \ldots, x^t, \pi) \in P$ such that

$$\left\langle \underline{y}, \sum_{l=1}^{t} \pi^l f^l(x^l) \right\rangle \leq \left\langle \underline{z}, \sum_{l=1}^{t} \pi^l g^l(x^l) \right\rangle.$$

By linearity, this implies that there exists an $l \in [t]$ such that

$$\langle \underline{y}, f^l(x^l) \rangle \leq \langle \underline{z}, g^l(x^l) \rangle.$$

Since we cycle through all $l \in [t]$, we shall be able to find an $\hat{x}^l \in P^l$ such that $\langle \underline{y}, f^l(\hat{x}^l) \rangle \leq \exp(\epsilon) \cdot \langle \underline{z}, g^l(\hat{x}^l) \rangle$. Hence the lemma. ∎

Since we use the oracle (6.4) with $\alpha = \exp(\epsilon)$, the discussion at the end of Section 6.2.1 implies that the solution defined in (6.7) satisfies

$$\max_{i \in [m]} \sum_{l=1}^{t} \tilde{\pi}^l \cdot f_i^l(\tilde{x}^l) \leq \exp(\epsilon) \cdot \exp(3\epsilon) \cdot \min_{j \in [k]} \sum_{l=1}^{t} \tilde{\pi}^l \cdot g_j^l(\tilde{x}^l).$$

Since $\epsilon = \omega/4$, this solution satisfies the condition in Theorem 6.2.1.

Bounding the number of oracle calls

Now we argue that the total number of oracle calls made by the algorithm is $O((m + k + t)\epsilon^{-2} \log(m + k))$. We divide our algorithm into phases. At any point in the algorithm, let $\alpha = \langle \mathbf{1}, y \rangle / \langle \mathbf{1}, z \rangle$ for the current values of y and z. Let us denote the initial value of α as α_0; note that $\alpha_0 = m/k$. Since the dual variables y do not decrease and the dual variables z do not increase (see procedure **Update** in Algorithm 6.2), the value of α does not decrease. We say that the algorithm is in phase $p \geq 1$ if $\exp((p-1)\epsilon) \leq \alpha/\alpha_0 < \exp(p\epsilon)$. The following lemma gives a bound on the total number of phases in the algorithm.

Lemma 6.2.3 *The number of phases in the algorithm is* $O(\epsilon^{-2} \log(m + k))$.

Proof. Let N denote the total number of rounds in the algorithm. The dual variables y_i and z_j are updated in the procedure **Update** in Algorithm 6.2. Note that at the end of the algorithm, we have $y_i = \exp(\epsilon \sum_{r=1}^{N} w_r f_i^{l(r)}(x_r^{l(r)}))$. Since we terminate (see **Stop-condition** in Algorithm 6.2) when $\max_{i \in [m]} \sum_{r=1}^{N} w_r f_i^{l(r)}(x_r^{l(r)}) \geq \Gamma$, the maximum value that any y_i can take is $\exp(\epsilon(\Gamma + 1))$. Note that we have added 1 to Γ since in the last round, y_i can get multiplied by a factor of at most $\exp(\epsilon)$. This, in turn, follows from the fact that $w_r f_i^{l(r)}(x_r^{l(r)}) \leq 1$ in any round r. Similarly, from the way we modify z_j in the procedure **Update**, the minimum value that any z_j can take is $\exp(-\epsilon(\Gamma + 1))$. Therefore, the maximum value that $\alpha = \langle \mathbf{1}, y \rangle / \langle \mathbf{1}, z \rangle$ can take is $(m \exp(\epsilon(\Gamma + 1)))/(k \exp(-\epsilon(\Gamma + 1))) = \alpha_0 \exp(2\epsilon(\Gamma + 1))$. Since the value of α increases by $\exp(\epsilon)$ in each phase, the number of phases is at most $2\Gamma + 2 = O(\epsilon^{-2} \log(m + k))$. ∎

Lemma 6.2.4 *If the input instance is feasible, the number of oracle calls is* $O((m + k + t)\epsilon^{-2} \log(m + k))$.

Proof. We divide the oracle calls into two categories. We call an oracle call in line (4c) "fruitful" if it is able to find an $x^l \in P^l$ with the desired property; otherwise, we call it "unfruitful". The proof of Lemma 5.1.5 implies that the total number of rounds in the algorithm is $O((m + k)\epsilon^{-2} \log(m + k))$. Since there is exactly one fruitful call per round, the number of fruitful oracle calls is also $O((m + k)\epsilon^{-2} \log(m + k))$.

We argue that there are at most t unfruitful oracle calls in any phase so that from Lemma 6.2.3, there are $O(t\epsilon^{-2} \log(m + k))$ unfruitful oracle calls in all. We argue that when we cycle through all $l \in [t]$ once, the value of α increases by a factor of at least $\exp(\epsilon)$ so that the phase changes. Let us fix a phase p and assume, in order to get a contradiction, that we make more than t unfruitful oracle calls in this phase. Let l_1 be the value of l at the beginning of the phase. We make a series of (possibly zero) fruitful oracle calls with $l = l_1$ ending in an unfruitful oracle call. At this point we have

$$\min_{x^{l_1} \in P^{l_1}} \frac{\langle y, f^{l_1}(x^{l_1})\rangle / \langle z, g^{l_1}(x^{l_1})\rangle}{\langle \mathbf{1}, y\rangle / \langle \mathbf{1}, z\rangle} = \min_{x^{l_1} \in P^{l_1}} \frac{\langle \underline{y}, f^{l_1}(x^{l_1})\rangle}{\langle \underline{z}, g^{l_1}(x^{l_1})\rangle} > \exp(\epsilon).$$

Therefore we have

$$\min_{x^{l_1} \in P^{l_1}} \frac{\langle y, f^{l_1}(x^{l_1})\rangle}{\langle z, g^{l_1}(x^{l_1})\rangle} > \alpha \exp(\epsilon) \geq \alpha_0 \exp(p\epsilon). \tag{6.12}$$

Now we increment the value of l to $(l_1 \bmod t + 1)$. We again make a series of (possibly zero) fruitful oracle calls ending in an unfruitful oracle call for this value of l. Thus the inequality (6.12) holds for this value of l as well. After t unfruitful oracle calls, the inequality (6.12) holds for all $l \in [t]$. Since we are still in phase p, by definition, we have $\alpha < \alpha_0 \exp(p\epsilon)$. Therefore at this point for all $l \in [t]$, we have

$$\min_{x^l \in P^l} \frac{\langle \underline{y}, f^l(x^l)\rangle}{\langle \underline{z}, g^l(x^l)\rangle} = \min_{x^l \in P^l} \frac{\langle y, f^l(x^l)\rangle / \langle z, g^l(x^l)\rangle}{\langle \mathbf{1}, y\rangle / \langle \mathbf{1}, z\rangle} > \frac{\alpha_0 \exp(p\epsilon)}{\alpha_0 \exp(p\epsilon)} = 1.$$

This implies (Lemma 6.2.2) that the input instance is infeasible which is a contradiction. Thus the number of unfruitful calls per phase is, in fact, at most $t-1$ and the proof is complete. ∎

Thus we have proved Theorem 6.2.1.

6.2.3 The SUBSTITUTE PACKING problem

In Section 5.2, we considered some special cases of the MIXED problem that followed directly from the MIXED problem. These special cases included the min-max resource sharing or PACKING (Corollary 5.2.2) and max-min resource sharing or COVERING (Corollary 5.2.5). Similar special cases can also be derived in the case of the SUBSTITUTE MIXED problem.

We first consider the SUBSTITUTE PACKING problem and prove the following theorem.

Corollary 6.2.5 (SUBSTITUTE PACKING) *There is an algorithm that finds $x^l \in P^l$ for $l \in [t]$ and $\pi \in \Pi$ such that*

$$\max_{i \in [m]} \sum_{l=1}^{t} \pi^l f_i^l(x^l) \leq \exp(\omega) \cdot \lambda^*$$

where λ^* is the minimum value of $\max_{i \in [m]} \sum_{l=1}^{t} \pi^l f_i^l(x^l)$ where $x^l \in P^l$ for $l \in [t]$ and $\pi \in \Pi$. The algorithm makes $O((m+t)\omega^{-2} \log m)$ calls to an oracle that given $y \in \mathbb{R}_+^m$ and $l \in [t]$, finds an $x^l \in P^l$ that minimizes $\langle y, f^l(x^l)\rangle$.

The proof of Corollary 6.2.5 is similar to that of Corollary 5.2.3. Define a constant function $g_0^l \equiv \lambda^*$ for $l \in [t]$. Consider the instance of the SUBSTITUTE MIXED problem with functions f_i^l and g_0^l. Thus, we have $k = 1$ in this instance and hence for any non-zero $z \in \mathbb{R}_+^k$, we have $\underline{z} = 1$. Hence the oracle call in line (4c) in Algorithm 6.3 reduces to finding an $x^l \in P^l$ such that $\langle \underline{y}, f^l(x^l)\rangle \leq \exp(\epsilon) \cdot \lambda^*$. Since we do not know the value of λ^*, we replace the implementation of the oracle call given in Algorithm 6.3 by the one given in Algorithm 6.4.

In the first execution of lines (4a)–(4h), we initialize $\alpha = \min_{l \in [t]} \min_{x^l \in P^l} \langle y, f^l(x^l)\rangle$. This value can be computed by making t calls to the oracle which given $y \in \mathbb{R}_+^m$ and $l \in [t]$ computes $x^l \in P^l$ that minimizes $\langle y, f^l(x^l)\rangle$. The oracle call in line (4c) can be implemented by minimizing $\langle y, f^l(x^l)\rangle$ over $x^l \in P^l$ and comparing the minimum value with α. We now show that the oracle call in line (4c) in Algorithm 6.4 is "equivalent" to finding an $x^l \in P^l$ such that $\langle \underline{y}, f^l(x^l)\rangle \leq \exp(\epsilon) \cdot \lambda^*$.

(4a)	While $x^l \in P^l$ for some $l \in [t]$ is not found in this round do	
(4b)	If l is undefined, let $l := 1$	
	If α is undefined, let $\alpha = \min_{l \in [t]} \min_{x^l \in P^l} \langle y, f^l(x^l) \rangle$	
(4c)	Find $x^l \in P^l$ such that $\langle y, f^l(x^l) \rangle \leq \alpha$	{oracle call}
(4d)	If there is no such $x^l \in P^l$,	
	then	
(4e)	If for the current value of α	
	we have cycled through all $l \in [t]$ without finding x^l,	
(4f)	then $\alpha := \exp(\epsilon) \cdot \alpha$	
(4g)	else $l := (l \bmod t) + 1$	{go to the next convex set}
(4h)	else $l(r) := l$ and $x_r^{l(r)} := x^l$	{solution in round r}
	end While	

Algorithm 6.4: An implementation of the oracle call for the SUBSTITUTE PACKING algorithm

Lemma 6.2.6 *In any execution of line (4c) in Algorithm 6.4, if an $x^l \in P^l$ satisfies $\langle y, f^l(x^l) \rangle \leq \alpha$, it also satisfies $\langle \underline{y}, f^l(x^l) \rangle \leq \exp(\epsilon) \cdot \lambda^*$.*

Proof. At any point in the algorithm let $\Lambda = \min_{l \in [t]} \min_{x^l \in P^l} \langle y, f^l(x^l) \rangle$. We first prove by induction that at any point $\alpha \leq \exp(\epsilon) \cdot \Lambda$ holds. From the way we have initialized α, this is true in the beginning. Since the values of the dual variables y do not decrease, Λ does not decrease. Now we update α in line (4f), only if we find that $\Lambda > \alpha$. Therefore $\alpha \leq \exp(\epsilon) \cdot \Lambda$ continues to hold after the update. Thus the induction is complete. Now suppose we find an $x^l \in P^l$ satisfying $\langle y, f^l(x^l) \rangle \leq \alpha$. We have

$$\begin{aligned} \langle \underline{y}, f^l(x^l) \rangle &\leq \frac{\alpha}{\langle \mathbf{1}, y \rangle} \\ &\leq \frac{\exp(\epsilon) \cdot \Lambda}{\langle \mathbf{1}, y \rangle} \\ &= \exp(\epsilon) \cdot \min_{l \in [t]} \min_{x^l \in P^l} \langle \underline{y}, f^l(x^l) \rangle \\ &\leq \exp(\epsilon) \cdot \lambda^*. \end{aligned}$$

The last inequality holds since we have $\min_{l \in [t]} \langle \underline{y}, f^l(x^{*l}) \rangle \leq \lambda^*$, for an optimum solution $x^* = (x^{*1}, \ldots, x^{*t}, \pi^*)$. ∎

Lemma 6.2.7 *The number of oracle calls (line (4c)) in the algorithm is $O((m + t)\omega^{-2} \log m)$.*

Proof. The proof is similar to those of Lemmas 6.2.3 and 6.2.4. Call an oracle call in line (4c) "fruitful" if we find $x^l \in P^l$ as desired; and "unfruitful" otherwise. Note that each y_i is initialized to 1. From the **Stop-condition** in Algorithm 6.2, the maximum value that any y_i can take is $O(\exp(\epsilon(\Gamma + 1))$. Since after each fruitful oracle call, at least one y_i gets multiplied by $\exp(\epsilon)$, the total number of fruitful oracle calls in the algorithm is $m(\Gamma + 1) = O(m\omega^{-2} \log m)$.

At any point in the algorithm, let $\Lambda = \min_{l \in [t]} \min_{x^l \in P^l} \langle y, f^l(x^l) \rangle$. Let initial value of Λ be Λ_0. Note that Λ_0 is also the initial value of α. Since the maximum value that any y_i can take is $O(\exp(\epsilon(\Gamma + 1))$, the final value of Λ is at most $\Lambda_0 \exp(\epsilon(\Gamma + 1))$. Now, from the proof of Lemma 6.2.6, we have $\alpha \leq \exp(\epsilon) \cdot \Lambda$. Thus the final value of α is at most $\Lambda_0 \exp(\epsilon(\Gamma + 2))$. Since α gets updated by a multiplicative factor of $\exp(\epsilon)$, the number of updates of α is at most $\Gamma + 2 = O(\omega^{-2} \log m)$. Note that the number of unfruitful oracle calls between successive updates of α is t (see line (4e)).

Therefore, the total number of unfruitful oracle calls in the algorithm is $O(t\omega^{-2}\log m)$. We make t additional oracle calls to initialize α. Therefore the total number of oracle calls in the algorithm is $O((m+t)\omega^{-2}\log m)$. ∎

Thus we have proved Corollary 6.2.5. We can, in fact, prove that the above algorithm also computes a near optimum dual solution. For $y \geq 0$, define $d(y) = \min_{l\in[t]} \min_{x^l \in P^l} \langle \underline{y}, f^l(x^l)\rangle$.

Corollary 6.2.8 *There is an algorithm that finds $x^l \in P^l$ for $l \in [t]$, $\pi \in \Pi$, and $y \geq 0$ such that*

$$\max_{i\in[m]} \sum_{l=1}^{t} \pi^l f_i^l(x^l) \leq \exp(\omega)\cdot d(y).$$

The algorithm makes $O((m+t)\omega^{-2}\log m)$ calls to the oracle that given $y \in \mathbb{R}_+^m$ and $l \in [t]$, finds an $x^l \in P^l$ that minimizes $\langle y, f^l(x^l)\rangle$.

