Designing Competitive Online Algorithms via a Primal-Dual Approach

Niv Buchbinder

Designing Competitive Online Algorithms via a Primal-Dual Approach

Research Thesis

Submitted in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy

Niv Buchbinder

Submitted to the Senate of the
Technion - Israel Institute of Technology

Elul 5768 | Haifa | September 2008

To my parents, Pnina and Shalom Buchbinder.

The research thesis was done under the supervision of Prof. Seffi Naor in the deprtment of Computer Science.

I would like to thank Seffi, my advisor, who has assisted me and guided me throughout the long years of my work. For pointing out many interesting new research directions. For many long long-distance calls through the night (in both sides). And most of all for your great support and encouragement. I highly appreciate the good fortune of having the chance to work with you.

I would like to thank the Microsoft research lab at Redmond and the IBM research lab at New York for their generous support during my visits, and also to the researchers from these labs for their collaborations on interesting research topics.

I would like to thank my many friends at the Technion, and outside, which have been a constant source of support and help. For many interesting coffee breaks at the 3rd floor that enriched my world.

I would like to thank my mother and father who where always there for me, assisting me.

Finally, I would like to thank my wife, Sharon, for her support and love all these years, through the pressure, the travels, and the deadlines. I couldn't have made it without you!

The generous financial help of the Technion is gratefully acknowledged.

Contents

List of Figures

List of Tables

Abstract

The primal-dual method is a powerful algorithmic technique that has proved to be extremely useful for a wide variety of problems in the area of approximation algorithms. The method has its origins in the realm of exact algorithms, e.g., for matching and network flow. In the area of approximation algorithms the primal-dual method has emerged as an important unifying design methodology starting from the seminal work of Goemans and Williamson [59].

We show in this thesis how to extend the primal-dual method to the setting of online algorithms, and show its applicability to a wide variety of interesting problems. Among the online problems that we consider here are the weighted caching problem, generalized caching, the set-cover problem, several graph optimization problems, routing, load balancing and the problem of allocating ad-auctions. We also show that classic online problems such as the ski rental problem and the dynamic TCP-acknowledgement problem can be optimally solved using a simple primal-dual approach.

The primal-dual method has several advantages over existing methods. First, it gives a general recipe for the design and analysis of online algorithms. The analysis of the competitive ratio is direct, without a potential function appearing "out of nowhere". Finally, since the analysis is done via duality, the competitiveness of the online algorithm is with respect to an optimal fractional solution which can be advantageous in certain scenarios.

Abbreviations and Notations

LP	Linear program.
P	A primal (minimum) linear program.
D	A dual (maximum) linear program.
ΔP, ΔD	The change in the cost of the primal and dual programs, respectively.
$G = (V, E)$	Graph with set of vertices V and set of edges E.
$\sigma = \sigma_1, \sigma_2, \ldots$	A request sequence.
$\mathrm{O}PT(\sigma)$	The cost of the optimal solution on the request sequence σ.

Chapter 1

Preface

The primal-dual method is a powerful algorithmic technique that has proved to be extremely useful for a wide variety of problems in the area of approximation algorithms. The method has its origins in the realm of exact algorithms, e.g., for matching and network flow. In the area of approximation algorithms the primal-dual method has emerged as an important unifying design methodology starting from the seminal work of Goemans and Williamson [59].

We show here how to extend the primal-dual method to the setting of online algorithms, and show that it is applicable to a wide variety of problems. The method we propose has several advantages over existing methods:

- A general recipe for the design and analysis of online algorithms is developed.
- The framework is shown to be applicable to a wide range of interesting online problems.
- The use of a linear program helps detecting the difficulties of the online problem in hand.
- The competitive ratio analysis is direct, without a potential function appearing "out of nowhere".
- The competitiveness of the online algorithm is with respect to an optimal fractional solution.

In Chapter 2 we briefly discuss necessary background needed for the rest of the discussion. This includes a short exposition on linear programming, duality, offline approximation methods, and basic definitions of online computation. Many readers may already be familiar with these basic definitions and techniques, however, we advise the readers not to skip this chapter, and in particular the part on approximation algorithms. Approximation algorithms techniques are presented in a way that allows the reader to later see the similarity to the online techniques we develop. This chapter also provides some of the basic notation that we use in the rest of our discussion. In Chapter 3 we give a first taste of how the primal-dual approach is helpful in the context of online algorithms. This is done via the well understood ski rental problem. We show how it is possible to derive alternative optimal algorithms for the ski rental problem using a simple primal-dual approach. In Chapter 4 we place the foundations of the online primal-dual approach. We describe the general online setting and design the basic algorithms for that setting. We also study two toy examples that demonstrate the online framework. The rest of the chapters show how to apply the primal-dual approach to many interesting problems. We tried to make the chapters independent and so the reader may skip some of the results. However, there are still some connections between the chapters, and so closely related problems appear in consecutive chapters, and in increasing order of complexity.

Among the problems that we consider are the weighted caching problem, generalized caching, the online set-cover problem, several graph optimization problems, routing, load balancing, and even the problem of allocating ad-auctions. We also show that classic online problems like the dynamic TCP-acknowledgement problem can be optimally solved using a primal-dual approach. There are also several more problems that can be solved via the primal-dual approach and are not discussed here.

Chapter 2

Necessary Background

In this chapter we briefly discuss necessary background needed for the rest of the thesis. In Section 2.1 we briefly discuss the notion of linear programming and duality. In Section 2.2 we define the notion of optimization problems and discuss several classic methods for deriving (offline) approximation algorithm. We demonstrate these ideas on the set cover problem which is later considered in the online setting. In Section 2.3 we give the basic concepts and definitions related to online computation. This chapter is not meant to give a comprehensive introduction, but rather only provide the basic notation and definitions used later on in the text. For a more comprehensive discussion of these subjects we refer the reader to the many excellent textbooks on these subjects. For more information on linear programming and duality we refer the reader to [41]. For further information on approximation techniques we refer the reader to [88]. Finally, for more details on online computation and competitive analysis we refer the reader to [28].

2.1 Introduction to Linear programming and Duality

Linear programming is the problem of minimizing or maximizing a linear objective function over a feasible set defined by a set of linear inequalities. There are several equivalent forms of formulating a linear program. In our discussion the most convenient format is the following:

$$(P) : \min \quad \sum_{i=1}^{n} c_i x_i$$

Subject to:
For any $1 \leq j \leq m$: $\quad \sum_{i=1}^{n} a_{ij} x_i \geq b_j$

$$\forall 1 \leq i \leq n \qquad x_i \geq 0$$

It is well known that any linear program can be formulated this way. We refer to such a minimization problem as the primal problem (P). Every primal linear program has a corresponding dual program with certain properties that we discuss in the sequel.

The dual linear program is a maximization linear program. It has m dual variables that correspond to the primal constraints. It has n packing constraints that correspond to the primal variables. The dual program (D) that corresponds to the linear program formulation (P) is the following.

$$(D) : \max \quad \sum_{j=1}^{m} b_j y_j$$

Subject to:
For any $1 \leq i \leq n$: $\quad \sum_{j=1}^{m} a_{ij} y_j \leq c_i$

$$\forall 1 \leq j \leq m \qquad y_j \geq 0$$

The dual linear formulation is very useful. The main properties of the dual program that make it so useful are summarized in the following Theorems:

Theorem 2.1 (Weak duality)**.** *Let $x = (x_1, x_2, \ldots, x_n)$ and $y = (y_1, y_2, \ldots, y_m)$ be feasible solutions to the primal and the dual linear programs respectively, then:*

$$\sum_{i=1}^{n} c_i x_i \geq \sum_{j=1}^{m} b_j y_j$$

The weak duality theorem says that the value of any feasible dual solution is at most the value of any feasible primal solution. Thus, the dual program can actually be used as a lower bound for any feasible primal solution. The proof of this theorem is quite simple.

Proof.

$$\begin{aligned} \sum_{i=1}^{n} c_i x_i &\geq \sum_{i=1}^{n} \left(\sum_{j=1}^{m} a_{ij} y_j \right) \cdot x_i && (2.1) \\ &= \sum_{j=1}^{m} \left(\sum_{i=1}^{n} a_{ij} x_i \right) \cdot y_j && (2.2) \\ &\geq \sum_{j=1}^{m} b_j y_j, && (2.3) \end{aligned}$$

where Inequality (2.1) follows since $y = (y_1, y_2, \ldots, y_m)$ is feasible and each x_i is non-negative. Equality (2.2) follows by changing the order of summation. Inequality (2.3) follows since $x = (x_1, x_2, \ldots, x_n)$ is feasible and each y_j is non-negative. □

The next theorem is sometimes referred to as the strong duality Theorem. It states that if the primal and dual programs are bounded, then the optima of the two programs is equal. The proof of the strong duality Theorem is harder and we only state the theorem here without a proof.

Theorem 2.2 (Strong duality)**.** *The primal linear program has a finite optimal solution if and only if the dual linear program has a finite optimal solution. In this case the value of the optimal solutions of the primal and dual programs is equal.*

Finally, we prove an important theorem that is used extensively in the context of approximation algorithms. The theorem states that if two conditions hold with respect to the primal and the dual then the solution is optimal (or approximately optimal).

Theorem 2.3 (Complementary slackness)**.** *Let* $x = (x_1, x_2, \ldots, x_n)$ *and* $y = (y_1, y_2, \ldots, y_m)$ *be feasible solutions to the primal and dual linear programs respectively, satisfying the following conditions:*

- *Primal complementary slackness condition: Let* $\alpha \geq 1$.
  For each $1 \leq i \leq n$, *if* $x_i > 0$ *then* $\frac{c_i}{\alpha} \leq \sum_{j=1}^{m} a_{ij} y_i \leq c_i$.
- *Dual complementary slackness condition: Let* $\beta \geq 1$.
  For each $1 \leq j \leq m$, *if* $y_j > 0$ *then* $b_j \leq \sum_{i=1}^{n} a_{ij} x_i \leq b_j \cdot \beta$.

Then:

$$\sum_{i=1}^{n} c_i x_i \leq \alpha \cdot \beta \sum_{j=1}^{m} b_j y_j$$

In particular if the complementary slackness conditions hold with $\alpha = \beta = 1$ then we get that $\vec{x}$ and $\vec{y}$ are both optimal solutions to the primal and dual linear programs respectively. The proof of the theorem is again very short:

Proof.

$$\begin{aligned}
\sum_{i=1}^{n} c_i x_i &\leq \alpha \sum_{i=1}^{n} \left(\sum_{j=1}^{m} a_{ij} y_i \right) x_i && (2.4)\\
&= \alpha \sum_{j=1}^{m} \left(\sum_{i=1}^{n} a_{ij} x_i \right) y_j && (2.5)\\
&\leq \alpha \cdot \beta \sum_{j=1}^{m} b_j y_j, && (2.6)
\end{aligned}$$

where (2.4) follows by the primal complementary slackness condition. Equality (2.5) follows by changing the order of summation, and Inequality (2.6) follows by the dual complementary slackness condition. □

Theorem 2.3 gives an efficient tool for finding approximate solutions. Consider, for example, a minimization problem. Suppose that you can find primal and dual solutions that satisfy the complementary slackness conditions. Then, you get that the solution for the minimization problem is at most $\alpha \cdot \beta$ times a feasible dual solution. Since, by weak duality (Theorem 2.1), the value of any dual solution is a lower bound on the value of any primal solution, the solution you found is also at most $\alpha \cdot \beta$ times the optimal (minimal) primal solution.

Covering/Packing linear formulations: A special subclass of linear programs consists of programs in which the coefficients a_{ij}, b_j and c_i are all non-negative. In this case the primal formulation is called *covering problem* and the dual formulation forms a *packing problem*. The meaning of the names will become clear in Section 2.2, where we discuss the set cover problem. In our following discussion we sometime use this notion of covering-packing primal-dual pair.

2.2 Introduction to Approximation Algorithms

In this section we give a very short background on some basic methods used in approximation algorithms. Later we show that the same ideas can be extended and used in the context of online algorithms. We start by formally defining the notions of optimization problems and approximation factors. In an (offline) minimization optimization problem we are given set of instances $\mathbb{I}$. For each instance $I \in \mathbb{I}$ there is a set of feasible solutions. Each feasible solution is associated with a *cost*. Let $\mathrm{O}PT(I)$ be the cost of the minimal feasible solution for instance I. A polynomial time algorithm A is called a c-approximation for a minimization optimization problem if for every instance I it outputs a solution with cost at most $c \cdot \mathrm{O}PT(I) + \alpha$, where α is some constant independent of the input. In all our discussion α is zero and so we leave it out. The definitions for maximization optimization problems are analogues. In this case, each instance is associated with a *profit*. A c-approximation algorithm is guaranteed to return a solution with cost at least $\mathrm{O}PT(I)/c$, where $\mathrm{O}PT(I)$ is the maximum profit solution.

The set cover problem: We demonstrate several classic ideas used for developing approximation algorithms via the set cover problem. In the set cover problem we are given a set of n elements $X = e_1, e_2, \ldots, e_n$, and a family $\mathbb{S} = s_1, s_2, \ldots, s_m$ of subsets of X, $|\mathbb{S}| = m$. Each set s_j is associated with a non negative cost c_s. A *cover* is a collection of sets such that their union is X. The objective is to find a cover of X of minimum cost, and this problem is known to be NP-hard. LP relaxations constitute a very useful way for obtaining lower bounds on the original solution of the combinatorial problem. To this end we introduce a non negative variable x_s for each sets $\in \mathbb{S}$. Next, consider the following linear formulation of the problem:

$$(P) : \min \sum_{s \in \mathbb{S}} c_s x_s \qquad \text{s.t.}$$

for each element e_i, $(1 \leq i \leq n)$: $\qquad \sum_{s | e_i \in S} x_s \geq 1$

$$\forall s \in \mathbb{S}, \qquad x_s \geq 0$$

Constraining x_s to be 0 or 1 instead of $x_s \geq 0$ would yield an equivalent integral formulation of the set cover problem (by setting $x_s = 1$ for each set in an optimal cover). Such formulations are called *integer programming* formulations. Since the feasible space of the linear formulation contains as a subspace all the integral solutions of the integer formulation, we get that the optimal solution to the linear formulation sets a lower bound on the value of any integral set cover solution. A solution to the linear formulation is called a fractional set cover solution. In such a relaxed set cover solution one is allowed to take a fraction x_s of each set and pay only $c_s x_s$ for this fraction. The restriction is that the sum of fractions of sets that contain each element e_i should be at least 1. In the corresponding dual linear program of (P) there is a variable for each element e_i. The dual program (D) is the following.