Proof. The proof is similar to that of Corollary 6.2.5 except that the functions g_0^l are now defined as $g_0^l \equiv d(y)$ for $y \geq 0$ as defined below. Let y^{r-1} denote the value of the dual variables y at the end of round $r-1$. Define

$$y = y^{p-1} \text{ where } p = \arg\max_{1\leq r\leq N} d(y^{r-1}).$$

∎

The maximum multicommodity flow problem revisited

The maximum multicommodity flow problem discussed in Section 6.1 is, in fact, an instance of the SUBSTITUTE PACKING problem. The objective is to minimize

$$\sum_{l=1}^{t} \pi^l \left(\frac{\sum_{p\sim l} x_p}{c_e}\right)$$

where $\pi \in \Pi$ and $\sum_{p\sim l} x_p = 1$ for $l \in [t]$. From Corollary 6.2.8 we can compute an $\exp(\omega)$ approximation by making $O((m+t)\omega^{-2}\log m)$ shortest-path computations. This can be reduced to $O(m\omega^{-2}\log m)$ by grouping the commodities with a common source node and observing that computing shortest paths from a source to all other nodes takes the same asymptotic running time as computing a shortest path from a source to one specified node. This running time was first given by Fleischer [20].

6.2.4 The maximum cut problem in weakly bipartite graphs

Given an undirected graph $G = (V, E)$ with weights $w_e \geq 0$ on the edges, the maximum cut problem is to find a bipartite sub-graph of G with maximum total weight. The max-cut problem is known to be **NP**-hard even in the unweighted case [51, 29]. However in *weakly bipartite* graphs, it can be solved in polynomial time.

Since bipartite sub-graphs do not have odd cycles, a natural question to ask is: which are the graphs such that the convex hull of incidence vectors of bipartite sub-graphs is defined by the odd cycle inequalities. Grötschel and Pulleyblank [37] called such graphs weakly bipartite. Formally, a graph $G = (V, E)$ is called weakly bipartite if the following polyhedron Q is integral (i.e., all its extreme points are integral):

$$Q = \{x \in \mathbb{R}_+^{|E|} \mid \sum_{e\in C} x_e \geq 1 \text{ for all odd cycles } C \text{ in } G\}.$$

Let x be an extreme 0-1 point of Q. Then x is the incidence vector of an edge-set which intersects every odd cycle. In other words, $\mathbf{1} - x$ is the incidence vector of a bipartite sub-graph of G. Thus the optimum vertex solution to

$$\min\{\sum_{e \in E} w_e x_e \mid x \in Q\} \tag{6.13}$$

forms the incidence vector of the maximum cut in G. Grötschel & Pulleyblank [37] showed that this optimization problem (and hence the max-cut problem in weakly bipartite graphs) can be solved in polynomial time by the ellipsoid method. Guenin [38, 39] characterized weakly bipartite graphs as the graphs which do not contain K_5 as an odd minor (see also Schrijver [73]). We say that a graph G contains H as an odd minor if H can be obtained from G by deleting vertices and edges, and by contracting all edges in a cut. In particular, all graphs not contractible[1] to K_5 are weakly bipartite [22]. Thus planar graphs are also weakly bipartite.

In this section, we outline an $O^*(m^2\omega^{-2})$-time algorithm for computing an $\exp(\omega)$ approximation to the linear program (6.13). Here m denotes the number of edges in G. Thus in the weakly bipartite graphs, we can compute $\exp(\omega)$ approximation to the weight of the maximum cut in time $O^*(m^2\omega^{-2})$.

The dual of (6.13) is the following packing linear program.

$$\begin{array}{rrcll} \max & \sum_C y_C & & & \\ \text{s.t.} & \sum_{C: e \in C} y_C & \leq & w_e & \text{for } e \in E \\ & y_C & \geq & 0 & \text{for each odd cycle } C \end{array} \tag{6.14}$$

The variables and the columns of the constraint matrix are indexed by the odd cycles C in G. Observe that the above linear program is very similar to the one for the maximum multicommodity flow problem (6.1). To solve this linear program in a manner similar to the maximum multicommodity flow problem, we group the odd cycles C according to a common vertex. More precisely, for a vertex $u \in V$, we define a polytope $P^u = \{(y_C : C \sim u) \geq 0 \mid \sum_C y_C = 1\}$. Here $C \sim u$ denotes an odd cycle C passing through the vertex $u \in V$. Then the linear program (6.14) is equivalent (up to scaling) to finding $y^u \in P^u$ and $\pi^u \geq 0$ for $u \in V$ such that $\sum_{u \in V} \pi^u = 1$ and

$$\max_{e \in E} \sum_{u \in V} \pi^u \left(\frac{\sum_{C: e \in C, C \sim u} y_C^u}{w_e} \right) \tag{6.15}$$

is minimized.

Lemma 6.2.9 *If* OPT *denotes the optimum value of the linear programs (6.13) and (6.14), then the optimum value of the above formulation (6.15) is* $1/\mathsf{OPT}$.

Proof. The proof is very similar to that for the maximum multicommodity flow problem given in Section 6.1 and is omitted. ∎

Now we need an oracle that given $\{l_e \geq 0 \mid e \in E\}$ and $u \in V$, computes an odd cycle $C \sim u$ such that $\sum_{e \in C} l_e/w_e$ is minimum. Such a cycle can be found by doing a single shortest-path computations as described below. Construct an undirected graph H consisting of two disjoint copies V_1 and V_2 of the vertex-set of G. Let v_1 (resp. v_2) denote the copy of a vertex $v \in V$ in V_1 (resp. V_2). For each edge $e = (v, w) \in E$, introduce two edges (v_1, w_2) and (v_2, w_1) in H and assign them a length of l_e/w_e each. Now, compute the shortest path between u_1 and u_2. The edge-set of this path, then, corresponds to the minimum length odd cycle passing through u under the length function l_e/w_e. Thus an oracle call can be implemented

[1] A graph G is contractible to H if H can be obtained from G by a sequence of edge contractions, in which a pair of adjacent vertices is identified and all other adjacencies are preserved (multiple edges arising from the identification being replaced by single edges).

in $O^*(m)$ time. Now from Corollary 6.2.8, there is an algorithm that computes $y^u \in P^u$ and $\pi^u \geq 0$ for $u \in V$ such that $\sum_{u \in V} \pi^u = 1$ and a dual solution $\{x_e \geq 0 \mid e \in E\}$. These solutions satisfy

$$\exp(\omega) \cdot d(x) \geq \max_{e \in E} \sum_{u \in V} \pi^u \left(\frac{\sum_{C: e \in C, C \sim u} y^u_C}{w_e} \right) \geq 1/\mathsf{OPT}$$

where the last inequality follows from Lemma 6.2.9.

Theorem 6.2.10 *The assignment $\tilde{x}_e = \underline{x}_e/(w_e d(x))$ for $e \in E$ satisfies $\tilde{x} \in Q$ and $\sum_{e \in E} w_e \tilde{x}_e \leq \exp(\omega) \cdot \mathsf{OPT}$. Here $\underline{x}_e = x_e / \sum_{e' \in E} x_{e'}$ for $e \in E$.*

Proof. We have $\sum_{e \in E} w_e \tilde{x}_e = \sum_{e \in E} \underline{x}_e / d(x) = 1/d(x) \leq \exp(\omega) \cdot \mathsf{OPT}$. By definition, we have

$$d(x) = \min_{u \in V} \min_{y^u \in P^u} \sum_{e \in E} \underline{x}_e \left(\frac{\sum_{C: e \in C, C \sim u} y^u_C}{w_e} \right).$$

Now fix any odd cycle C, say passing through a vertex $u \in V$. Consider a point $y^u \in P^u$ defined by $y^u_C = 1$ and $y^u_{C'} = 0$ for $C' \neq C$. For such a point, we have $d(x) \leq \sum_{e \in C} \underline{x}_e / w_e$. This, in turn, implies that $1 \leq \sum_{e \in C} \underline{x}_e/(w_e d(x)) = \sum_{e \in C} \tilde{x}_e$. Therefore $\tilde{x} \in Q$. ∎

Since the algorithm makes $O((m+n)\omega^{-2} \log m)$ oracle calls where each oracle call takes $O^*(m)$ time, the overall running time of the algorithm is $O^*(m^2 \omega^{-2})$.

Other approaches for the max-cut in weakly bipartite graphs

Grötschel & Pulleyblank [37] first defined weakly bipartite graphs and presented an ellipsoid-method based polynomial-time algorithm to solve the linear program considered above. The graphs that are not contractible to K_5 are weakly bipartite. Hadlock [40] gave an $O(n^3)$-time algorithm for computing the max-cut in planar graphs while Barahona [5] presented an $O(n^5)$-time combinatorial algorithm for computing the max-cut in graphs that are not contractible to K_5. Barahona [6] proved that the graphs which have two vertices which cover all odd cycles are weakly bipartite.

6.2.5 The SUBSTITUTE COVERING problem

Results similar to the ones obtained for the SUBSTITUTE PACKING problem also hold for the SUBSTITUTE COVERING problem.

Corollary 6.2.11 (SUBSTITUTE COVERING) *There is an algorithm that finds $x^l \in P^l$ for $l \in [t]$ and $\pi \in \Pi$ such that*

$$\min_{j \in [k]} \sum_{l=1}^{t} \pi^l g^l_j(x^l) \geq \exp(-\omega) \cdot \lambda^*$$

where $\lambda^* = \max_{\{x^l \in P^l\}, \pi \in \Pi} \min_{j \in [k]} \sum_{l=1}^{t} \pi^l g^l_j(x^l)$ is the optimum value. The algorithm makes $O((k+t)\omega^{-2} \log k)$ calls to the oracle that given $z \in \mathbb{R}^k_+$ and $l \in [t]$, finds an $x^l \in P^l$ that maximizes $\langle z, g^l(x^l) \rangle$.

For $z \geq 0$, define $d(z) = \max_{l \in [t]} \max_{x^l \in P^l} \langle \underline{z}, g^l(x^l) \rangle$.

Corollary 6.2.12 *There is an algorithm that finds $x^l \in P^l$ for $l \in [t]$, $\pi \in \Pi$, and $z \geq 0$ such that*

$$\min_{j \in [k]} \sum_{l=1}^{t} \pi^l g^l_j(x^l) \geq \exp(-\omega) \cdot d(z).$$

The algorithm makes $O((k+t)\omega^{-2}\log k)$ *calls to the oracle that given* $z \in \mathbb{R}_+^k$ *and* $l \in [t]$*, finds an* $x^l \in P^l$ *that maximizes* $\langle z, g^l(x^l)\rangle$*.*

These corollaries extend also to the case of approximate oracles. The proofs are similar to those of Corollaries 5.2.2 and 5.2.3 and are omitted.

6.2.6 The maximum independent set problem in t-perfect graphs

A subset of vertices $S \subset V$ in an undirected graph $G = (V, E)$ is called *independent* if each pair of vertices in S is nonadjacent. The independent set problem is to determine an independent set of maximum cardinality. This problem is **NP**-hard, see for example [29]. The independent set problem can be formulated as an integer program with the following linear programming relaxation.

$$\begin{array}{lrcl} \max & \sum_{u\in V} x_u & & \\ \text{s.t.} & & & \\ & x_u + x_v & \leq & 1 \quad \forall e = (u,v) \in E \\ & x_u & \geq & 0 \quad \forall u \in V \end{array} \tag{6.16}$$

Let us define an *odd cycle* to be a multiset $\{v_1, \ldots, v_p\}$ of vertices such that $p \leq |V|$ is odd and $\{v_i, v_{i+1}\} \in E$ for $i = 1, \ldots, p-1$ and $\{v_p, v_1\} \in E$. Chvátal [13] gave the following class of inequalities that are valid for any integral solution to the linear program (6.16).

$$\sum_{u\in C} x_u \leq \lfloor |C|/2 \rfloor \qquad \text{for every odd cycle } C. \tag{6.17}$$

He called a graph t-perfect if the inequalities (6.17) together with the inequalities in the linear program (6.16) define an integral polytope, i.e., all the extreme points of the polytope are integral. Let us call the linear program (6.16) together with the inequalities (6.17) as an *augmented* linear program. Thus one can find the maximum independent set in t-perfect graphs by finding an optimum vertex solution to the augmented linear program.

Recently, Eisenbrand et al. [18] presented an $O^*(n^3 m\omega^{-2})$-time combinatorial algorithm to compute a $(1+\omega)$-approximate solution to the augmented linear program. Here n and m represent the number of vertices and edges respectively. We improve this result by a factor $\Omega(n)$ by giving an algorithm to compute an $(1+\omega)$-approximate solution to the augmented linear program in time $O^*(n^2 m\omega^{-2})$. The dual of the augmented linear program is the following covering linear program. Let $\mathcal{C}$ denote the set of all odd cycles in G.

$$\begin{array}{lrcl} \min & \sum_{e\in E} y_e + \sum_{C\in\mathcal{C}} y_C \cdot \lfloor |C|/2 \rfloor & & \\ \text{s.t.} & & & \\ & \sum_{e:u\in e} y_e + \sum_{C:u\in C} y_C & \geq & 1 \quad \forall u \in V \\ & y_e, y_C & \geq & 0 \quad \forall e \in E, C \in \mathcal{C} \end{array} \tag{6.18}$$

Similar to the discussion for the maximum multicommodity flow problem in Section 6.1, we can show that the above linear program is equivalent, up to scaling of variables, to the following.

$$\begin{array}{lrcl} \max & \min_{u\in V} \left(\sum_{e:u\in e} y_e + \sum_{C:u\in C} y_C \right) & & \\ \text{s.t.} & & & \\ & \sum_{e\in E} y_e + \sum_{C\in\mathcal{C}} y_C \cdot \lfloor |C|/2 \rfloor & = & 1 \\ & y_e, y_C & \geq & 0 \quad \text{for } e \in E, C \in \mathcal{C} \end{array} \tag{6.19}$$

Now for each vertex $u \in V$, define a polytope P^u as

$$P^u = \{(y_C : C \ni u) \geq 0 \mid \sum_{C \sim u} y_C \cdot \lfloor |C|/2 \rfloor = 1\}.$$

Also define a polytope P^E as

$$P^E = \{(y_e : e \in E) \geq 0 \mid \sum_{e \in E} y_e = 1\}.$$

Now solving (6.19) is equivalent to finding $y^u \in P^u, \pi^u \geq 0$ for $u \in V$ and $y^E \in P^E, \pi^E \geq 0$ such that $\pi^E + \sum_{u \in V} \pi^u = 1$ and

$$\min_{u \in V} \left(\left(\pi^E \sum_{e : u \in e} y_e^E \right) + \sum_{u \in V} \left(\pi^u \sum_{C : u \in C} y_C^u \right) \right)$$

is maximized. This formulation can be thought of as a SUBSTITUTE COVERING problem with $n+1$ convex sets $P^u, u \in V$ and P^E. The key difference between the approach of Eisenbrand et al. [18] and our approach is that while they think of the above formulation as a COVERING problem, we think of it as a SUBSTITUTE COVERING problem.

To solve this SUBSTITUTE COVERING problem, we need an oracle that given $\{r_v \geq 0 \mid v \in V\}$,

- computes $y^E \in P^E$ such that $\sum_{v \in V} \left(r_v \sum_{e : v \in e} y_e^E \right) = \sum_{e=(v,w) \in E} y_e^E (r_v + r_w)$ is maximized,
- and further given $u \in V$, computes $y^u \in P^u$ such that $\sum_{v \in V} \left(r_v \sum_{C : u, v \in C} y_C^u \right)$ is maximized.

In the first case, since $\sum_{e \in E} y_e^E = 1$ for $y^E \in P^E$, to maximize $\sum_{e=(v,w) \in E} y_e (r_v + r_w)$, we set $y_e^E = 1$ for an edge $e = (v, w)$ for which $r_v + r_w$ is maximum. We can find such an edge in $O(m)$ time. In the second case, since $\sum_{C \in \mathcal{C}} y_C^u \cdot \lfloor |C|/2 \rfloor = 1$ for any $y^u \in P^u$, to maximize $\sum_{v \in V} \left(r_v \sum_{C : u, v \in C} y_C^u \right)$, we need to set $y_C^u = 1/\lfloor |C|/2 \rfloor$ for the odd cycle $C \sim u$ for which $\sum_{v \in C} r_v / \lfloor |C|/2 \rfloor$ is maximum.

Eisenbrand et al. [18] give an $O^*(nm)$-time procedure to find such a cycle C. For the sake of completeness, we outline their procedure here. Note that we consider odd cycles of length at most n. Thus the value of $\lfloor |C|/2 \rfloor$ lies between 1 and $\lfloor n/2 \rfloor$. We fix a value $l, 1 \leq l \leq \lfloor n/2 \rfloor$ and find an odd cycle $C \sim u$ such that $\lfloor |C|/2 \rfloor = l$ and $\sum_{v \in C} r_v$ is maximum. Trying all possible values of l, we can then find a cycle C as desired. We can perform the entire computation for all values of $l, 1 \leq l \leq \lfloor n/2 \rfloor$, by doing a single one-to-all shortest-path computation in a graph with $O(n^2)$ vertices and $O(nm)$ edges with non-negative lengths as shown below. We construct a graph H whose vertex-set is the union of $n+1$ disjoint copies $V_1, \ldots, V_{n+1}$ of the vertex-set V. Denote the ith copy of a vertex v by v_i. For each edge $(v, w) \in E$, we introduce directed edges (v_i, w_{i+1}) for $1 \leq i \leq n$ in this layered graph and assign them a length of $r_{\max} - r_w$ each. Here $r_{\max} = \max_{v \in V} r_v$. Now compute the shortest paths from u_1 to u_{2l+2} for $1 \leq l \leq \lfloor (n-1)/2 \rfloor$ under this length assignment. The shortest path from u_1 to u_{2l+2} for each $1 \leq l \leq \lfloor (n-1)/2 \rfloor$ corresponds to a cycle C of size $2l+1$ for which $(2l+1) r_{\max} - \sum_{v \in C} r_v$ is minimum. Thus C is precisely the cycle that maximizes $\sum_{v \in C} r_v$ among the cycles of length $2l+1$. Thus the oracle can be implemented in $O^*(nm)$ time.

Now from Corollary 6.2.12, there is an algorithm that computes $\exp(\omega)$-approximate primal and dual solutions by making $O^*(n\omega^{-2})$ oracle calls. Similar to the discussion for the maximum cut problem in Section 6.2.4, we can show that the dual solution $\{x_u \mid u \in V\}$ (after scaling) forms an $\exp(\omega)$ approximation to the linear program 6.16. Since each oracle takes $O^*(nm)$ time, the overall running time of the algorithm is $O^*(n^2 m \omega^{-2})$.

6.3 The BLOCK-ANGULAR PACKING or COVERING problems

In Section 5.2, we considered some special cases of the MIXED problem. These special cases included the PACKING (min-max resource sharing) and the COVERING (max-min resource sharing) problems earlier considered by Grigoriadis & Khachiyan [32, 33], Grigoriadis et al. [35] and Jansen & Zhang [48]. They also consider special cases of PACKING and COVERING problems called BLOCK-ANGULAR PACKING and COVERING. In this section, we derive algorithms for these special cases that are more efficient than those given by Grigoriadis & Khachiyan [32, 33] and Grigoriadis et al. [35]. We first motivate these problems by considering the concurrent flow problem.

6.3.1 The maximum concurrent flow problem

Consider the maximum concurrent flow problem defined as follows. We are given a (directed or undirected) graph $G = (V, E)$ with edge-capacities $c_e > 0$ for $e \in E$ and t pairs of vertices (u_l, v_l) for $l \in [t]$. Each pair (u_l, v_l) is associated with a demand $d_l > 0$ and a commodity l. The maximum concurrent flow problem is to maximize $\lambda > 0$ such that λd_l units of flow of lth commodity can be routed between u_l and v_l for all $l \in [t]$ *simultaneously*, while respecting the edge capacities. We can formulate this problem as a linear program. We use a notation similar to the maximum multicommodity flow problem discussed in Section 6.1. Let us associate variables $x_p \geq 0$ giving the amount of flow on path p between u_l and v_l for some $l \in [t]$. Let $\mathcal{P}$ denote the set of paths between all t pairs. The maximum concurrent flow problem can now be formulated as follows.

$$\begin{array}{lrcll} \max & & \lambda & & \\ \text{s.t.} & \sum_{p\sim l} x_p & \geq & \lambda d_l & \text{for } l \in [t] \\ & \sum_{p:e\in p} x_p & \leq & c_e & \text{for } e \in E \\ & x_p & \geq & 0 & \text{for } p \in \mathcal{P} \end{array} \tag{6.20}$$

Here $p \sim l$ denotes a path p between u_l and v_l. An equivalent way of formulating this problem is as follows. Find a way of routing d_l units of flow between u_l and v_l for $l \in [t]$ simultaneously such that the maximum *relative congestion* of any edge is minimized. The relative congestion is defined as the ratio of the total flow through the edge to the capacity of the edge.

$$\begin{array}{llcll} \min & \max_{e\in E} \frac{\sum_{p:e\in p} x_p}{c_e} & & & \\ \text{s.t.} & \sum_{p\sim l} x_p & = & d_l & \text{for } l \in [t] \\ & x_p & \geq & 0 & \text{for } p \in \mathcal{P} \end{array} \tag{6.21}$$

Similar to the maximum multicommodity flow problem, we have that the optimum value of the formulation (6.21) is λ^* if and only if the optimum value of the formulation (6.20) is $1/\lambda^*$. There is also a correspondence between the feasible solutions of the two formulations via scaling. Therefore the two formulations are "equivalent". Now the formulation (6.21) can be thought of as an instance of the PACKING problem as follows. For $l \in [t]$, define a polytope $P^l = \{(x_p : p \sim l) \geq 0 \mid \sum_{p\sim l} x_p = d_l\}$ as the polytope of flows of d_l units between u_l and v_l. Define $P = P^1 \times \cdots \times P^t$ as the Cartesian product of these polytopes. Now for $x \in P$ and $e \in E$, define the function

$$f_e(x) = \frac{\sum_{p:e\in p} x_p}{c_e}.$$

Thus the formulation (6.21) is equivalent to the PACKING problem of finding an $x \in P$ such that $\max_{e\in E} f_e(x)$ is minimized. From Corollary 5.2.2, we can find an $\exp(\omega)$ approximation by making $O(m\omega^{-2} \log m)$ calls to an oracle that given $\{y_e \geq$

$0 \mid e \in E\}$, finds an $x \in P$ such that

$$\sum_{e \in E} y_e f_e(x) = \sum_{e \in E} y_e \left(\frac{\sum_{p: e \in p} x_p}{c_e} \right) = \sum_{l \in [t]} \sum_{e \in E} \left(\frac{y_e}{c_e} \right) \sum_{p: e \in p, p \sim l} x_p$$

is minimized. Here we assume that any single path does not go between two distinct pairs of vertices. This oracle decomposes naturally into t independent oracles, one for each $l \in [t]$, that finds an $x^l \in P^l$ such that

$$\sum_{e \in E} \left(\frac{y_e}{c_e} \right) \sum_{p: e \in p, p \sim l} x_p = \sum_{p \sim l} x_p \left(\sum_{e \in p} \frac{y_e}{c_e} \right)$$

is minimized. Such an $x^l \in P^l$ is obtained by setting $x_p = d_l$ for the path p that is shortest under the length function y_e/c_e. Thus the oracle can be implemented by doing t shortest-path computations, one for each $l \in [t]$. Thus to obtain an $\exp(\omega)$ approximation to the maximum concurrent flow problem, we need $O(mt\omega^{-2} \log m)$ shortest-path computations.

Note that the functions f_e defined above can be written as $f_e(x) = \sum_{l=1}^{t} f_e^l(x^l)$ where $x = (x^1, \ldots, x^t) \in P^1 \times \cdots \times P^t$ and $f_e^l : P^l \to \mathbb{R}_+$ are functions defined as

$$f_e^l(x^l) = \frac{\sum_{p: e \in p, p \sim l} x_p}{c_e}.$$

We consider exactly these problems and present faster algorithms for them.

In particular, for the maximum concurrent flow problem, we present an algorithm that computes an $\exp(\omega)$ approximation by making $O((m+t)\omega^{-2} \log(m+t))$ shortest-path computations.

6.3.2 The BLOCK-ANGULAR PACKING problem

In this section, we consider a special form of the PACKING problem (Corollary 5.2.2). Suppose that the given convex set P is the Cartesian product of the form $P = P^1 \times \cdots \times P^t$ where $P^1, \ldots, P^t$ are non-empty convex sets. Suppose also that the functions f_i have the following form. For $x = (x^1, \ldots, x^t) \in P$, we have

$$f_i(x) = \sum_{l=1}^{t} f_i^l(x^l) \quad \text{for } i \in [m],$$

where $f_i^l : P^l \to \mathbb{R}_+$ are convex functions. The BLOCK-ANGULAR PACKING problem is to compute

$$\min_{x \in P} \max_{1 \le i \le m} \sum_{l=1}^{t} f_i^l(x^l).$$

To solve the above problem as a PACKING problem, we need an oracle that given $y \in \mathbb{R}_+^m$, computes $x = (x^1, \ldots, x^t) \in P$ such that

$$\left\langle y, \sum_{l=1}^{t} f^l(x^l) \right\rangle = \sum_{l=1}^{t} \langle y, f^l(x^l) \rangle$$

is minimized. Note that the oracle decomposes into t independent oracles that minimize $\langle y, f^l(x^l) \rangle$ over $x^l \in P^l$. Thus from Corollary 5.2.3, we can get an $\exp(\omega)$ approximation (to the primal and the dual) by making a total of $O(mt\omega^{-2} \log m)$ calls to individual oracles. In this section, we reduce this to $O((m+t)\omega^{-2} \log(m+t))$.

Feasibility version

We first consider the *feasibility* version of the BLOCK-ANGULAR PACKING problem. Given an "estimate" α on the optimum value, the feasibility version of the BLOCK-ANGULAR PACKING problem is to find an $x \in P$ (if it exists) such that

$$\max_{1 \le i \le m} \sum_{l=1}^{t} f_i^l(x^l) \le \alpha.$$

For the feasibility version, we assume that we have a feasibility oracle which given $y \in \mathbb{R}_+^m$, $l \in [t]$, and $\beta \ge 0$, computes an $x^l \in P^l$ (if it exists) such that $\langle \underline{y}, f^l(x^l) \rangle \le \beta$.

Reduction to the SUBSTITUTE MIXED problem

We now show how to reduce the feasibility version of the BLOCK-ANGULAR PACKING problem to the SUBSTITUTE MIXED problem.

We call an $l \in [t]$ as "redundant" if there exists an $x^l \in P^l$ such that $f_i^l(x^l) = 0$ for all $i \in [m]$. A redundant l can be identified by making one oracle call to check if $\min_{x^l \in P^l} \langle \mathbf{1}, f^l(x^l) \rangle = 0$. Note that a redundant index l can be removed from the given instance, since in any feasible solution, we can set x^l such that $f_i^l(x^l) = 0$ for all $i \in [m]$. In what follows, we assume that there are no redundant indices $l \in [t]$.

Let us now introduce t^2 functions $g_{l'}^l$ for $l, l' \in [t]$ as follows. Define functions $g_{l'}^l : P^l \to \mathbb{R}_+$ as

$$g_{l'}^l(x^l) = \begin{cases} \alpha, & \text{if } l = l'; \\ 0, & \text{otherwise.} \end{cases}$$

Note that each $g_{l'}^l$ is a constant function. Now consider the SUBSTITUTE MIXED problem with the mt functions f_i^l and the t^2 functions $g_{l'}^l$ over the polytopes P^l. The SUBSTITUTE MIXED problem is to find $x^l \in P^l$ for $l \in [t]$ and $\pi \in \Pi$ such that

$$\max_{i \in [m]} \sum_{l=1}^{t} \pi^l f_i^l(x^l) \le \min_{l' \in [t]} \sum_{l=1}^{t} \pi^l g_{l'}^l(x^l).$$

From the definition of the functions $g_{l'}^l$, this is equivalent to

$$\max_{i \in [m]} \sum_{l=1}^{t} \pi^l f_i^l(x^l) \le \alpha \cdot \min_{l \in [t]} \pi^l.$$

We now argue that for any feasible solution, we have $\min_{l \in [t]} \pi^l \neq 0$. Assume to the contrary that $\min_{l \in [t]} \pi^l = 0$. In this case, the left-hand-side of the above expression is zero. Since $\sum_{l=1}^{t} \pi^l = 1$, there is at least one $l \in [t]$ such that $\pi^l \neq 0$. For such an l, we have $f_i^l(x^l) = 0$ for all $i \in [m]$. This implies that l is redundant which is a contradiction.

Thus $\min_{l \in [t]} \pi^l \neq 0$. Now, after dividing by $\min_{l \in [t]} \pi^l$, we get

$$\max_{i \in [m]} \sum_{l=1}^{t} f_i^l(x^l) \le \max_{i \in [m]} \sum_{l=1}^{t} \left(\frac{\pi^l}{\min_{l' \in [t]} \pi^{l'}} \right) f_i^l(x^l) \le \alpha.$$

Thus if $x^l \in P^l, l \in [m]$ and $\pi \in \Pi$ form a feasible solution to the above SUBSTITUTE MIXED problem, then $x^l \in P^l, l \in [t]$ form a feasible solution to the given BLOCK-ANGULAR PACKING problem. On the other hand, if $x^l \in P^l, l \in [t]$ form a feasible solution to the given BLOCK-ANGULAR PACKING problem, then $x^l \in P^l, l \in [t]$ and $\pi = \mathbf{1}/m$ form a feasible solution to the above SUBSTITUTE MIXED problem.

Oracle

To complete the reduction, we have to show that the oracle for the instance of the SUBSTITUTE MIXED problem defined above can be obtained from the given oracle for the BLOCK-ANGULAR PACKING problem.

The oracle for the SUBSTITUTE MIXED problem, given $y, z \geq 0$, $l \in [t]$, and $\beta > 0$, needs to find an $x^l \in P^l$ (if it exists) such that

$$\langle \underline{y}, f^l(x^l) \rangle \leq \beta \cdot \langle \underline{z}, g^l(x^l) \rangle.$$

From the definitions of the functions $g^l_{l'}$, this is equivalent to

$$\langle \underline{y}, f^l(x^l) \rangle \leq \beta\alpha \cdot z_l / \langle \mathbf{1}, z \rangle.$$

Such an $x^l \in P^l$ (if it exists) can be found by making one call to the oracle with β replaced by $\beta\alpha \cdot z_l / \langle \mathbf{1}, z \rangle$. This completes the reduction.

Thus we can use the algorithm for the SUBSTITUTE MIXED problem to solve the feasibility version of the BLOCK-ANGULAR PACKING problem. Now Theorem 6.2.1 implies the following.