$$(D) : \max \sum_{e \in X} y_{e_i} \qquad \text{s.t.}$$

for each set $s \in \mathbb{S}$: $\qquad \sum_{e_i \in S} y_{e_i} \leq c_s$

$$\forall 1 \leq i \leq n, \qquad y_{e_i} \geq 0$$

We next demonstrate several classic approximation techniques using the set cover problem as our running example.

2.2.1 Dual Fitting Method

We present dual fitting by analyzing a simple greedy algorithm for the set cover problem and show that the algorithm is an $O(\log n)$-approximation. The greedy algorithm is the following: Let $\mathbb{C}$ be the sets in the cover produced by the algorithm. Initially, $\mathbb{C} = \emptyset$. Let U be the set of yet uncovered elements. Initially $U = X$. As long as $U \neq \emptyset$, add to $\mathbb{C}$ the set $s \in \mathbb{S}$ that minimizes the ratio $\frac{c_s}{|U \cap s|}$.

To prove that this simple algorithm is an $O(\log n)$-approximation, we first describe the same algorithm a bit differently as a primal-dual algorithm. This kind of description is used in the sequel.

Greedy algorithm: Initially, $\mathbb{C} = \emptyset$. Let U be the set of yet uncovered elements. As long as $U \neq \emptyset$, let $s \in \mathbb{S}$ be the set that minimizes the ratio $\frac{c_s}{|U \cap s|}$:

1. Add s to $\mathbb{C}$ and set $x_s \leftarrow 1$.
2. For each $e_i \in (U \cap s)$, $y_{e_i} \leftarrow \frac{c_s}{|U \cap s|}$.

Theorem 2.4. *The greedy algorithm is an $O(\log n)$-approximation algorithm for the set cover problem.*

Proof. Note that the algorithm produces throughout its execution both primal and dual solutions. Let P and D be the values of the objective function of the primal and dual solutions the algorithm produces, respectively. Initially, $P = D = 0$. We focus on a single iteration of the algorithm and denote by ΔP and ΔD the change in the primal and dual cost, respectively. We prove three simple claims:

1. The algorithm produces a primal (covering) feasible solution.

2. In each iteration: $\Delta P \leq \Delta D$.

3. Each packing constraint in the dual program is violated by a multiplicative factor of at most $O(\log n)$.

The proof follows immediately from the three claims together with weak duality. First by claim (1) our solution is feasible. By the fact that initially $P = D = 0$ and by claim (2) we get that we produce a primal and dual solutions such that $P \leq D$. Finally, by claim (3) we get that dividing the dual solution by $c \log n$, for a constant c, we obtain a dual feasible solution with value $D' = \frac{D}{c \log n}$. Therefore, we get that the primal cost is at most $c \log n$ times a feasible dual solution. Since by weak duality a feasible dual solution is at most the cost of any primal solution, we get that the primal solution is at most $c \log n$ times the optimal (minimal) primal solution.
Proof of (1)**:** It is easy to verify that if there exists a feasible primal solution, then the algorithm will also produce a feasible solution.
Proof of (2)**:** In each iteration of the algorithm the change in the primal cost (due to the addition of set s) is c_s. In the dual we set $|U \cap s|$ dual variables each to $\frac{c_s}{|U \cap s|}$, and so the total change in the dual profit is also c_s. Note that we only set the dual variable of each element once in the round in which it was covered.
Proof of (3)**:** Consider the dual constraint corresponding to set s. Let $e_1, e_2, \ldots, e_k$ be the elements belonging to the set s ordered in the same order as covered by the greedy algorithm. Consider element e_i; we claim that $y_{e_i} \leq \frac{c_s}{k-i+1}$. This is true since at the time e_i was covered by the greedy algorithm, the set s contained at least $k - i + 1$ elements that were still uncovered. Therefore, the algorithm could pick the set s and set y_{e_i} to $\frac{c_s}{k-i+1}$. Since the algorithm chose the set that minimizes this ratio and so minimizes the value of y_{e_i} then y_{e_i} is at most $\frac{c_s}{k-i+1}$. Therefore, for the set s we get that:

$$\sum_{e_i \in s} y_{e_i} \leq \sum_{i=1}^{k} \frac{c_s}{k - i + 1} \leq H_k \cdot c_s = c_s \cdot O(\log n),$$

where H_k is the kth harmonic number. $\square$

2.2.2 Rounding Linear Programming Solutions

In this section we design a different $O(\log n)$-approximation algorithm for the set cover problem using a technique called randomized rounding. We first compute an optimal fractional solution to linear program (P). Recall that in a feasible solution, for each element, the sum of fractions of the sets containing it is at least 1. We describe the rounding algorithm a bit differently than in standard textbooks. This description will be useful in the sequel. The rounding algorithm is the following:

Randomized Rounding algorithm:

1. For each set $s \in \mathbb{S}$, choose $2 \ln n$ independently random variables $X(s,i)$ uniformly at random in the interval $[0,1]$.
2. For each set s, let $\Theta(s) = \min_{i=1}^{2\ln n} X(s,i)$.
3. Solve the program (P).
4. Take s to the cover if $\Theta(s) \leq x_s$, where x_s is the value of the variable in the solution to the LP.

Theorem 2.5. *The algorithm produces a solution with the following properties:*

- *The expected cost of the solution is $O(\log n)$ times the cost of the fractional solution.*
- *The solution is feasible with probability $1 - 1/n > 1/2$.*

It is not hard to prove using standard arguments that with constant probability the algorithm produces a feasible solution whose cost is at most $O(\log n)$ times the fractional solution. However, for our discussion the above properties are enough. Since the fractional solution provides a lower bound on any integral solution, we get that the algorithm is an $O(\log n)$-approximation.

Proof. To prove (1) note that for each i, $1 \leq i \leq 2\ln n$, the probability that $X(s,i) \leq x_s$ is exactly x_s. The probability that s is chosen to the solution is the probability that there exists an i, $1 \leq i \leq 2\ln n$, such that $X(s,i) \leq x_s$. Let A_i be the event that $X(s,i) \leq x_s$. Using this definition the probability that s is chosen to the solution is the probability of $\bigcup_{i=1}^{2\ln n} A_i$. By the union bound this probability is at most the sum of the probabilities of the events which is $2x_s \ln n$. Therefore, using linearity of expectation the expected cost of the solution is at most $2\ln n$ times the fractional solution.