Theorem 6.3.1 *There exists an algorithm for the feasibility version of the* BLOCK-ANGULAR PACKING *problem that given* $\alpha > 0$ *and an error parameter* $\omega \in (0, 1)$,

- *either computes* $(x^1, \ldots, x^t) \in P^1 \times \cdots \times P^t$ *such that*

$$\max_{i \in [m]} \sum_{l=1}^{t} f^l_i(x^l) \leq \exp(\omega) \cdot \alpha,$$

- *or proves that there does not exist* $(x^1, \ldots, x^t) \in P^1 \times \cdots \times P^t$ *such that*

$$\max_{i \in [m]} \sum_{l=1}^{t} f^l_i(x^l) \leq \alpha.$$

The algorithm makes $O((m + t)\omega^{-2} \log(m + t))$ *calls to an oracle that given* $y \geq 0$, $l \in [t]$, *and* $\beta > 0$, *finds an* $x^l \in P^l$ *(if it exists) such that* $\langle \underline{y}, f^l(x^l) \rangle \leq \beta$ *and takes additional time equal to the time to compute* $f^l(x^l)$ *for some* $x^l \in P^l, l \in [t]$ *between successive oracle calls.*

Optimization version

To solve the optimization version of the BLOCK-ANGULAR PACKING problem, we reduce it to the feasibility version. Given an instance of the optimization version, we first compute lower and upper bounds on the optimum value OPT that are within factor m. Let $y = \mathbf{1}/m \in \mathbb{R}^m_+$. Let $\Lambda_l = \min_{x^l \in P^l} \langle y, f^l(x^l) \rangle$. We can compute Λ_l by a single oracle call. Note that $\Lambda_l \leq \min_{x^l \in P^l} \max_{i \in [m]} f^l_i(x^l) \leq m \cdot \Lambda_l$. Therefore

$$\sum_{l=1}^{t} \Lambda_l \leq \min_{x \in P} \max_{i \in [m]} \sum_{l=1}^{t} f^l_i(x^l) \leq m \sum_{l=1}^{t} \Lambda_l.$$

Thus we have lower and upper bounds on OPT that are within a factor m of each other. Now using ideas similar to the one used in Section 3.1.2, we can reduce the optimization problem to a sequence of $O(\log\log m)$ feasibility problems with $\omega' = 1/2$ and $O(\log(1/\omega))$ feasibility problems where ith-to-last subproblem has $\omega' = \Omega(\omega(\frac{3}{2})^i)$. The idea again is to do a bisection search with $\omega' = 1/2$ till we get lower and upper bounds within a constant factor (first phase) and then geometrically reduce ω' (i.e., increase precision) and the ratio of lower and upper bounds (second phase). Since the ω' decreases geometrically in the second phase, the time taken is dominated by that for the last feasibility problem. Thus to find an $\exp(\omega)$ approximation to OPT, we make $O((m+t)\log(m+t)(\log\log m + \omega^{-2}))$ oracle calls.

The maximum concurrent flow problem revisited

The maximum concurrent flow problem is an optimization version of the BLOCK-ANGULAR PACKING problem. From the above discussion, we can get an $\exp(\omega)$ approximation by doing $O((m+t)\log(m+t)(\log\log m + \omega^{-2}))$ shortest-path computations. Note that in order to compute lower and upper bounds on OPT that are within a factor m of each other, we just need to do t shortest-path computations. In contrast, Fleischer [20] does t maximum bottleneck path computations each of which takes $O(m \log m)$ time.

Karakostas [49] improved this running time to $O^*(m\omega^{-2})$ shortest-path computations by grouping the commodities with a common source. Here we briefly describe how to obtain this result by reformulating the concurrent flow problem as a BLOCK-ANGULAR PACKING problem. Consider an instance of the maximum concurrent flow problem with pairs (u_l, v_l) and demands $d_l > 0$ for $l \in [t]$. We again consider a formulation similar to (6.21) in which we try to minimize the maximum congestion of any edge while routing d_l units of flow of commodities $l \in [t]$. Let S be the set of distinct sources u_l. Note that $|S| \le n$ where n is the number of nodes in the graph. For $u \in S$, define a u-*star* to be a collection of u_l-v_l paths with the source node $u_l = u$. Different paths in a u-star may share edges. Let $\mathcal{T}_u$ be the set of all such u-stars. Now define a polytope $P^u = \{(x_T : T \in \mathcal{T}_u) \ge 0 \mid \sum_{T \in \mathcal{T}_u} x_T = 1\}$. The variable x_T denotes the fraction of flow of the commodities with source at u that is routed along the u-star T. Thus for such a commodity $l \in [t]$ with $u_l = u$, a total flow of $x_T \cdot d_l$ units is routed along the u_l-v_l path in T. For $T \in \mathcal{T}_u$ and $e \in E$, let $\mathcal{C}(e,T) \subseteq [t]$ be the subset of commodities $l \in [t]$ such that $u_l = u$ and the edge e occurs on the u_l-v_l path in T. Now for $x \in P^u$ and $e \in E$, define $f_e^u(x)$ to be the total relative congestion on the edge e due to the flows of all commodities with source at u. Formally, for $x^u = (x_T : T \in \mathcal{T}_u) \in P^u$, we define the function f_e^u as

$$f_e^u(x^u) = \frac{1}{c_e} \sum_{T \in \mathcal{T}_u} \left(x_T \sum_{l \in \mathcal{C}(e,T)} d_l \right). \tag{6.22}$$

Let P be the Cartesian product of P^u for $u \in S$. It is now easy to see that the formulation (6.21) is equivalent to

$$\min_{x=(x^u : u \in S) \in P} \max_{e \in E} \sum_{u \in S} f_e^u(x^u).$$

Starting with a feasible solution to the above formulation, we can decompose the flow into paths and get a feasible solution to (6.21) with the same value. On the other hand, starting with a feasible solution to (6.21), we can decompose the flows from a source node $u \in S$ into u-stars and get a feasible solution to the above formulation with the same value.

Now the oracle to solve the above BLOCK-ANGULAR PACKING problem, when given $\{y_e \ge 0 \mid e \in E\}$ and $u \in S$ needs to find $x \in P^u$ such that $\sum_{e \in E} y_e f_e^u(x)$ is minimized. We can rewrite this expression as follows.

$$\sum_{e \in E} y_e f_e^u(x) \quad = \quad \sum_{T \in \mathcal{T}_u} \left(x_T \sum_{e \in E} \left(\frac{y_e}{c_e} \sum_{l \in \mathcal{C}(e,T)} d_l \right) \right)$$

$$= \sum_{T\in\mathcal{T}_u} \left(x_T \sum_{l:u_l=u} \left(d_l \sum_{e:l\in\mathcal{C}(e,T)} \frac{y_e}{c_e} \right) \right).$$

Since $\sum_{T\in\mathcal{T}_u} x_T = 1$, the above expression is minimized if we set $x_T = 1$ for a u-star T such that the paths in T corresponding to the commodities $l \in [t]$ with $u_l = u$ are the shortest u_l-v_l paths under the length function $y_e/c_e \geq 0$. Observe that all such paths can be computed in the same asymptotic running time as computing a shortest path from u to one specified node. But now we have at most $n \leq m$ convex set P^u. Thus by an analysis similar to the one given above, we can compute an $\exp(\omega)$ approximation by doing $O(m \log m(\log\log m + \omega^{-2}))$ shortest-path computations. Note, however, that the algorithm outputs an *implicit* representation of the flow. By implicit representation, we mean that the output is the collection of flow-paths (grouped with respect to a common source) with the amount of flow routed through each of them. We refer the reader to Karakostas [49] for more discussion on implicit and explicit representations of the flow.

Bienstock & Iyengar [9, 8] recently presented an algorithm that computes a $(1+\omega)$ approximation to packing problems by solving $O(\omega^{-1}\sqrt{Kn})$ separable convex quadratic programs, where K is the maximum number of non-zeros per row and n is the number of variables. For the special case of the maximum concurrent flow problem with rational capacities and demands, they obtain an algorithm that computes a $(1+\omega)$-optimal flow by doing $O(\omega^{-1}t^{3/2}m\sqrt{n}(\log(1/\omega) + L_U + L_D))$ shortest-path computations, where t is the number of commodities, m is the number of edges, n is the number of nodes, and L_U and L_D are, respectively, the number of bits needed to store the capacities and demands. This is the first algorithm that needs $O^*(\omega^{-1})$ iterations.

The minimum cost concurrent flow problem

The minimum cost concurrent flow problem is defined as follows. Along with the input of the maximum concurrent flow problem, we are given edge-costs $w_e \geq 0$ for routing unit flow through $e \in E$ and a budget B on the total cost. The objective is to maximize the minimum relative demand of any commodity routed such that the capacity constraints are satisfied and the total cost of routing the flow is at most B. In the feasibility version, given $\lambda > 0$, the objective is to check if one can route λd_l units of demand of each commodity $l \in [t]$ such that the capacity and the budget constraints are satisfied. We think of λd_l as the new demand for commodity l.

We now show how to formulate this feasibility version as a BLOCK-ANGULAR PACKING problem. Define functions f_e^u as given in (6.22). We define additional functions f_b^u to capture the budget constraint as follows.

$$f_b^u(x^u) = \sum_{T\in\mathcal{T}_u} \left(x_T \sum_{l:u_l=l} \left(d_l \sum_{e:l\in\mathcal{C}(e,T)} \frac{w_e}{B} \right) \right).$$

Here the subscript b stands for the budget constraint. Note that $f_b^u(x^u)$ denotes the fraction of the budget needed to route the flow from the source u. Thus the total flow $x = (x^u : u \in S)$ satisfies the budget constraint if $\sum_{u\in S} f_b^u(x^u) \leq 1$. Therefore, the feasibility version of the minimum cost concurrent flow problem is equivalent to finding $x \in P$ such that

$$\begin{aligned} \sum_{u\in S} f_e^u(x^u) &\leq 1 \qquad \text{for } e \in E, \\ \sum_{u\in S} f_b^u(x^u) &\leq 1. \end{aligned}$$

This is clearly a feasibility version of the BLOCK-ANGULAR PACKING problem with $m+1$ constraints. Now the oracle, when given $\{y_e \geq 0 \mid e \in E\}$, y_b, and $u \in S$, needs to compute $x^u \in P^u$ such that

$$y_b f_b^u(x^u) + \sum_{e \in E} y_e f_e^u(x) = \sum_{T \in \mathcal{T}_u} \left(x_T \sum_{l: u_l = u} \left(d_l \sum_{e: l \in \mathcal{C}(e,T)} \left(\frac{y_e}{c_e} + \frac{y_b w_e}{B} \right) \right) \right)$$

is minimized. This can again be done by setting $x_T = 1$ for a u-star T such that the paths in T corresponding to the commodities $l \in [t]$ with $u_l = u$ are the shortest u_l-v_l paths under the length function $(y_e/c_e + y_b w_e/B) \geq 0$. Thus in time $O^*(m^2\omega^{-2})$, we can compute an implicit representation of a flow that violates the capacity and budget constraints by a factor of at most $\exp(\omega)$. Similar running times were also obtained by Garg & Könemann [31], Fleischer [20], Young [84], and Karakostas [49]. These results also extend to concurrent flow problems in lossy networks as defined in [49].

6.3.3 The BLOCK-ANGULAR COVERING problem

The BLOCK-ANGULAR COVERING problem can also be solved using similar techniques.

Theorem 6.3.2 *There exists an algorithm for the feasibility version of the* BLOCK-ANGULAR COVERING *problem that given* $\alpha > 0$ *and an error parameter* $\omega \in (0, 1)$*,*

- *either computes* $(x^1, \ldots, x^t) \in P^1 \times \cdots \times P^t$ *such that*

$$\min_{j \in [k]} \sum_{l=1}^{t} g_j^l(x^l) \geq \exp(-\omega) \cdot \alpha,$$

- *or proves that there does not exist* $(x^1, \ldots, x^t) \in P^1 \times \cdots \times P^t$ *such that* $\min_{j \in [k]} \sum_{l=1}^{t} g_j^l(x^l) \geq \alpha$.

The algorithm makes $O((k+t)\omega^{-2}\log(k+t))$ *calls to an oracle which given* $z \geq 0$, $l \in [t]$, *and* $\alpha > 0$, *finds an* $x^l \in P^l$ *(if it exists) such that* $\langle \underline{z}, g^l(x^l) \rangle \geq \alpha$ *and takes additional time equal to the time to compute* $g^l(x^l)$ *for some* $x^l \in P^l, l \in [t]$ *between successive oracle calls.*

The reduction of the optimization version to the above feasibility version is analogous to the one for the BLOCK-ANGULAR PACKING problem.

6.4 Summary of key ideas developed in this chapter

We crucially use the fact that approximate oracles can be used to obtain approximate solutions. We present the idea of cycling through the convex sets one by one. Such "round-robin increments" were first used by Fleischer [20] to obtain a running time for the maximum multicommodity flow problem that is independent of the number of commodities. Subsequently, this technique was used by Young [84] for the MIXED-LP problem and Fleischer [21] for covering linear programs with upper bounds on the variables.

Chapter 7

Linear programming problems

Linear programs form an important special case of the mathematical programming problems considered in this thesis. In fact most of the work in designing Lagrangian relaxation algorithms was motivated by various linear programming problems including the multicommodity flow problems. Since the linear functions are both convex and concave, the abstract frameworks developed in previous chapters can be applied directly to the linear programs. In this chapter, we first consider the linear fractional PACKING or COVERING problems considered earlier by Plotkin et al. [70], Luby & Nisan [64], Young [83], and Garg & Könemann [31]. Using the PACKING and COVERING framework developed in Chapters 5 and 6, we obtain approximation algorithms for these problems. We then consider the above problems when the instances are given *explicitly*. In Section 7.3, we show that under some assumptions, the above techniques can be extended to the linear programs that may have *negative entries* in the constraint matrix or in the objective function. In Section 7.3.1, we apply these techniques to the linear fractional COVERING problem with upper bounds on the variables and obtain a slight improvement to results of Fleischer [21]. The COVERING problems with variable upper bounds can be thought of as a special case of the MIXED-LP problem considered by Young [84] (see Section 5.1.1). However in this chapter, we will be able to find *strictly* feasible solutions for this problem.

7.1 Linear fractional PACKING problem

The linear fractional PACKING problem is as defined below.

Linear fractional PACKING:

$$\begin{array}{rrcl} \max & c^\top x & & \\ \text{s.t.} & Ax & \leq & b \\ & x & \geq & 0 \end{array} \tag{7.1}$$

where A is an $m \times t$ matrix with *non-negative* entries and $c, b \geq 0$ are vectors with *non-negative* entries.

In order to solve the linear fractional PACKING problem, we assume that we have the following *min-ratio-column* or *column-generation* oracle.

Given $y \in \mathbb{R}^m_+$, the oracle finds a column $l \in [t]$ of A that minimizes

$$\frac{1}{c_l} \sum_{i=1}^{m} \frac{y_i A_{il}}{b_i}. \tag{7.2}$$

Note that if $c_l = 0$ for some $l \in [t]$, then without loss of generality, we can set $x_l = 0$. While if $b_i = 0$ for some $i \in [m]$, then any feasible solution has $x_l = 0$ for all $l \in [t]$ for which $A_{il} > 0$. After setting all such forced variables and removing redundant constraints, we get an "equivalent" reduced system in which all entries in c and b are positive. Note that the oracle (7.2) for the new instance can be easily obtained from the oracle given for the original instance.