To prove (2) pick an element e. Fix any i, $1 \leq i \leq 2\ln n$. The probability that e is not covered due to the set $X(s,i)$ is the probability that we are not choosing any set s covering the element e. This probability is:

$$\prod_{s \in \mathbb{S} | e \in s} (1 - x_s) \leq \exp\left(- \sum_{s \in \mathbb{S} | e \in s} x_s \right) \leq \exp(-1),$$

where the first inequality follows since $1 - x \leq \exp(-x)$. The second inequality follows since each element is fractionally covered. Since we choose $2\ln n$ random variables independently, the probability that e is not covered is at most $\exp(-2\ln n) = \frac{1}{n^2}$. Using the union bound we get that the probability that there exists an element e which is not covered is at most $n \cdot \frac{1}{n^2} = \frac{1}{n}$. □

2.2.3 The Primal-dual Schema

In this section we give a third approximation algorithm for the set cover problem using the complementary slackness Theorem. The algorithm is the following:

Primal-dual algorithm:
While there exists an uncovered element e_i:

1. Increase the dual variable y_{e_i} continuously.
2. If there exists a set s such that $\sum_{e \in s} y_e = c_s$: Take s to the cover and set $x_s \leftarrow 1$.

Let f be the maximum frequency of an element (i.e., the maximum number of sets an element belong to).

Theorem 2.6. *The algorithm is an f-approximation for the set cover problem.*

Proof. Clearly, the primal solution produced by the algorithm is feasible, since we pick sets to the cover as long as the solution is infeasible. The dual solution produced by the algorithm is also feasible since whenever a dual constraint of a set becoming tight, we take s to the solution, and then we never increase a dual variable corresponding to an element belonging to s. Finally, the primal complementary slackness condition holds with $\alpha = 1$, since, if $x_s > 0$, then $\sum_{e \in s} y_{e_i} = c_s$. The dual complementary slackness condition holds with $\beta = f$, since, if $y_{e_i} > 0$, then $1 \leq \sum_{s|e \in s} x_s \leq f$. Thus, by Theorem 2.3 we get that the algorithm ia an f-approximation. □

2.3 Introduction to Online Computation

The performance of online algorithms is defined very similarly to the performance of offline approximation algorithms. Suppose we are given a minimization optimization problem. For each instance of the problem I there is a set of feasible solutions. Each feasible solution is associated with a *cost*. Let $\mathrm{O}PT(I)$ be the cost of the minimal feasible solution for instance I. In the online case the instance is given to the algorithm in parts. These parts are usually referred to as requests. Each specific online problem also defines certain restrictions on the way the online algorithm is allowed to process these requests. An online algorithm A is said to be c-competitive for a minimization optimization problem if for every instance I it outputs a solution of cost at most $c\mathrm{O}PT(I) + \alpha$, where α is independent of the request sequence. if $\alpha = 0$ then the algorithm is called **strictly** c-competitive. We do not distinguish between the two notions. Analysis of online algorithm with respect to this measure is referred to as *competitive analysis*. The definition of competitiveness maximization optimization problems is analogues. When considering a maximization problem each instance is associated with a *profit*. A c-competitive algorithm is guaranteed to return a solution with cost at least $\mathrm{O}PT(I)/c - \alpha$, where $\mathrm{O}PT(I)$ is the maximum profit solution, and α is independent of the request sequence.

A common concept in competitive analysis that of an *adversary*. The online solution is being viewed as a game between an online algorithm and a malicious adversary. While the online algorithm would like to minimize its cost, the adversary would like to construct the worst possible input for the algorithm. Using this view the adversary produces a sequence $\sigma = \sigma_1, \sigma_2, \ldots$ of requests that define the instance I. A c-competitive online algorithm should then be able to produce a solution of cost no more than c times $\mathrm{O}PT(\sigma)$ for every request sequence σ.

It is also possible to define several natural models of randomized online algorithms. In this work we only consider the model where the adversary knows the algorithm and the probability distribution the algorithm uses to make its random decisions. The adversary is not aware, however, of the actual random choices made by the algorithm throughout its execution. This kind of adversary is called *oblivious adversary*. A randomized online algorithm is c-competitive against an oblivious adversary, if for every request sequence σ, the expected cost of the algorithm on σ is at most $c \cdot \mathrm{O}PT(\sigma) + \alpha$. The expectation is taken over all random choices made by the algorithm. Since the oblivious adversary has no information about the actual random choices made by the algorithm, the sequence σ can be constructed ahead of this and $\mathrm{O}PT(\sigma)$ is not a random variable. In the sequel, whenever we have a random online algorithm, we simply say that it is c-competitive, and mean that it is c-competitive against an oblivious adversary. Again, the definitions for maximization problems are analogues.

2.4 Notes

The dual-fitting analysis in Section 2.2.1 of the greedy heuristic is due to [76, 40]. The algorithm in Section 2.2.3 is due to Bar-Yehuda and Even [17]. The set cover problem is a classic NP-hard problem that was studied extensively in the literature. The best approximation factor achievable for it in polynomial time (assuming $P \neq NP$) is $\Theta(\log n)$ [40, 46, 67, 76].

The introduction here is only meant to provide basic notation and definition for the rest of our discussion. The area of linear programming, duality, approximation algorithms and online computation have been studied extensively in many directions. For more information on linear programming and duality we refer the reader to [41]. For further information on approximation techniques we refer the reader to [88]. Finally, for more details on online computation other online models and competitive analysis we refer the reader to [28].

Chapter 3

A First Glimpse: The Ski Rental Problem

Let us start with a Let us start with an example which will provide us with the flavor of a primal-dual approach in the context of online computations. We demonstrate the method on a classic, yet very simple problem, called the *ski rental* problem. A customer arrives at a ski resort, where renting skis costs \$1 per day, while buying them costs \$$B$. The unknown factor is the number of skiing days left before the snow melts. This is the customer's last vacation, so the goal is to minimize the total expenses. In spite of its apparently simple description, the ski rental problem captures the essence of online rent/buy dilemmas. The ski rental problem is well understood. There exists a simple deterministic 2-competitive algorithm for the problem and a randomized $e/(e-1)$-competitive algorithm. Both results are tight. We show here how to obtain these results using a primal-dual approach.

The first step towards a primal-dual algorithm is formulating the problem in hand as a linear program. Since the offline ski-renal problem is so simple casting it as a linear program may seem a bit unnatural. However, this formulation turn out to be very useful. We define a variable x which is set to 1 if we decide to buy the skis. For each day j, $1 \leq j \leq k$, we define a variable z_j which is set to 1 if we decide to rent the skis on that day. The constraints guarantee that on each day we either rent skis or buy them. We now relax the problem and allow x and each z_j to be in $[0, 1]$. The linear program is depicted in Figure 3.1 as the primal program. Note that the optimal solution is always integral, and thus the relaxation has no integrality gap. The dual program is also extremely simple and consists of variables y_j corresponding to each day j. Note that the linear programming formulation forms a covering-packing primal-dual pair.

Next, consider the online scenario in which k (the number of ski days) is unknown in advance. This online scenario is captured in the linear formulation in a very natural way. Whenever we have a new ski day, the primal linear program is updated by adding a new covering constraint. The dual program is updated by adding a new dual variable which is added to the packing constraints. The online requirement is that previous decisions cannot be regretted. That is, if we already rented skies yesterday, we cannot change

Dual (Packing)		Primal (Covering)	
Maximize:	$\sum_{j=1}^{k} y_j$	Minimize :	$B \cdot x + \sum_{j=1}^{k} z_j$
Subject to:		Subject to:	
	$\sum_{j=1}^{k} y_j \leq B$	For each day j:	$x + z_j \geq 1$
For each day j:	$0 \leq y_j \leq 1$		$x \geq 0, \forall j, z_j \geq 0$

Figure 3.1: The fractional ski problem (the primal) and the corresponding dual problem

this decision today. This requirement is captured in the primal linear program by the restriction that the primal variables cannot be decreased.

Obtaining an optimal deterministic online algorithm for the ski-rental problem is very simple. Rent skies the first B days, and then buy skies on the Bth day. If $k < B$, the algorithm is optimal. If $k \geq B$, then the algorithm spends \$$2B$ dollars while the optimal solution buys skies on the first day and spends only \$$B$ dollars. Thus, the algorithm is 2-competitive. This simple algorithm and analysis can be obtained in a bit less natural way using a primal-dual analysis. The description of the algorithm is the following. On the jth day, a new primal constraint $x + z_j \geq 1$ and a new dual variable y_j arrive. If the primal constraint is already satisfied then do nothing. Otherwise, increase y_j continuously until some dual constraint becomes tight. Set the corresponding primal variable to be 1. The above algorithm is a simple usage of the primal-dual schema and its analysis is very simple using complementary slackness conditions: If $y_j > 0$ then $1 \leq x + z_j \leq 2$. Moreover, if $x > 0$ then $\sum_{j=1}^{k} y_j = B$, and if $z_j > 0$ then $y_j = 1$. Thus, by the complementary slackness theorem (Theorem 2.3) the algorithm is 2-competitive. It is not hard to see that both algorithms are actually identical.

Obtaining an optimal randomized algorithm is a bit harder, but can be done very easily using a primal-dual approach. The first step is the design of a fractional competitive algorithm. Recall that in the fractional case the primal variables can be in the interval $[0, 1]$, and we require that they cannot be decreased during the execution of the algorithm. The online algorithm is the following:

1. Initially, $x \leftarrow 0$.
2. Each new day (jth new constraint), if $x < 1$:
   (a) $z_j \leftarrow 1 - x$.
   (b) $x \leftarrow x\left(1 + \frac{1}{B}\right) + \frac{1}{c \cdot B}$. (The value of c is determined later.)
   (c) $y_j \leftarrow 1$.

The analysis is simple. We show that: (i) the primal and dual solutions are feasible; (ii) in each iteration, the ratio between the change in the primal and dual objective functions is bounded by $(1+1/c)$. Using weak duality theorem (Theorem 2.1) we immediately conclude that the algorithm is $(1 + 1/c)$-competitive.

The proof is very simple. First, since we set $z_j = 1 - x$ whenever $x < 1$, the primal solution produced is feasible. Second, if $x < 1$, the dual objective function increases by 1, and the increase in the primal objective function is $B\Delta x + z_j = x + 1/c + 1 - x = 1 + 1/c$, thus the ratio is $(1 + 1/c)$. Third, to show feasibility of the dual solution, we need to show $\sum_{j=1}^{k} y_j \leq B$. We prove $x \geq 1$ after at most B days of ski. It is easy to verify that the variable x is the sum of a geometric sequence in which $a_1 = 1/(cB)$ and $q = 1 + 1/B$. Thus, after B days, $x = \frac{\left(1+\frac{1}{B}\right)^B - 1}{c}$. Choosing $c = (1 + \frac{1}{B})^B - 1$ guarantees that $x = 1$ after B days. Thus, the competitive ratio is $1 + 1/c \approx e/(e - 1)$ when $B \gg 1$.

Transforming the fractional solution into a randomized competitive algorithm with the same competitive ratio is easy. We arrange the increments of x on the interval $[0, 1]$ and choose uniformly in random $\alpha \in [0, 1]$ before executing the algorithm. We buy skies on the day corresponding to the increment of x to which α belongs. It can be seen that if there were k days of ski then the probability of buying skis is exactly the value of x on the kth day. Also, the probability of renting the skis on the jth day (if we did not buy before the jth day) is $1 - x_j$, where x_j is the value of x on the jth day. Since $z_j = 1 - x_{j-1} \geq 1 - x_j$ we get that the probability of renting on the jth day is at most z_j. Thus, by linearity of expectation, for any number of ski days, the expected cost of the randomized algorithm is at most the cost of the fractional solution.

Chapter 4

The Basic Approach

In this chapter we generalize the ideas used in the ski rental problem and devise a general recipe for the design and analysis of online algorithms for problems which can be formulated as packing-covering. In Section 4.1 we formally define a general online framework. In Section 4.2 we give several alternative algorithms for this online framework. In Section 4.3 we prove lower bounds that show these algorithms are optimal for the problem. Finally, in Section 4.4 we give two simple examples that make use of our ideas to devise algorithms for certain problems.

4.1 The Online Setting

We formally define our online framework. Think first on an "offline" covering problem. In such a problem the objective is to minimize the total cost given by a linear cost function $\sum_{i=1}^{n} c_i x_i$. The feasible solution space is defined by a set of m linear constraints of the form $\sum_{i=1}^{n} a(i,j)x_i \geq b(j)$, where the entries $a(i,j), b(j)$ and c_i are non-negative. For simplicity we consider in this chapter a simpler setting in which $b(j) = 1$ and $a(i,j) \in \{0,1\}$. In Chapter 14 we show how to extend the ideas we present here to handle general (non-negative) values of $a(i,j)$ and $b(j)$. In the simpler setting each covering constraint j is associated with a set $S(j)$ such that $i \in S(j)$ if $a(i,j) = 1$. The jth covering constraint then reduces to simply $\sum_{i \in S(j)} x_i \geq 1$. Any primal covering instance has a corresponding dual packing problem that provides a lower bound on any feasible solution to the instance. A general form of a (simpler) primal covering problem along with its dual packing problem is given in Figure 4.1.

The *online covering problem* is an online version of the covering problem. In this setting the cost function is known in advance, but the linear constraints that define the feasible solution space are given to the algorithm one-by-one. In order to maintain a feasible solution to the current set of given constraints, the algorithm is allowed to increase the variables x_i. It may not, however, decrease any previously increased variable. The objective of the algorithm is to minimize the objective function. The reader may already see that this online setting captures the setting of the ski rental problem (Chapter 3) as a special case.