Since the vectors c and b have positive entries, the optimum value of the above linear program is positive. We now show that this problem can be thought of as a special case of the PACKING problem defined in Corollary 5.2.2. Define a polytope P and functions $f_i : P \to \mathbb{R}_+$ for $i \in [m]$ as follows.

$$P = \{x \in \mathbb{R}^t_+ \mid c^\top x = 1\} \qquad \text{and} \qquad f_i(x) = \frac{A_i x}{b_i} \text{ for } i \in [m], \tag{7.3}$$

where A_i denotes the ith row of the matrix A. The PACKING problem is then to find an $x \in P$ that minimizes $\max_{i\in[m]} f_i(x)$. We argue that solving the linear program (7.1) is "equivalent" to solving this instance of the PACKING problem. For $x \in P$, let $\lambda(x) = \max_{i\in[m]} f_i(x)$. Note that the vector $\hat{x} = x/\lambda(x)$ satisfies $A\hat{x} \le b, \hat{x} \ge 0$ and has the objective value $c^\top \hat{x} = 1/\lambda(x)$. On the other hand, for any $\hat{x} \ge 0$ that satisfies $A\hat{x} \le b$ and has the objective value $\mu(\hat{x}) = c^\top \hat{x}$, the vector $x = \hat{x}/\mu(\hat{x})$ satisfies $x \in P$ and $\lambda(x) = 1/\mu(\hat{x})$. Thus, given an error parameter $\omega \in (0, 1)$, obtaining an $x \in P$ with $\lambda(x) \le \exp(\omega) \cdot \lambda^*$ is equivalent to obtaining an $\hat{x} \ge 0$ satisfying $A\hat{x} \le b$ such that $\mu(\hat{x}) \ge \exp(-\omega) \cdot \mu^*$ where λ^* and μ^* are the optimum values of the PACKING problem (7.3) and the given linear fractional PACKING problem (7.1) respectively.

Oracle

Recall that the oracle needed to solve the PACKING problem is: given $y \in \mathbb{R}^m_+$, find $\min_{x\in P} \langle y, f(x)\rangle$. This can be obtained from the oracle (7.2) as follows. Since $\langle y, f(x)\rangle$ is a linear function of $x \in P$ where P is a polytope, the minimum occurs at a vertex of P. The vertices of P are of the form $x = (0, \ldots, 0, 1/c_l, 0, \ldots, 0)$ where the only non-zero entry is $1/c_l$ at the lth coordinate for some $l \in [t]$. Thus $\min_{x\in P} \langle y, f(x)\rangle = \min_{l\in[t]} \frac{1}{c_l} \sum_{i=1}^m \frac{y_i A_{il}}{b_i}$. Such an x can be found directly from the oracle (7.2). Thus the reduction is complete and from Corollary 5.2.2, we get the following theorem.

Theorem 7.1.1 *There is an algorithm which given an error parameter $\omega \in (0, 1)$, computes an $x \ge 0$ that forms a feasible solution to the linear fractional* PACKING *problem and satisfies*

$$c^\top x \ge \exp(-\omega) \cdot \lambda^*$$

where λ^* is the optimum value. The algorithm makes $O(m\omega^{-2} \log m)$ calls to the oracle (7.2), where m denotes the number of rows in A.

The linear programming dual of the given linear fractional PACKING problem is the following linear fractional COVERING problem.

$$\begin{array}{rrcl} \min & b^\top y & & \\ \text{s.t.} & A^\top y & \ge & c \\ & y & \ge & 0 \end{array} \tag{7.4}$$

In Corollary 5.2.3, we observed that the algorithm for the PACKING problem also solves the dual problem near-optimally. Similarly, the algorithm for the linear fractional PACKING also produces a solution to the dual (7.4).

Theorem 7.1.2 *There is an algorithm that given an error parameter $\omega \in (0, 1)$, computes an $\hat{x} \ge 0$ and $\hat{y} \ge 0$ that form feasible solutions to (7.1) and (7.4) respectively such that*

$$c^\top \hat{x} \ge \exp(-\omega) \cdot b^\top \hat{y}.$$

The algorithm makes $O(m\omega^{-2}\log m)$ *calls to the oracle (7.2), where* m *denotes the number of rows in* A.

Proof. Corollary 5.2.3 applied to the PACKING instance (7.3) implies that with the given number of oracle calls, we can compute $x \in P$ and $y \geq 0$ such that $\lambda(x) \leq \exp(\omega) \cdot d(y)$ where

$$\lambda(x) \doteq \max_{i\in[m]} \frac{A_i x}{b_i} \quad \text{and} \quad d(y) \doteq \min_{x\in P} \sum_{i=1}^{m} \frac{\underline{y}_i A_i x}{b_i} = \min_{l\in[t]} \sum_{i=1}^{m} \frac{\underline{y}_i A_{il}}{b_i c_l}.$$

The last equality holds since the minimum occurs at a vertex of the polytope P. Now define $\hat{x} = x/\lambda(x)$ and $\hat{y} = (\hat{y}_1,\ldots,\hat{y}_m)$ where $\hat{y}_i = \underline{y}_i/(d(y)b_i)$ for $i \in [m]$. Note that $A\hat{x} \leq b$. Also we have $\sum_{i=1}^{m} \frac{\underline{y}_i A_{il}}{b_i c_l} \geq d(y)$ for $l \in [t]$. Therefore $(A^\top \hat{y})_l = \sum_{i=1}^{m} \frac{\underline{y}_i A_{il}}{d(y) b_i} \geq c_l$ for $l \in [t]$. Thus $A^\top \hat{y} \geq c$. Now since $x \in P$, we have $c^\top \hat{x} = c^\top x/\lambda(x) = 1/\lambda(x)$. Further, we have $b^\top \hat{y} = \sum_{i=1}^{m} b_i \hat{y}_i = \sum_{i=1}^{m} \underline{y}_i/d(y) = 1/d(y)$. Since $\lambda(x) \leq \exp(\omega)\cdot d(y)$, we have $c^\top \hat{x} \geq \exp(-\omega)\cdot b^\top \hat{y}$. Thus the proof is complete. ∎

7.2 Linear fractional COVERING problem

The results analogous to those presented in Section 7.1 also for the linear fractional COVERING problem.

Linear fractional COVERING:

$$\begin{array}{rrcl} \min & c^\top x & & \\ \text{s.t.} & Ax & \geq & b \\ & x & \geq & 0 \end{array} \tag{7.5}$$

where A is an $k \times t$ matrix with *non-negative* entries and $c, b \geq 0$ are vectors with *non-negative* entries.

In order to solve the linear fractional COVERING problem, we assume that we have the following *max-ratio-column* subroutine.

Given $z \in \mathbb{R}^k_+$, the oracle finds a column $l \in [t]$ of A that maximizes

$$\frac{1}{c_l} \sum_{j=1}^{k} \frac{z_j A_{jl}}{b_j}. \tag{7.6}$$

Note that if $c_l = 0$ for some $l \in [t]$, then by setting $x_l = \max_{j:A_{jl}>0} b_j/A_{jl}$, we can satisfy each constraint $(Ax)_j \geq b_j$ with $A_{jl} > 0$. Thus we can remove these constraints and the variable x_l. Note that the constraints $j \in [k]$ with $A_{jl} = 0$ do not get affected by removal of x_l. If $b_j = 0$ for some $j \in [k]$, then the constraint $A_j x \geq b_j$ is trivially satisfied by any $x \geq 0$ and can be ignored. Thus we can again assume, without loss of generality, that all entries in c and b are positive.

Now using Corollary 5.2.5, we have the following theorem.

Theorem 7.2.1 *There is an algorithm that given an error parameter* $\omega \in (0,1)$*, computes an* $\hat{x} \geq 0$ *and* $\hat{z} \geq 0$ *that form feasible solutions to (7.5) and its dual* $\max\{b^\top z \mid A^\top z \leq c, z \geq 0\}$ *respectively such that*

$$c^\top \hat{x} \leq \exp(\omega) \cdot b^\top \hat{z}.$$

The algorithm makes $O(k\omega^{-2}\log k)$ *calls to the oracle (7.6), where* k *denotes the number of rows in* A.

Similar result holds also for the MIXED-LP problem considered by Young [84] as discussed in Section 5.1.1.

7.2.1 Explicitly given instances

Now we address the linear PACKING or COVERING problems in which the matrix A and vectors c and b are given *explicitly*. Let D denote the number of non-zero entries in A. Note that the oracles (7.2) and (7.6) can be implemented in $O(D)$ time by computing the expression to be minimized or maximized for each column. Thus the overall running time to find an $\exp(\omega)$ approximation is $O(mD\epsilon^{-2}\log m)$. However, using the SUBSTITUTE MIXED framework developed in Chapter 6, we can reduce this running time to $O((md+D)\epsilon^{-2}\log m)$ where d is the maximum number of constraints any variable appears in. Such a result was first proved by Young [84].

Consider the PACKING problem formulated in (7.3). Let us introduce variables $\pi^1,\ldots,\pi^t$ defined as $\pi^l = c_l x_l$. Now $x \in P$ implies that $\pi = (\pi^1,\ldots,\pi^t) \in \Pi$ where $\Pi = \{(\pi^1,\ldots,\pi^t) \geq 0 \mid \sum_{l=1}^t \pi^l = 1\}$. Thus the problem in (7.3) is equivalent to finding $\pi \in \Pi$ such that

$$\max_{i\in[m]} \sum_{l=1}^{t} \pi^l \cdot \frac{A_{il}}{b_i c_l}$$

is minimized. Now this can be thought of as a special case of the SUBSTITUTE PACKING problem (see Corollary 6.2.5) as described below. For each column $l \in [t]$, let P^l be *any* non-empty convex set. Let $f_i^l : P^l \to \mathbb{R}_+$ be constant functions

$$f_i^l \equiv \frac{A_{il}}{b_i c_l}.$$

To solve this SUBSTITUTE PACKING problem, we need the following oracle. Given $y \in \mathbb{R}_+^m$ and $l \in [t]$, minimize $\langle y, f^l(x^l)\rangle$ over $x^l \in P^l$. Note that this oracle is trivial since f_i^l are constant functions. However just to evaluate this expression, one needs time proportional to the number of non-zero entries in that column. Since each column has at most d non-zero entries, any oracle call takes $O(d)$ time. Thus from Corollary 6.2.5, the total time taken to compute an $\exp(\omega)$ approximation is $O((m+t)d\omega^{-2}\log m)$. Since t is the number of variables and d is the maximum number of constraints that any variable appears in, we have $D \leq td$. A careful analysis shows that we can, in fact, replace the term td in the running time by D. We outline the argument here. In Section 6.2.2, Lemma 6.2.4 bounds the total number of oracle calls in the algorithm for the SUBSTITUTE MIXED problem. In this lemma, we bound the number of fruitful oracle calls by $O(m\epsilon^{-2}\log m)$ and the number of unfruitful oracle calls by $O(t\epsilon^{-2}\log m)$ (note that $k=1$ in this case). The bound on the number of unfruitful oracle calls follows from the fact that there are $O(\epsilon^{-2}\log m)$ phases (Lemma 6.2.3) and in each phase we do at most one unfruitful oracle call for each column $l \in [t]$. Now an oracle call for a column takes time proportional to the number of non-zero entries in that column. Since there are D non-zero entries in the entire matrix, the total time for all the unfruitful oracle calls in a single phase is $O(D)$. Thus the total time for all the unfruitful oracle calls is $O(D\epsilon^{-2}\log m)$. Note that such an argument does not hold for the fruitful oracle calls, since these oracle calls may not be equally distributed over the columns and the total time for the fruitful oracle calls may be bounded only by $O(md\epsilon^{-2}\log m)$ since d is the maximum number of non-zero entries in any column. Thus we have the following theorem.

Theorem 7.2.2 *There is an algorithm that given an error parameter $\omega \in (0,1)$, computes an $x \geq 0$ that forms a feasible solution for the* explicitly given *linear fractional* PACKING *problem and satisfies*

$$c^\top x \geq \exp(-\omega)\cdot\lambda^*$$

where λ^* is the optimum value. The algorithm takes $O((md+D)\epsilon^{-2}\log m)$ time, where m denotes the number of rows, d denotes the maximum number of constraints that any variable appears in and D denotes the number non-zero entries in A.

Similarly we have the following theorem for explicitly given linear fractional COVERING problems.

Theorem 7.2.3 *There is an algorithm that given an error parameter $\omega \in (0,1)$, computes an $x \geq 0$ that forms a feasible solution for the* explicitly given *linear fractional* COVERING *problem and satisfies*

$$c^\top x \leq \exp(\omega) \cdot \lambda^*$$

where λ^* is the optimum value. The algorithm takes $O((kd + D)\omega^{-2} \log k)$ time, where k denotes the number of rows, d denotes the maximum number of constraints that any variable appears in and D denotes the number non-zero entries in A.

An analogous result also holds for the explicitly given linear fractional MIXED problem that Young [84] considers. Although Young [84] claims a running time of $O((m + k)d\epsilon^{-2} \log(m + k))$ where m and k are the number of packing and covering constraints and d is the maximum number of constraints that any variable appears in, the running time of his algorithm is in fact $O(((m + k)d + D)\epsilon^{-2} \log(m + k))$. This matches our running time.

7.3 The linear programs with negative entries

The techniques developed for the linear fractional PACKING and COVERING problems rely crucially on the non-negativity of the constraint matrix. However in some cases, we can extend the argument to linear programs with negative entries in the constraint matrix and the objective function. Suppose, for example, that we would like to solve a linear program of the form

$$\begin{array}{rrcl} \max & c^\top x & & \\ \text{s.t.} & Ax & \leq & b \\ & x & \geq & 0 \end{array} \tag{7.7}$$

where the entries in A and c *may be negative*. Suppose also that we are aware that there is an optimum solution x^* to the above linear program such that $c^\top x^* \geq 0$ and $Ax^* \geq 0$. That is, the optimum value is non-negative and all the entries in Ax^* are non-negative. In such a case, we can add new constraints

$$\begin{array}{rcl} c^\top x & \geq & 0 \\ Ax & \geq & 0 \end{array}$$

to the above linear program without affecting the optimum solution. The resulting linear program can then be thought of as being equivalent to the following *semi-infinite* linear program that has infinitely many variables. We introduce the semi-infinite linear program only for conveying the intuition. We demonstrate an application of this technique more rigorously in Section 7.3.1.

$$\begin{array}{rrcl} \max & \zeta^\top \chi & & \\ \text{s.t.} & \mathcal{A}\chi & \leq & b \\ & \chi & \geq & 0 \end{array} \tag{7.8}$$

Here the vectors ζ, χ and the columns of the matrix $\mathcal{A}$ are indexed by candidate solutions $x \geq 0$ that satisfy $c^\top x \geq 0$ and $Ax \geq 0$. The matrix $\mathcal{A}$ has m rows. We let $\zeta_x = c^\top x$ and $\mathcal{A}_{ix} = A_i x$ for such an $x \geq 0$. Observe that, if x is a feasible solution to the linear program (7.7), then χ with $\chi_x = 1$ and $\chi_{x'} = 0$ for $x' \neq x$ forms a feasible solution to the linear program (7.8) with an equal objective value. On the other hand, if χ is a feasible solution to (7.8), then $\sum_x \chi_x \cdot x$ forms a feasible solution to (7.7) with an equal objective value. Thus the two linear programs are "equivalent". Now since the infinite linear program (7.8) has only non-negative entries, it can be thought of as an instance of the PACKING problem and can be solved using an algorithm for the PACKING problem.