(P): Primal (Covering)		(D): Dual (Packing)	
Minimize:	$\sum_{i=1}^{n} c_i x_i$	Maximize:	$\sum_{j=1}^{m} y_j$
Subject to:		Subject to:	
For each $1 \leq j \leq m$:	$\sum_{i \in S(j)} x_i \geq 1$	For each $1 \leq i \leq n$:	$\sum_{j \mid i \in S(j)} y_j \leq c_i$
For each $1 \leq i \leq n$:	$x_i \geq 0$	For each $1 \leq j \leq m$:	$y_j \geq 0$

Figure 4.1: Primal (covering) and dual (packing) problems

We also define an online version of the packing problem. In the *online packing problem* the values c_i ($1 \leq i \leq n$) are known in advance. However, the profit function and the exact packing constraints are not known in advance. In the jth round a new variable y_j is introduced to the algorithm, along with the set of packing constraints it appears in. The algorithm may increase the value of a variable y_j only in the round when it is given, and may not decrease or increase the values of any previously given variables. Note that other variables that have not yet been introduced may also later appear in the same packing constraints. This actually means that each packing constraint is revealed to the algorithm gradually. The objective of the algorithm is to maximize the objective function while maintaining all packing constraints feasible. Although this online setting seems at first glance a bit unnatural, we show later that many natural online problems reduce to this online setting.

We first observe that these two online settings form a primal-dual pair in the following sense: At any point of time an algorithm for the online fractional covering problem maintains a subset of the final linear constraints. This subset defines a *sub-instance* of a final covering instance. The dual packing problem of this sub-instance is a *sub-instance* of the final dual packing problem. In the dual packing sub-instance, only part of the dual variables are known, along with their corresponding coefficients. The two sub-instances form a *primal-dual pair*. In each round of the online fractional covering problem a new covering constraint is given. The primal covering sub-instance is updated by adding this new constraint. To update the dual sub-instance we add a new dual variable to the profit function along with its coefficients that are defined by the new primal constraint. Note that the dual update is the same as in the setting of the online packing problem.

The algorithms we propose in the following maintain at each step solutions for both the primal and dual sub-instances. When a new constraint is given to the online fractional covering problem, our algorithms also considers the new *corresponding* dual variable and its coefficients. When the algorithms for the online fractional packing problem receives a new variable along with its coefficients, it also considers the *corresponding* new constraint in the primal sub-instance.

4.2 Three Simple Algorithms

In this section we present three algorithms with the same (optimal) performance guarantees for the online covering/packing problem. Although the performance of all three algorithms in the worst case is the same, some are better fit for certain applications. Also, the ideas of each algorithm can be extended later to more complex settings. The first algorithm is referred to as the *basic discrete algorithm* and is a direct extension of the algorithm for the ski-rental in Chapter 3. The algorithm is the following:

> **Algorithm 1:**
> Whenever a new primal constraint $\sum_{i \in S(j)} x_i \geq 1$ and the corresponding dual variable y_j appear:
>
> 1. While $\sum_{i \in S(j)} x_i < 1$:
>   (a) For each $i \in S(j)$: $x_i \leftarrow x_i \left(1 + \frac{1}{c_i}\right) + \frac{1}{|S(j)|c_i}$.
>   (b) $y_j \leftarrow y_j + 1$

We analyze the algorithm assuming that each $c_i \geq 1$. This assumption is not restrictive and we discuss this later on. Let $d = \max_j |S(j)| \leq m$ be the maximal "size" of a covering constraint. We prove the following theorem:

Theorem 4.1. *The algorithm produces:*

- *A fractional covering solution which is $O(\log d)$-competitive.*
- *An* **integral** *packing solution which is 2-competitive and violates each packing constraint by at most factor of $O(\log d)$.*

We remark that it is not hard to make the packing solution feasible by dividing the update of each y_j by $O(\log d)$. This, however, will yield a packing solution which is not integral. It is also beneficial for the reader to note the similarity (in spirit) to the proof of Theorem 4.1 and the dual fitting based proof of the greedy heuristic for the set cover problem in Section 2.2.1.

Proof. Let P and D be the values of the objective function of the primal and the dual solution the algorithm produce respectively. Initially, $P = D = 0$. Let ΔP and ΔD be the changes in the primal and dual cost, respectively, in a particular iteration of the algorithm in which we enter the inner loop. We prove three simple claims:

1. The algorithm produces a primal (covering) feasible solution.
2. In each iteration: $\Delta P \leq 2\Delta D$.
3. Each packing constraint in the dual program is violated by at most $O(\log d)$.

The Theorem then follows immediately by the three claims along with weak duality.
Proof of (1)**:** Consider a primal constraint $\sum_{i \in S(j)} x_i \geq 1$. During the jth iteration the algorithm increases the values of the variables x_i until the constraint is satisfied. Subsequent increases of the variables cannot make the solution infeasible.
Proof of (2)**:** Whenever the algorithm updates the primal and dual solutions, the change in the dual profit is 1. The change in the primal cost is:

$$\sum_{i \in S(j)} c_i \Delta x_i = \sum_{i \in S(j)} c_i \left(\frac{x_i}{c_i} + \frac{1}{|S(j)| c_i} \right) = \sum_{i \in S(j)} \left(x_i + \frac{1}{|S(j)|} \right) \leq 2,$$

where the final inequality follows since the covering constraint is infeasible at the time of the update.

Proof of (3): Consider any dual constraint $\sum_{j|i \in S(j)} y_j \leq c_i$. Whenever we increase some y_j such that $i \in S(j)$ by one unit we also increase the variable x_i in line 1a. We prove by simple induction that the variable x_i is bounded from below by the sum of a geometric sequence with $a_1 = \frac{1}{dc_i}$ and $q = (1 + \frac{1}{c_i})$. That is,

$$x_i \geq \frac{1}{d} \left(\left(1 + \frac{1}{c_i} \right)^{\sum_{j|i \in S(j)} y_j} - 1 \right). \tag{4.1}$$

Initially $x_i = 0$ so the induction hypothesis holds. Next, consider an iteration in which some y_k increases by 1. Let $x_i(\text{start})$ and $x_i(\text{end})$ be the value of x_i before and after the increment respectively. Then,

$$\begin{aligned} x_i(\text{end}) &= x_i(\text{start}) \cdot \left(1 + \frac{1}{c_i} \right) + \frac{1}{|S(j)| \cdot c_i} \geq x_i(\text{start}) \cdot \left(1 + \frac{1}{c_i} \right) + \frac{1}{d \cdot c_i} \\ &\geq \frac{1}{d} \left(\left(1 + \frac{1}{c_i} \right)^{\sum_{j|i \in S(j) \setminus \{k\}} y_j} - 1 \right) \cdot \left(1 + \frac{1}{c_i} \right)^{y_k} + \frac{1}{d \cdot c_i} \qquad (4.2) \\ &= \frac{1}{d} \left(\left(1 + \frac{1}{c_i} \right)^{\sum_{j|i \in S(j)} y_j} - 1 \right), \end{aligned}$$

Inequality (4.2) follows from the induction hypothesis.

Next, observe that the algorithm never updates any variable $x_i \geq 1$ (since it cannot be in any unsatisfied constraint). Since each $c_i \geq 1$ and $d \geq 1$ we have that $x_i < 1(1+1) + 1 = 3$. Combining this with Inequality (4.1) we get that:

$$3 \geq x_i \geq \frac{1}{d} \left(\left(1 + \frac{1}{c_i} \right)^{\sum_{j|i \in S(j)} y_j} - 1 \right).$$

Using again the fact that $c_i \geq 1$ and simplifying we get the desired result that:

$$\sum_{j|i \in S(j)} y_j \leq c_i \log_2(3d + 1) = c_i \cdot O(\log d)$$

□

The basic discrete algorithm is extremely simple and we show in the following many applications for it. However, we would like to derive a slightly different algorithm that is continuous and is more in the spirit of the primal-dual schema. This algorithm also helps us in gaining intuition for the right relationship between primal and dual variables. The algorithm is the following:

Algorithm 2:
Whenever a new primal constraint $\sum_{i\in S(j)} x_i \geq 1$ and the corresponding dual variable y_j appear:

1. While $\sum_{i\in S(j)} x_i < 1$:
   (a) Increase the variable y_j continuously.
   (b) For each variable x_i that appears in the (yet unsatisfied) primal constraint increase x_i according to the following function:

$$x_i \leftarrow \frac{1}{d}\left[\exp\left(\frac{\ln(1+d)}{c_i}\sum_{j|i\in S(j)} y_j\right) - 1\right].$$

Note that the exponential function for x_i contains variables y_j that correspond to future constraints. However, these variables are all initialized to 0, so they do not contribute to the value of the function. Although the algorithm is described in a continuous fashion it is not hard to implement it in a discrete fashion in any desired accuracy. We discuss the intuition of the exponential function we use after proving the following Theorem:

Theorem 4.2. *The algorithm produces:*

- *A fractional covering solution which is feasible and also $O(\log d)$-competitive.*
- *A fractional packing solution which is feasible and also $O(\log d)$-competitive.*

Proof. Let P and D be the values of the objective function of the primal and the dual solution the algorithm produce respectively. Initially, $P = D = 0$. Since our algorithm is "continuous" we use derivatives. We prove three simple claims:

1. The algorithm produces a primal (covering) feasible solution.
2. In each iteration j: $\frac{\partial P}{\partial y_j} \leq 2\ln(1+d)\cdot\frac{\partial D}{\partial y_j}$.
3. Each packing constraint in the dual program is feasible.

The Theorem then follows immediately by the three claims along with weak duality.
Proof of (1)**:** Consider a primal constraint $\sum_{i\in S(j)} x_i \geq 1$. During the jth iteration in which the the jth primal constraint and dual variable y_j appear, the algorithm increases the values of the variables x_i until the constraint is satisfied. Subsequent increases of variables cannot make the solution infeasible.
Proof of (2)**:** Whenever the algorithm updates the primal and dual solutions, $\frac{\partial D}{\partial y_j} = 1$. The derivative of the primal cost is:

$$\begin{aligned}
\frac{\partial P}{\partial y_j} &= \sum_{i \in S(j)} c_i \frac{\partial x_i}{y_j} \\
&= \sum_{i \in S(j)} c_i \left(\frac{\ln(1+d)}{c_i} \cdot \frac{1}{d} \left[\exp\left(\frac{\ln(1+d)}{c_i} \sum_{j | i \in S(j)} y_j \right) \right] \right) \\
&= \ln(1+d) \cdot \sum_{i \in S(j)} \left(\frac{1}{d} \left[\exp\left(\frac{\ln(1+d)}{c_i} \sum_{j | i \in S(j)} y_j \right) - 1 \right] + \frac{1}{d} \right) \\
&= \ln(1+d) \cdot \sum_{i \in S(j)} \left(x_i + \frac{1}{d} \right) \leq 2 \ln(1+d) \qquad (4.3)
\end{aligned}$$

The last inequality follows since the covering constraint is infeasible.

Proof of (3)**:** Consider any dual constraint $\sum_{j|i \in S(j)} y_j \leq c_i$. The corresponding variable x_i is always at most 1 since otherwise it cannot be in any unsatisfied constraint. Thus, we get that:

$$x_i = \frac{1}{d} \left[\exp\left(\frac{\ln(1+d)}{c_i} \sum_{j | i \in S(j)} y_j \right) - 1 \right] \leq 1$$

Simplifying we get the that:

$$\sum_{j | i \in S(j)} y_j \leq c_i$$

$\square$

Discussion: As can be seen from the proof, the basic discrete algorithm and the continuous algorithm are essentially the same. The main idea is that $(1 + \frac{1}{c_i})$ is approximately $\exp(\frac{1}{c_i})$. The function in the continuous algorithm is then approximated by Inequality 4.1 in Theorem 4.1. The approximate equality is true as long as c_i is not too small. This is why the discrete algorithm needs the assumption that $c_i \geq 1$. In addition, the discrete algorithm allows the primal variables to get values of more than 1 which is unnecessary (and can easily be avoided). For these reasons, the proof of the continuous algorithm is also a bit simpler. However, the description of the discrete algorithm is simpler and more intuitive.

The reader may wonder at this point how did we choose the function used in the algorithm for updating the primal and dual variables. We will try to give here a systematic way of deriving this function. Consider the point in time in which the jth primal constraint is given and assume that it is not satisfied. Our goal is to bound the derivative of the primal cost (denoted by P) as a function of the dual profit (denoted by D). That is, show that

$$\frac{\partial P}{\partial y_j} = \sum_{i \in S(j)} c_i \cdot \frac{\partial x_i}{\partial y_j} \leq \alpha \cdot \frac{\partial D}{\partial y_j},$$

where α is going to be the competitive factor. Suppose that the derivative of the primal cost satisfies:

$$\sum_{i \in S(j)} c_i \cdot \frac{\partial x_i}{\partial y_j} = A \cdot \sum_{i \in S(j)} \left(x_i + \frac{1}{d} \right). \tag{4.4}$$

Then, since $\sum_{i \in S(j)} x_i \leq 1$, $\sum_{i \in S(j)} \frac{1}{d} \leq 1$, and $\frac{\partial D}{\partial y_j} = 1$, we get that

$$A \cdot \sum_{i \in S(j)} \left(x_i + \frac{1}{d} \right) \leq 2A \cdot \frac{\partial D}{\partial y_j}.$$

Thus, $\alpha = 2A$. Now, satisfying Equality (4.4) requires solving the following differential equation for each $i \in S(j)$:

$$\frac{\partial x_i}{\partial y_j} = \frac{A}{c_i} \cdot \left(x_i + \frac{1}{d} \right).$$

It is easy to verify that the solution is a family of functions of the following form:

$$x_i = B \cdot \exp\left(\frac{A}{c_i} \sum_{\ell \ | i \in S(j)} y_\ell \right) - \frac{1}{d},$$

where B is any number. Next, we have the following two boundary conditions on the solution:

- Initially, $x_i = 0$, and this happens when $\frac{1}{c_i} \sum_{j | i \in S(j)} y_j = 0$.
- If $\frac{1}{c_i} \cdot \sum_{j | i \in S(j)} y_j = 1$, (i.e., the dual constraint is tight), then $x_i = 1$. (Then, the primal constraint is also satisfied.)