7.3.1 The linear fractional COVERING programs with upper bounds on the variables

To illustrate how to solve linear programs with negative entries using the ideas described above, we consider the linear fractional COVERING problems with upper bounds on the variables.

$$\begin{array}{lrcl} \min & c^\top x & & \\ \text{s.t.} & Ax & \geq & b \\ & x & \leq & u \\ & x & \geq & 0 \end{array} \tag{7.9}$$

where $A \geq 0$ is an $n \times m$ matrix, $c, u \geq 0$ are m-dimensional vectors and $b \geq 0$ is an n-dimensional vector.

Such a linear program was first considered by Fleischer [21] who assumes the following *most-violated-constraint* oracle.

Given $x \in \mathbb{R}_+^m$, the oracle finds a row $j \in [n]$ of A that minimizes

$$\frac{(Ax)_j}{b_j} = \sum_{i=1}^{m} \frac{A_{ji} x_i}{b_j}. \tag{7.10}$$

Given an estimate $\alpha > 0$ on the optimum value, the problem of finding an $x \in \mathbb{R}_+^m$ such that $Ax \geq b$ and $c^\top x \leq \alpha, x \leq u$ can be thought of as a special case of the MIXED-LP problem considered in Section 5.1.1. However the algorithm for the MIXED-LP problem computes only an approximately feasible solution that violates either the packing or the covering constraints by a factor of $\exp(\omega)$. Here are objective is find a near-optimum *strictly feasible* solution, i.e., an $x \in \mathbb{R}_+^m$ such that $Ax \geq b, x \leq u$ and $c^\top x \leq \exp(\omega) \cdot \lambda^*$ where $\omega \in (0,1)$ is the given error parameter and λ^* is the optimum.

Fleischer [21] gives an algorithm to compute such a solution by making $O(m\omega^{-2} \log(mC))$ calls to the oracle (7.10) where $C = \frac{\max_{i\in[m]} c_i u_i}{\min_{i:c_i u_i > 0} c_i u_i}$.[1] In this section, we present an algorithm very similar to her algorithm and improve the number of oracle calls to $O(m\omega^{-2} \log m + \log\log C)$. Our algorithm falls out naturally from the algorithm for the PACKING problem given in Chapter 5.

Theorem 7.3.1 *There exists an algorithm that given an error parameter $\omega \in (0,1)$, either computes an $\exp(\omega)$ approximation to the linear program (7.9), or proves that the linear program (7.9) is infeasible. The algorithm makes $O(m\omega^{-2} \log m + \log\log C)$ calls to the oracle (7.10).*

Without loss of generality, we assume that all entries of c, u, and b are positive. If $c_i = 0$, we can reduce the problem by setting $x_i = u_i$ and replacing each b_j with $b_j - A_{ji} u_i$. If $u_i = 0$, then clearly $x_i = 0$ in any feasible solution and hence we can remove this variable. Similarly, if $b_j = 0$, then the jth covering constraint is trivially satisfied and can be removed. Without loss of generality, we can also assume that the objective function $c = \mathbf{1}$ is the vector of all ones. This can be achieved by working with new variables $x_i' = c_i x_i$ and replacing each A_{ji} with A_{ji}/c_i and replacing each u_i with $c_i u_i$. We also assume, without loss of generality, that $\min_{i\in[m]} u_i = 1$. This is achieved by dividing each entry of u and b by $\min_{i\in[m]} u_i$. Note that in doing this, it is not necessary to scale the vector c by $\min_{i\in[m]} u_i$, since that amounts to just scaling the optimum value. Observe that after this modification, we have $\max_{i\in[m]} u_i = C$. Note that the oracle (7.10) for the modified instance can be obtained from the given oracle for the original instance. We modify the instance as described above just for making the presentation simpler; the algorithm can modified easily to work with the original instance. Since b and c have only positive entries, the optimum value of the linear program (7.9) is *positive*.

[1] Fleischer [21] defines C as $\max_{i\in[m]} c_i / \min_{i:c_i>0} c_i$. However, the running time of her algorithm in fact depends on C as defined here. Note that this value of C is invariant under the scaling of variables.

Before starting to solve this linear program, we make a call to the oracle (7.10) with $x = u$ to find a row $j \in [n]$ that minimizes $(Au)_j/b_j$. If $(Au)_j < b_j$, then the jth covering constraint cannot be satisfied even if we set each variable x_i to its upper bound u_i. In this case, we output "infeasible". In the rest of the section, we assume that the given instance is feasible, i.e., $Au \geq b$.

Reduction to the PACKING problem

We now consider the dual of the linear program (7.9).

$$\begin{array}{lrcl} \max & b^\top y - u^\top z & & \\ \text{s.t.} & A^\top y - z & \leq & \mathbf{1} \\ & y, z & \geq & 0 \end{array} \qquad (7.11)$$

The dual linear program (7.11) has negative entries in the constraint matrix as well as in the objective function. However as shown below, this linear program satisfies the conditions discussed in the beginning of Section 7.3 and can be solved using the PACKING framework.

Lemma 7.3.2 *Any optimum solution (y^*, z^*) to the dual linear program (7.11) satisfies $b^\top y^* - u^\top z^* > 0$ and $A^\top y^* - z^* \geq 0$.*

Proof. Since the optimum of the original program (7.9) has a positive value, by linear programming duality, the optimum value $b^\top y^* - u^\top z^*$ of the dual linear program is also positive.

Now the constraints in the dual linear program imply that $z_i \geq 0$ and $z_i \geq (A^\top y)_i - 1$. To maximize the objective function, we would set z_i to the smallest possible value, i.e., $\max\{0, (A^\top y)_i - 1\}$. ∎

Thus we can add constraints $b^\top y - u^\top z > 0$ and $A^\top y - z \geq 0$ to the dual linear program without affecting its optimum value. The dual linear program after adding these constraints is given as follows.

$$\begin{array}{lrcl} \max & b^\top y - u^\top z & & \\ \text{s.t.} & A^\top y - z & \leq & \mathbf{1} \\ & b^\top y - u^\top z & > & 0 \\ & A^\top y - z & \geq & 0 \\ & y, z & \geq & 0 \end{array} \qquad (7.12)$$

We now define an instance of the PACKING problem (see Corollary 5.2.2) that is "equivalent" to the linear program (7.12). Define a polytope P and functions $f_i : P \to \mathbb{R}_+$ as follows.

$$\begin{array}{rcl} P & = & \{(y, z) \geq 0 \mid A^\top y - z \geq 0, b^\top y - u^\top z = 1\}, \\ f_i(y, z) & = & (A^\top y)_i - z_i \ \text{ for } i \in [m]. \end{array} \qquad (7.13)$$

Now the linear program (7.12) can be shown equivalent to the PACKING problem

$$\min_{(y,z) \in P} \max_{i \in [m]} f_i(y, z).$$

It is easy to see that the above PACKING problem has the optimum value λ if and only if the linear program (7.12) has the optimum value $1/\lambda$. The details are very similar to the one given in Section 7.1 and are omitted.

Oracle

To solve the PACKING problem (7.13), we need an oracle that given $x \in \mathbb{R}_+^m$, computes $(y,z) \in P$ such that $x^\top(A^\top y - z)$ is minimum. Recall that in the PACKING algorithm, we make oracle calls with $x \in \mathbb{R}_+^m$ such that $x_i > 0$ for all $i \in [m]$. Define

$$R(x) := \min_{(y,z)\in P} x^\top(A^\top y - z).$$

Since $b^\top y - u^\top z = 1$ for $(y,z) \in P$, we have

$$R(x) = \min\left\{\frac{x^\top(A^\top y - z)}{b^\top y - u^\top z} \mid y,z \geq 0, A^\top y - z \geq 0, b^\top y - u^\top z > 0\right\}.$$

From Corollary 5.2.4, we know that it is enough to compute $R(x)$ approximately. Now we show how to find (y,z) that achieve an approximate value of $R(x)$.

For $x \in \mathbb{R}_+^m$ and $\alpha \geq 0$, define $(x|_\alpha) \in \mathbb{R}_+^m$ as the "truncated" vector with $(x|_\alpha)_i = \min\{x_i, \alpha u_i\}$ for $i \in [m]$. Define

$$r_\alpha(x) = \min_{j\in[n]} \frac{(A(x|_\alpha))_j}{b_j}.$$

Note that we can compute $r_\alpha(x)$ using the oracle (7.10). We now prove some useful properties of the function $r_\alpha(x)$.

Lemma 7.3.3 *For a fixed $x \in \mathbb{R}_+^m$, the function $r_\alpha(x)$ is a non-decreasing and concave function of α.*

Proof. It is easy to see that r_α is a non-decreasing function of α. Now for $i \in [m]$, note that $\min\{x_i, \alpha u_i\}$ is a concave function of α. Therefore if $\alpha = \lambda_1\alpha_1 + \lambda_2\alpha_2$, for some $\alpha, \alpha_1, \alpha_2, \lambda_1, \lambda_2 \geq 0$ with $\lambda_1 + \lambda_2 = 1$, we have $(x|_\alpha) \geq \lambda_1(x|_{\alpha_1}) + \lambda_2(x|_{\alpha_2})$. Therefore, for any $j \in [n]$, we have $(A(x|_\alpha))_j \geq \lambda_1(A(x|_{\alpha_1}))_j + \lambda_2(A(x|_{\alpha_2}))_j$. Therefore,

$$\min_{j\in[n]} \frac{(A(x|_\alpha))_j}{b_j} \geq \lambda_1 \min_{j\in[n]} \frac{(A(x|_{\alpha_1}))_j}{b_j} + \lambda_2 \min_{j\in[n]} \frac{(A(x|_{\alpha_2}))_j}{b_j}$$

and the proof is complete. ∎

Lemma 7.3.4 *For any $x \in \mathbb{R}_+^m$ with $x_i > 0$ for $i \in [m]$, we have $r_{R(x)}(x) = R(x)$.*

Proof. Let $y,z \geq 0$ be such that $A^\top y - z \geq 0, b^\top y - u^\top z > 0$ and

$$R(x) = \frac{x^\top(A^\top y - z)}{b^\top y - u^\top z}.$$

It is no loss of generality to assume that the above minimum is achieved for $\langle \mathbf{1}, y\rangle = 1$; this can be achieved by scaling y, z. Recall that $R(x)$ is the minimum possible value of this fraction. Note that the coefficient of z_i in the numerator is x_i, while its coefficient in the denominator is u_i. If $x_i/u_i > R(x)$, the variable z_i must be set to its highest possible value $(A^\top y)_i$; otherwise by increasing z_i, we can decrease the value of the fraction. Observe that if we increase z_i, the value of $b^\top y - u^\top z$ may reach zero. But in this case, since $x_i/u_i > R(x)$, by increasing z_i the numerator $x^\top(A^\top y - z)$ reaches zero *before* the denominator. Thus we have $x^\top(A^\top y - z) = 0$ and $b^\top y - u^\top z > 0$ for some $y, z \geq 0$. Since $x_i > 0$ for $i \in [m]$, we have $A^\top y - z = 0$. This, in turn, implies that the linear program (7.11) is unbounded and the linear program (7.9) is infeasible which is a contradiction.

Now if $x_i/u_i \leq R(x)$, we can without loss of generality set $z_i = 0$. Since in this case, by decreasing z_i, the value of the fraction does not increase. Thus without loss of generality, the vector z is of the form

$$z_i = \begin{cases} (A^\top y)_i, & \text{if } x_i > R(x) \cdot u_i; \\ 0, & \text{otherwise.} \end{cases} \tag{7.14}$$

Now since $x^\top(A^\top y - z) = R(x) \cdot (b^\top y - u^\top z)$, we have $x^\top A^\top y - (x^\top - R(x) \cdot u^\top)z = R(x) \cdot b^\top y$.

Claim 7.3.1 *If z satisfies $z_i = (A^\top y)_i$ if $x_i > \alpha \cdot u_i$, and 0 otherwise, then we have $x^\top A^\top y - (x^\top - \alpha \cdot u^\top)z = (x|_\alpha)^\top A^\top y$.*

Proof of Claim 7.3.1. We show that $x_i(A^\top y)_i - (x_i - \alpha \cdot u_i)z_i = (x|_\alpha)_i(A^\top y)_i$ holds for each $i \in [m]$. If $x_i - \alpha \cdot u_i \leq 0$, we have $z_i = 0$ and $(x|_\alpha)_i = x_i$, hence this inequality holds. On the other hand, if $x_i - \alpha \cdot u_i > 0$, we have $z_i = (A^\top y)_i$ and $x_i(A^\top y)_i - (x_i - \alpha \cdot u_i)z_i = (\alpha \cdot u_i)(A^\top y_i) = (x|_\alpha)_i(A^\top y)_i$. ∎

Thus we have $(x|_{R(x)})^\top A^\top y = R(x) \cdot b^\top y$, i.e.,

$$\frac{(x|_{R(x)})^\top A^\top y}{b^\top y} = R(x).$$

Further we can assume that y is of the following form.

$$\begin{aligned} y_j &= 1 \quad \text{for a row } j \in [n] \text{ such that } ((x|_{R(x)})^\top A^\top)_j / b_j \text{ is minimum}, \\ y_{j'} &= 0 \quad \text{for } j' \neq j. \end{aligned} \tag{7.15}$$

This is true because by replacing y with the vector as defined above, the value of the ratio does not increase. Therefore indeed we have $r_{R(x)}(x) = R(x)$ and the proof of Lemma 7.3.4 is complete. ∎

Lemma 7.3.5 *We have $r_\alpha(x) < \alpha$ if and only if $\alpha > R(x)$.*

Proof. Since the function r_α is concave and $r_0(x) = 0, r_{R(x)} = R(x)$, we have $r_\alpha(x) \geq \alpha$ for $0 \leq \alpha \leq R(x)$. Therefore $r_\alpha(x) < \alpha$ implies $\alpha > R(x)$.

Now let $\alpha > R(x)$. Again using the concavity of r_α, we have $r_\alpha(x) \leq \alpha$. We now show that the strict inequality holds. Let (y, z) be the vectors as given in (7.15) and (7.14) that achieve the minimum ratio $R(x)$. Note that we have $(A^\top y)_i - z_i > 0$ for some $i \in [m]$; otherwise the argument in the proof of Lemma 7.3.4 implies that the linear program (7.9) is infeasible. For such $i \in [m]$, we have $(A^\top y)_i > 0$ and $z_i = 0$. This implies $A_{ij} > 0$ and $x_i \leq R(x) \cdot u_i$. Note that $x_i \leq R(x) \cdot u_i$ implies $(x|_\alpha)_i = (x|_{R(x)})_i$. This combined with the fact that $A_{ij} > 0$ implies that $(A(x|_\alpha))_j / b_j < \alpha / R(x) \cdot (A(x|_{R(x)}))_j / b_j = \alpha / R(x) \cdot R(x) = \alpha$. Since this holds for all $j \in [n]$ that attain the minimum in (7.15), we have $r_\alpha(x) < \alpha$. ∎

Now from Lemmas 7.3.3 and 7.3.4, the function r_α must be of the form given in Figure 7.1. Note however that we may have $r_\alpha(x) = \alpha$ in the range $0 \leq \alpha \leq R(x)$. Thus $R(x)$ is the largest value of α such that $r_\alpha(x) = \alpha$.

In the PACKING algorithm, we initialize $x = \mathbf{1}$. Below we show how to compute the value of $R(\mathbf{1})$ approximately.

Computing $R(\mathbf{1})$ approximately

Lemma 7.3.6 *We have $R(\mathbf{1}) \geq 1/C$. Further if $R(\mathbf{1}) \geq 1$, then $R(\mathbf{1}) = \min_{j \in [n]}(A\mathbf{1})_j / b_j$.*

Proof. Recall that $\min_{i \in [m]} u_i = 1$ and $\max_{i \in [m]} u_i = C$. Since the linear program (7.9) is feasible, we have $\min_{j \in [n]}(Au)_j / b_j \geq 1$. This implies that

$$R(u) = \min_{(y,z)} \frac{u^\top A^\top y - u^\top z}{b^\top y - u^\top z} \geq 1$$

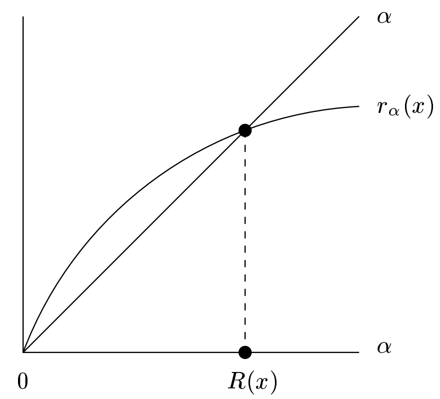

Figure 7.1: The function r_α. We may have $r_\alpha(x) = \alpha$ in the range $0 \leq \alpha \leq R(x)$.

where the minimum is taken over $(y, z) \geq 0$ such that $u^\top A^\top y - u^\top z \geq 0$ and $b^\top y - u^\top z > 0$. The above inequality holds since to minimize the ratio, one has to set z to zero and we get $R(u) = \min_{j \in [n]} (Au)_j / b_j \geq 1$. Since $C \cdot \mathbf{1} \geq u$, we have $R(C \cdot \mathbf{1}) \geq R(u) \geq 1$. Therefore $R(\mathbf{1}) = R(C \cdot \mathbf{1})/C \geq 1/C$.

Now assume $R(\mathbf{1}) \geq 1$. Since $u \geq \mathbf{1}$, we have $\min\{1, R(\mathbf{1}) \cdot u_i\} = 1$ for each $i \in [m]$. Therefore $(\mathbf{1}|_{R(\mathbf{1})}) = \mathbf{1}$. From Lemma 7.3.4, we have $R(\mathbf{1}) = r_{R(\mathbf{1})}(\mathbf{1})$. Therefore, $R(\mathbf{1}) = \min_{j \in [n]} (A\mathbf{1})_j / b_j$ as desired. ∎

Now we describe a simple procedure to compute an $\exp(\epsilon)$ approximation to $R(\mathbf{1})$. We set $\epsilon = \omega/6$ where $\omega \in (0, 1)$ is the given error parameter. We first compute $r_1(\mathbf{1})$. If $r_1(\mathbf{1}) \geq 1$, Lemma 7.3.5 implies that $R(\mathbf{1}) \geq 1$. In this case, Lemma 7.3.6 gives that $R(\mathbf{1}) = \min_{j \in [n]} (A\mathbf{1})_j / b_j$ and we can compute this value by doing one oracle call.

On the other hand, if $r_1(\mathbf{1}) < 1$, then we have $1/C \leq R(\mathbf{1}) < 1$. In such a case, we have lower and upper bounds on $R(\mathbf{1})$ within a factor C. We do a bisection search similar to the one given in Section 3.1.2. We set $\alpha = \sqrt{1/C}$ and compute $r_\alpha(\mathbf{1})$. If $r_\alpha(\mathbf{1}) \geq \alpha$, Lemma 7.3.5 implies that $R(\mathbf{1}) \in [\alpha, 1]$. On the other hand, if $r_\alpha(\mathbf{1}) < \alpha$, we get that $R(\mathbf{1}) \in [1/C, \alpha]$. In either cases we have reduced the ratio of lower and upper bounds from C to $\sqrt{C}$. By repeating this $O(\log \log C + \log(1/\epsilon))$ times, we can reduce the ratio of upper and lower bounds to $\exp(\epsilon)$. Thus we can compute α such that $\exp(-\epsilon) \cdot \alpha \leq R(\mathbf{1}) < \alpha$.

Lemma 7.3.7 *Given $x \in \mathbb{R}^m_+$ and α such that $\exp(-\epsilon) \cdot \alpha \leq R(x) < \alpha$, we can compute $(y, z) \in P$ such that $x^\top (A^\top y - z) \leq \exp(\epsilon) \cdot R(x)$ by making one call to the oracle (7.10).*

Proof. We use an argument similar to the one given in the proof of Lemma 7.3.4. We compute $j \in [n]$ such that $(A(x|_\alpha))_j / b_j$ is minimum. We, then, set $y_j = 1$ and $y_{j'} = 0$ for $j' \neq j$. Therefore we have $(x|_\alpha)^\top A^\top y \leq r_\alpha(x) \cdot b^\top y$. We also set

$$z_i = \begin{cases} (A^\top y)_i, & \text{if } x_i > \alpha \cdot u_i; \\ 0, & \text{otherwise.} \end{cases}$$

Now Claim 7.3.1 implies that $x^\top A^\top y - (x^\top - \alpha \cdot u^\top) z = (x|_\alpha)^\top A^\top y$. Hence $x^\top A^\top y - (x^\top - \alpha \cdot u^\top) z \leq r_\alpha(x) \cdot b^\top y$. This, in turn, implies $x^\top (A^\top y - z) \leq r_\alpha(x) \cdot b^\top y - \alpha \cdot u^\top z < \alpha(b^\top y - u^\top z)$. The strict inequality follows from $r_\alpha(x) < \alpha$ (Lemma 7.3.5) and $b^\top y > 0$. Therefore we have $b^\top y - u^\top z > 0$ and

$$\frac{x^\top (A^\top y - z)}{b^\top y - u^\top z} < \alpha \leq \exp(\epsilon) \cdot R(x).$$

∎

Using the above lemma, we can compute $(y, z) \in P$ that achieve an $\exp(\epsilon)$ approximation to $R(\mathbf{1})$.

Algorithm

In the PACKING algorithm, we initialize $x = \mathbf{1}$. During the algorithm, the values of x_i never decrease. Therefore, the value of

$$R(x) = \min_{(y,z)\in P} x^\top (A^\top y - z)$$

also never decreases. As explained above we compute an α such that $\exp(-\epsilon) \cdot \alpha \leq R(x) < \alpha$. Using Lemma 7.3.7, we compute $(y, z) \in P$ that forms an $\exp(\epsilon)$ approximation to the oracle call and update the "dual variables" x. Note that the value of $R(x)$ does not decrease as we update the dual variables. We repeat this till we find that $R(x) \geq \alpha$. We can detect this by checking if $r_\alpha(x) \geq \alpha$ (Lemma 7.3.5). In such a case, we update the value of α as $\alpha := \exp(\epsilon) \cdot \alpha$ and repeat.

Let α_0 be the value of α in the beginning of the algorithm. We say that the algorithm is in phase $p \geq 0$ if the current value of α is $\alpha_0 \exp(p\epsilon)$. In phase p, for the current value of x and α, we compute an $\exp(\epsilon)$ approximation to the oracle call using Lemma 7.3.7. We continue this till the **Stop-condition** of the PACKING algorithm becomes true.

Bounding the number of oracle calls

Call an oracle call in which we do not find the desired $(y, z) \in P$ as "unfruitful". After each unfruitful oracle call, we update the value of α and go into the next phase. Hence the number of unfruitful oracle calls to (7.10) is at most the number of phases in the algorithm. The following lemma proves a bound on the number of phases in the algorithm.

Lemma 7.3.8 *The algorithm has $O(\epsilon^{-2} \log m)$ phases.*

Proof. The algorithm initializes $x = \mathbf{1}$ and terminates when for some $i \in [m]$, we have $x_i \geq \exp(\epsilon\Gamma)$ where $\Gamma = O(\epsilon^{-2} \log m)$ (see procedure **Update** in Algorithm 5.3). Thus the maximum value α can take is $\alpha_0 \exp(\epsilon\Gamma) = \alpha_0 \exp(\epsilon^{-1} \log m)$. Since the value of α increases by $\exp(\epsilon)$ in each phase, the number of phases in the algorithm is $O(\epsilon^{-2} \log m)$. ∎

Thus the number of unfruitful oracle calls is $O(\epsilon^{-2} \log m)$. Call an oracle call in which we find $(y, z) \in P$ satisfying the desired inequality as "fruitful". Clearly the number of fruitful oracle calls is equal to the number of oracle calls in the PACKING algorithm and from Corollary 5.2.4, this is $O(m\epsilon^{-2} \log m)$. In the pre-processing, to compute an approximate value of $R(\mathbf{1})$, we make $O(\log \log C + \log(1/\epsilon))$ oracle calls. Thus the total number of calls to the oracle (7.10) in the algorithm is $O(m\epsilon^{-2} \log m + \log \log C + \log(1/\epsilon)) = O(m\omega^{-2} \log m + \log \log C)$.

Proving feasibility and near-optimality of the solutions

Now from Corollary 5.2.4, we obtain $x \geq 0$ and $(y, z) \in P$ such that

$$\lambda(y, z) \leq \sigma \cdot \exp(3\epsilon) \cdot d(x) \tag{7.16}$$

where $\sigma = \exp(\epsilon)$ and

$$\lambda(y, z) = \max_{i\in[m]} \Big((A^\top y)_i - z_i\Big) \qquad \text{and} \qquad d(x) = \frac{r_\alpha(x)}{\sum_{i=1}^m (x|_\alpha)_i}$$

where α is the upper bound on $R(x)$ that we work with in the corresponding phase. Since we work with an approximate oracle, the dual $d(x)$ is defined in terms of the approximate solution $(x|_\alpha)$.

Recall that the algorithm (see (5.7) in Corollary 5.2.3) outputs the best dual solution found in any round

$$x = x^{r-1} \text{ where } r = \arg\max_{1 \le r \le N} d(x^{r-1}).$$

We modify the algorithm so that it outputs the best dual solution computed at the *end of a phase*. The following lemma shows that we do not lose in the quality of the dual solution by a factor more than $\exp(\epsilon)$ by restricting the dual solution in this manner.

Lemma 7.3.9 *Let p be the phase in which the algorithm computes the best dual solution, say x_1. Let x_2 be the dual solution at the end of the phase $p-1$. We have $d(x_2) \ge \exp(-\epsilon) \cdot d(x_1)$.*

Proof. Let $\alpha_2 = \alpha_0 \exp((p-1)\epsilon)$ is the upper bound on R that we work with in phase $p-1$ and $\alpha_1 = \alpha_0 \exp(p\epsilon)$ be the upper bound in phase p. We have

$$\begin{aligned}
d(x_2) &= \frac{r_{\alpha_2}(x_2)}{\sum_{i=1}^m (x_2|_{\alpha_2})_i} \\
&\ge \frac{\alpha_2}{\sum_{i=1}^m (x_2|_{\alpha_2})_i} && \text{(since the oracle call with } x_2 \text{ was unfruitful)} \\
&\ge \frac{\alpha_2}{\sum_{i=1}^m (x_1|_{\alpha_1})_i} && \text{(since } x \text{ does not decrease, we get } (x_2|_{\alpha_2}) \le (x_1|_{\alpha_1})) \\
&= \exp(-\epsilon) \cdot \frac{\alpha_1}{\sum_{i=1}^m (x_1|_{\alpha_1})_i} \\
&\ge \exp(-\epsilon) \cdot \frac{r_{\alpha_1}(x_1)}{\sum_{i=1}^m (x_1|_{\alpha_1})_i} && \text{(since the oracle call with } x_1 \text{ was fruitful)} \\
&= \exp(-\epsilon) \cdot d(x_1).
\end{aligned}$$

∎

Since the algorithm computes the best dual solution x at the end of any phase, the inequality (7.16) and Lemma 7.3.9 gives

$$\lambda(y, z) \le \exp(5\epsilon) \cdot d(x). \tag{7.17}$$

Lemma 7.3.10 *Let α be the upper bound in the phase in which the dual solution x was computed. The vectors $x^* = (x|_\alpha)/\alpha$ and and $(y^*, z^*) = (y, z)/\lambda(y, z)$ form feasible solutions to the primal linear program (7.9) and the dual linear program (7.11) respectively. Furthermore,*

$$\mathbf{1}^\top x^* \le \exp(\omega) \cdot \left(b^\top y^* - u^\top z^*\right).$$

Thus x^* and (y^*, z^*) form near-optimum solutions to (7.9) and (7.11) respectively.

Proof. We have $(x|_\alpha) \le \alpha u$. Further since x was the solution at the end of a phase, we have $\min_{j \in [n]} A_j (x|_\alpha)/b_j = r_\alpha(x) \ge \alpha$. Therefore the vector $x^* = (x|_\alpha)/\alpha$ forms a primal feasible solution.

From the definition of $\lambda(y, z)$, it is clear that $A^\top y^* - z^* \le \mathbf{1}$ so that (y^*, z^*) forms a feasible solution to (7.11). Now we have

$$\begin{aligned}
\mathbf{1}^\top x^* &= \frac{\sum_{i=1}^m (x|_\alpha)_i}{\alpha} \\
&= \frac{r_\alpha(x)}{d(x) \cdot \alpha} && \text{(from the definition of } d(x))
\end{aligned}$$

$$
\begin{aligned}
&\leq \frac{\exp(\epsilon)}{d(x)} \quad \text{(since } r_\alpha(x) \leq \exp(\epsilon)\cdot\alpha \text{ at the end of any phase)}\\
&\leq \exp(6\epsilon)\cdot\frac{1}{\lambda(y,z)} \quad \text{(from (7.17))}\\
&= \exp(\omega)\cdot\frac{b^\top y - u^\top z}{\lambda(y,z)} \quad \text{(since } (y,z)\in P \text{ and } \omega = 6\epsilon\text{)}\\
&= \exp(\omega)\cdot\Big(b^\top y^* - u^\top z^*\Big).
\end{aligned}
$$

Therefore the proof is complete. ∎

Thus we have proved Theorem 7.3.1. This technique of updating α in multiples of $\exp(\epsilon)$ is similar to the one used by Fleischer to reduce the number of oracle calls in the maximum multicommodity flow problem [20] and also in [21]. This technique was also used in the SUBSTITUTE MIXED algorithm in Chapter 6.

7.4 Summary of key ideas developed in this chapter

Although the algorithms for the PACKING and COVERING problems rely crucially on the non-negativity of the functions, we can sometimes solve the linear PACKING or COVERING problems with negative entries provided we can restrict the set of feasible solutions so that the functions take non-negative values and design the necessary oracles.

Chapter 8

Conclusion and open problems

In this thesis, we presented a unified framework for Lagrangian relaxation based algorithms for various classes of convex programming problems. We presented essentially a single exponential potential based algorithm that can be modified slightly for different problems to match the state-of-the-art results for various problems. We have shown that there are really only a handful of ideas — exponential penalties for violating constraints, reducing the width, variable-size increments, round-robin increments, etc. — that appear again and again in these algorithms. Although most of these techniques were already known in the literature, the art of designing new algorithms was still restricted to experts. One of our objectives was to remove this "artistry" as far as possible. Of course, it is not possible to remove artistry from algorithm-designing completely and there is always a scope of providing more insights and designing better and better algorithms.

As such there are a lot of directions for future research. One of the main open problems is to prove lower bounds on the number of iterations required in any Dantzig-Wolfe algorithm that accesses the problem only through the given oracles. Klein & Young [56] proved such a lower bound. However their bound applies only in a very restricted setting.

A main disadvantage of most of the current algorithms is that their running time grows proportional to ϵ^{-2}. It is a very challenging open problem to reduce this dependence on the error parameter ϵ. Bienstock & Iyengar [9, 8] recently presented an algorithm for the packing problem that has a ϵ^{-1} dependence. However, in each iteration they have to solve a quadratic optimization problem. Are there any potential functions, other than exponential or logarithmic, that one can use to improve running times?

In Chapters 3 and 7, we pointed out some techniques for handling "negative entries" in the constraints. However, these techniques seem to be very limited in their applicability. There is still not a satisfactory way of taking care of negative entries.

Solving linear programs forms an important first step in designing approximation algorithms for many combinatorial optimization problems. Various tools developed in this thesis can be used in solving these linear programs. Can we use these tools in designing "integral" approximate solutions for the original problems, thus bypassing linear programs altogether? The congestion minimization problems like job-scheduling, packet-routing, and packing edge-disjoint paths may form good starting problems.

Bibliography

[1] R. Ahuja, T. Magnati, and J. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, Eaglewood Cliffs, NJ, 1993.

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

[3] I. Althöfer. On sparse approximations to randomized strategies and convex combinations. *Linear Algebra and its Applications*, 199:339–356, 1994.

[4] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. Schapire. Gambling in a rigged casino: the adversarial multi-armed bandit problem. In *Proceedings, IEEE Symposium on Foundations of Computer Science*, pages 322–331, 1995.

[5] F. Barahona. The max-cut problem on graphs not contractible to K_5. *Operation Research Letters*, 2(3):107–111, 1983.

[6] F. Barahona. On some weakly bipartite graphs. *Operation Research Letters*, 2(5):239–242, 1983.

[7] D. Bienstock. *Potential Function Methods for Approximately Solving Linear Programming Problems: Theory and Practice*. Kluwer Academic Publishers, Boston, 2002. An early version also appears as CORE Lecture Series Monograph, Core, UCL, Belgium (2001) (download from www.core.ucl.ac.be).

[8] D. Bienstock and G. Iyengar. Concurrent flows in $O^*(1/\epsilon)$ iterations. Technical Report TR-2003-03, Columbia University, 2003.

[9] D. Bienstock and G. Iyengar. Concurrent flows in $O^*(1/\epsilon)$ iterations. In *Proceedings, ACM Symposium on Theory of Computing*, 2004. To appear.

[10] A. Blum, V. Kumar, A. Rudra, and F. Wu. Online learning in online auctions. In *Proceedings, ACM-SIAM Symposium on Discrete Algorithms*, pages 202–204, 2003.

[11] G. Brown and J. Neumann. Solutions of games by differential equations. In *John von Neumann: Collected Works*, volume VI, pages 38–43. Pergamon Press, Oxford, 1963. Reprinted from Ann. Math Studies No. 24 pp. 73–79 (1950).

[12] N. Cesa-Bianchi, Y. Freund, D. Helmbold, D. Haussler, R. Schapire, and M. Warmuth. How to use expert advice. *Journal of the Association for Computing Machinery*, 44(3):427–485, 1997.

[13] V. Chvátal. On certain polytopes associated with graphs. *J. Combin. Theory Ser. B*, 18:138–154, 1975.

[14] T. Cover. Universal portfolios. *Mathematics of Finance*, 1(1):1–29, 1991.

[15] G. Dantzig. Maximization of a linear function of variables subject to linear inequalities. In *Activity Analysis of Production and Allocation (Tj. C. Koopmans, ed.)*, pages 339–347, 1951.

[16] A. Dawid. Statistical theory: the prequential approach. *Journal of the Royal Statistical Society, Series A*, 147:278–292, 1984.

[17] J. Edmonds. Edge-disjoint branchings. In R. Rustin, editor, *Combinatorial Algorithms*, pages 91–96. Algorithmic Press, New York, 1972.

[18] F. Eisenbrand, S. Funke, N. Garg, and J. Könemann. A combinatorial algorithm for computing a maximum independent set in a t-perfect graph. In *Proceedings, ACM-SIAM Symposium on Discrete Algorithms*, pages 517–522, 2003.

[19] F. Ergun, R. Sinha, and L. Zhang. An improved FPTAS for restricted shortest path. *Inform. Process. Lett.*, 83:287–291, 2002.

[20] L. Fleischer. Approximating fractional multicommodity flow independent of the number of commodities. *SIAM J. Discrete Math.*, 13:505–520, 2000.

[21] L. Fleischer. A fast approximation scheme for fractional covering problems with variable upper bounds. In *Proceedings, ACM-SIAM Symposium on Discrete Algorithms*, pages 994–1003, 2004.

[22] J. Fonlupt, A. Mahjoub, and J. Uhry. Composition of graphs and the bipartite subgraph polytope. Technical Report 459, Laboratoire ARTEMIS (IMAG), Université de Grenoble, 1984.

[23] D. Foster. Prediction in the worst case. *The Annuls of Statistics*, 19(2):1084–1090, 1991.

[24] L. Fratta, M. Gerla, and L. Kleinrock. The flow deviation method: An approach to store-and-forward communication network design. *Networks*, 3:97–133, 1973.

[25] Y. Freund and R. Schapire. Game theory, on-line prediction and boosting. In *Proceedings of the 9th Annual Conference on Computational Learning Theory*, pages 325–332, 1996.

[26] Y. Freund and R. Schapire. A decision-theoretic generalization of online learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

[27] Y. Freund and R. Schapire. Adaptive game playing using multiplicative weights. *Games and Economic Behavior*, 29:79–103, 1999.

[28] D. Fudenberg and D. Levine. Consistency and cautious fictitious play. *Journal of Economic Dynamics and Control*, 19:1065–1089, 1995.

[29] M. Garey and D. Johnson. *Computers and Intractability: A guide to the theory of NP-completeness*. W. H. Freeman and Company, San Francisco, 1979.

[30] N. Garg and R. Khandekar. Fast approximation algorithms for fractional steiner forest and related problems. In *Proceedings, IEEE Symposium on Foundations of Computer Science*, pages 500–509, 2002.

[31] N. Garg and J. Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In *Proceedings, IEEE Symposium on Foundations of Computer Science*, pages 300–309, 1998.

[32] M. Grigoriadis and L. Khachiyan. Coordination complexity of parallel price-directive decomposition. *Math. Oper. Res.*, 21:321–340, 1994.

[33] M. Grigoriadis and L. Khachiyan. Fast approximation schemes for convex programs with many blocks and coupling constraints. *SIAM J. Optimization*, 4(1):86–107, 1994.

[34] M. Grigoriadis and L. Khachiyan. A sublinear-time randomized approximation algorithm for matrix games. Technical Report 94-16, Department of Computer Science, Rutgers University, 1994.

[35] M. Grigoriadis, L. Khachiyan, L. Porkolab, and J. Villavicencio. Approximate max-min resource sharing for structured concave optimization. Technical Report DCS-TR-374, Rutgers University, 1999.

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

[37] M. Grötschel and W. Pulleyblank. Weakly bipartite graphs and the max-cut problem. *Operation Research Letters*, 1:23–27, 1981.

[38] B. Guenin. A characterization of weakly bipartite graphs. In *Proceedings, MPS Conference on Integer Programming and Combinatorial Optimization*, pages 9–22, 1998.

[39] B. Guenin. A characterization of weakly bipartite graphs. *J. Combin. Theory Ser. B*, 83:112–168, 2001.

[40] F. Hadlock. Finding a maximum cut of planar graph in polynomial time. *SIAM J. Comput.*, 4:221–225, 1975.

[41] R. Hassin. Approximation schemes for the restricted shortest path problems. *Math. Oper. Res.*, 17(1):36–42, 1992.

[42] M. Held and R. Karp. The traveling-salesman and minimum cost spanning trees. *Oper. Res.*, 18:1138–1162, 1970.

[43] M. Held and R. Karp. The traveling-salesman and minimum cost spanning trees: Part II. *Math. Programming*, 1:6–25, 1971.

[44] D. Helmbold, R. Schapire, Y. Singer, and M. Warmuth. On-line portfolio selection using multiplicative updates. In *International Conference on Machine Learning*, pages 243–251, 1996.

[45] K. Jansen. Personal communication, 2004.

[46] K. Jansen. Improved approximation algorithms for the general max-min resource sharing and fractional covering problem. Unpublished manuscript, 2004.

[47] K. Jansen and L. Porkolab. On preemptive resource constrained scheduling: polynomial-time approximations schemes. In *Proceedings, MPS Conference on Integer Programming and Combinatorial Optimization*, pages 329–343, 2002.

[48] K. Jansen and H. Zhang. Approximation algorithms for general packing problems with modified logarithmic potential function. In *Theoret. Comput. Sci.*, pages 255–266, 2002.

[49] G. Karakostas. Faster approximation schemes for fractional multicommodity flow problems. In *Proceedings, ACM-SIAM Symposium on Discrete Algorithms*, pages 166–173, 2002.

[50] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4:373–395, 1984.

[51] R. Karp. Reducibility among combinatorial problems. In R. Miller and J. Thatcher, editors, *Complexity of computer computations*, pages 85–103. Plenum Press, New York, 1956.

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

[53] L. Khachiyan. A polynomial algorithm in linear programming. *Soviet Mathematics Doklady*, 20:191–194, 1979.

[54] V. Klee and G. Minty. How good is the simplex algorithm? In O. Shisha, editor, *Inequalities, III*, pages 159–175. Academic Press, New York, 1972.

[55] 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. *SIAM J. Comput.*, 23(3):466–487, 1994.

[56] P. Klein and N. Young. On the number of iterations for Dantzig-Wolfe optimization and packing-covering approximation algorithms. In *Proceedings, MPS Conference on Integer Programming and Combinatorial Optimization*, pages 320–327, 1999.

[57] J. Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. Master's thesis, Computer Science, Universität des Saarlandes, 2002.

[58] S. Krantz. The fundamental theorem of calculus along curves. In *Handbook of Complex Variables*, page 22. Birkhäuser, Boston, MA, 1999.

[59] H. Kuhn and A. Tucker. Review of 'A numerical method for determination of the value and the best strategies of a zero-sum two-person game with large numbers of strategies', by John von Neumann. In *John von Neumann: Collected Works*, volume VI, pages 96–97. Pergamon Press, Oxford, 1963.

[60] F. Leighton and S. Rao. An approximate max-flow min-cut theorem for uniform multicommodity flow problems with application to approximation algorithms. In *Proceedings, IEEE Symposium on Foundations of Computer Science*, pages 422–431, 1988.

[61] T. Leighton, F. Makedon, S. Plotkin, C. Stein, S. Tragoudas, and E. Tardos. Fast approximation algorithms for multicommodity flow problems. *J. Comput. System Sci.*, 50(2):228–243, 1995.

[62] R. Lipton and N. Young. Simple strategies for large zero-sum games with applications to complexity theory. In *Proceedings, ACM Symposium on Theory of Computing*, pages 734–740, 1994.

[63] N. Littlestone and M. Warmuth. The weighted majority algorithm. *Information and Computation*, 108(2):212–261, 1994.

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

[65] J. Neumann. Zur theorie der Gesellschaftspiel. *Mathematische Annalen*, 100:295–320, 1928.

[66] E. Ordentlich and T. Cover. On-line portfolio selection. In *Proceedings of the 9th Annual Conference on Computational Learning Theory*, pages 310–313, 1996.

[67] J. Orlin. A faster strongly polynomial minimum cost flow algorithm. *Oper. Res.*, 4:338–350, 1993.

[68] M. Padberg and M. Rao. The Russian method for linear programming III: Bounded integer programming. Technical Report 81-39, New York University, 1981.

[69] S. Plotkin, D. Shmoys, and E. Tardos. Fast approximation algorithms for fractional packing and covering problems. In *Proceedings, IEEE Symposium on Foundations of Computer Science*, pages 495–504, 1991.

[70] S. Plotkin, D. Shmoys, and E. Tardos. Fast approximation algorithms for fractional packing and covering problems. *Math. Oper. Res.*, 20:257–301, 1995.

[71] D. Raz and D. Lorenz. Simple efficient approximation scheme for the restricted shortest path problem. Technical Report 10009674-991214-04TM, Bell Labs, 1999.

[72] T. Rockafellar. *Convex Analysis*. Princeton University Press, Princeton, NJ, 1970.

[73] A. Schrijver. A short proof of Guenin's characterization of weakly bipartite graphs. *J. Combin. Theory Ser. B*, 85:255–260, 2002.

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

[75] N. Shor. Convergence rate of the gradient descent method with dilatation of the space (in Russian). In *Kibernetica*, 2, pages 80–85, 1970. (English translation: Cybernetics, 6, pages 102–108, 1970).

[76] N. Shor. Utilization of the operation of space dilatation in the minimization of convex functions (in Russian). In *Kibernetica*, 1, pages 6–12, 1970. (English translation: Cybernetics, 6, pages 7–15, 1970).

[77] N. Shor. Cut-off method with space extension in convex programming problems (in Russian). In *Kibernetica*, 1, pages 94–95, 1977. (English translation: Cybernetics, 13, pages 94–96, 1977).

[78] R. Tarjan. Finding optimum branchings. *Networks*, 7:25–35, 1977.

[79] P. Vaidya. Speeding up linear programming using fast matrix multiplication. In *Proceedings, IEEE Symposium on Foundations of Computer Science*, pages 332–337, 1989.

[80] V. Vovk. Aggregating strategies. In *Proceedings of the 8th Annual Conference on Computational Learning Theory*, pages 371–383, 1990.

[81] V. Vovk. A game of prediction with expert advice. In *Proceedings of the 8th Annual Conference on Computational Learning Theory*, pages 51–60, 1995.

[82] N. Young. Greedy algorithms by derandomizing unknown distributions. Technical Report 1087, Department of Operations Research and Industrial Engineering, Cornell University, 1994.

[83] N. Young. Randomized rounding without solving the linear program. In *Proceedings, ACM-SIAM Symposium on Discrete Algorithms*, pages 170–178, 1995.

[84] N. Young. Sequential and parallel algorithms for mixed packing and covering. In *Proceedings, IEEE Symposium on Foundations of Computer Science*, pages 538–546, 2001.

[85] N. Young. Lagrangian relaxation via randomized rounding. Manuscript, 2004.

[86] D. Yudin and A. Nemirovskii. Evaluation of the informational complexity of mathematical problems (in Russian). In *Èkonomika i Matematicheskie Metody*, 12, pages 128–142, 1976. (English translation: Matekon, 13(2), pages 3–25, 1976-7).

[87] D. Yudin and A. Nemirovskii. Informational complexity and efficient methods for the solution of convex extremal problems (in Russian). In *Èkonomika i Matematicheskie Metody*, 12, pages 357–369, 1976. (English translation: Matekon, 13(3), pages 25–45, 1977).

Bio-data

Born in 1978, Rohit Khandekar had his initial schooling in Mumbai. Thereafter he moved to New Delhi for graduation. He received the degree of Bachelor of Technology from the Department of Computer Science & Engineering, Indian Institute of Technology (IIT) Delhi in 2000. He then joined the doctoral program at IIT Delhi. He visited Carnegie Mellon University, Pittsburgh, USA and Max-Planck-Institut, Saarbrücken, Germany for research in the summer months of 2000, 2001, and 2002. His broad area of research includes mathematical programming, combinatorial optimization, and approximation algorithms.