The first boundary condition gives $B = \frac{1}{d}$. The second boundary condition gives us $A = \ln(d+1)$. Putting everything together we get the exact function used in the algorithm.

We next describe a third algorithm for the problem. This algorithm is also continuous, but different from the previous one. The idea is to combine the primal dual schema into the online algorithm. This idea, turn out to be useful in some applications we discuss in later chapters. Again, let $d = \max_j |S(j)| \leq m$ be the maximal "size" of a constraint. The algorithm description is the following:

Algorithm 3:
Whenever a new primal constraint $\sum_{i \in S(j)} x_i \geq 1$ and the corresponding dual variable y_j appear:

1. While $\sum_{i \in S(j)} x_i < 1$:

   (a) Increase variable y_j continuously.

   (b) If $x_i = 0$ and $\sum_{j|i \in S(j)} y_j = c_i$ then set $x_i \leftarrow \frac{1}{d}$.

   (c) For each variable x_i, $\frac{1}{d} \leq x_i < 1$, that appears in the (yet unsatisfied) primal constraint, increase x_i according to the following function:

$$x_i \leftarrow \frac{1}{d} \exp\left(\frac{\sum_{j|i \in S(j)} y_j}{c_i} - 1\right).$$

First, note that the exponential function equals $\frac{1}{d}$ when $\sum_{j|i \in S(j)} y_j = c_i$ and so the algorithm is well defined. We next prove the following Theorem:

Theorem 4.3. *The algorithm produces:*

- *A fractional covering solution which is $O(\log d)$-competitive.*

- *A fractional packing solution which is 2-competitive and violates each packing constraint by at most $O(\log d)$.*

We remark that similarly to the basic discrete algorithm, it is not hard to make the packing solution feasible (and $O(\log d)$-competitive) by scaling down each y_j by $O(\log d)$.

Proof. Let P and D be the values of the objective function of the primal and dual solutions, respectively. Initially, $P = D = 0$. Since our algorithm is continuous, we will use derivatives. We prove three simple claims:

1. The algorithm produces a primal (covering) feasible solution.

2. Each packing constraint in the dual program is violated by at most $O(\log d)$.

3. $P \leq 2D$.

The Theorem then follows immediately by the three claims along with weak duality.
Proof of (1)**:** Consider a primal constraint $\sum_{i \in S(j)} x_i \geq 1$. During the jth round the algorithm increases the values of the variables x_i until the constraint is satisfied. Subsequent increases of the variables cannot make the solution infeasible.
Proof of (2)**:** Consider any dual constraint $\sum_{j|i \in S(j)} y_j \leq c_i$. The corresponding variable x_i is always at most 1, since otherwise it cannot belong to any unsatisfied constraint. Thus, we get that:

$$x_i = \frac{1}{d} \exp\left(\frac{\sum_{j|i \in S(j)} y_j}{c_i} - 1\right) \leq 1.$$

Simplifying we get the that

$$\sum_{j|i \in S(j)} y_j \leq c_i(1 + \ln d).$$

Proof of (3)**:** We partition the total cost of the primal into two parts. Let C_1 be the contribution of the primal cost from Step (1b), due to the increase of primal variables from 0 to $\frac{1}{d}$. Let C_2 be the contribution of the primal cost from Step (1c) of the algorithm. It is also beneficial for the reader to observe the similarity between the arguments used for bounding C_1 and those used for the proof of the approximation ratio of the primal-dual algorithm in Section 2.2.3.

Bounding C_1**:** Let $\tilde{x}_i = \min(x_i, \frac{1}{d})$. Our goal is to bound $\sum_{i=1}^{n} c_i \tilde{x}_i$. To do this we observe the following. First, the algorithm guarantees that if $x_i > 0$, and therefore $\tilde{x}_i > 0$ then:

$$\sum_{j|i \in S(j)} y_j \geq c_i \quad \text{(primal complementary slackness)} \tag{4.5}$$

Next, if $y_j > 0$ then:

$$\sum_{i \in S(j)} \tilde{x}_i \leq 1 \quad \text{(dual complementary slackness)} \tag{4.6}$$

Inequality (4.6) follows since $\tilde{x}_i \leq \frac{1}{d}$. Thus, even if for all i, $\tilde{x}_i = \frac{1}{d} \leq \frac{1}{|S(j)|}$, then $\sum_{i \in S(j)} \tilde{x}_i \leq 1$. By the primal and dual complementary slackness conditions we get that:

$$\begin{aligned}
\sum_{i=1}^{n} c_i \tilde{x}_i &\leq \sum_{i=1}^{n} \left(\sum_{j|i \in S(j)} y_j \right) \tilde{x}_i && (4.7)\\
&= \sum_{j=1}^{m} \left(\sum_{i \in S(j)} \tilde{x}_i \right) y_j && (4.8)\\
&\leq \sum_{j=1}^{m} y_j && (4.9)
\end{aligned}$$

Where Inequality (4.7) follows from Inequality (4.5). Equality (4.8) follows by changing the order of summation. Inequality (4.9) follows from Inequality (4.6). Thus, we get that C_1 is at most the dual cost.

Bounding C_2: Whenever the algorithm updates the primal and dual solutions, $\frac{\partial D}{\partial y_j} = 1$. It is easy to verify that the derivative of the primal cost is:

$$\begin{aligned} \frac{\partial P}{\partial y_j} &= \sum_{i \in S(j)} c_i \frac{\partial x_i}{y_j} \\ &= \sum_{i \in S(j)} c_i \frac{x_i}{c_i} \leq 1. \end{aligned} \tag{4.10}$$

The last inequality follows since the covering constraint is infeasible at the update time. Thus, C_2 is also bounded from above by the dual cost.

□

4.3 Lower Bounds

In this Section we show that the competitive ratios we obtained in Section 4.2 are optimal up to constants. We prove a lower bound for the online packing problem and another lower bound for the online covering problem.

Lemma 4.4. *There is an instance of the online fractional packing problem with n constraints, such that for any $B-$competitive online algorithm there exists a constraint for which $\sum_{j|i \in S(j)} y_j \geq c_i \cdot \frac{1}{B}\left(1 + \frac{\log_2 n}{2}\right) = c_i \cdot \Omega(\frac{\log n}{B})$.*

Proof. Consider the following instance with $n = 2^k$ packing constraints. The right hand side of each packing constraint is 1. In the first round a new variable $y(1,1)$ that belongs to all constraints arrives. In the second round two variables $y(2,1)$ and $y(2,2)$ arrive. $y(2,1)$ belongs to the first 2^{k-1} constraints and $y(2,2)$ belongs to the last 2^{k-1} constraints. In the third round four dual variables $y(3,1), y(3,2), y(3,3),$ and $y(3,4)$ arrive that belong each to 2^{k-2} packing constraints. The process ends in the $(k+1)$st round in which 2^k variables arrive, each belong to a single packing constraint. The optimal solution after the ith round is to set the new 2^{i-1} variables to 1. Since the algorithm is B-competitive we get the following set of constraints:

For each $1 \leq i \leq k+1$:

$$\sum_{j=1}^{i} \sum_{\ell=1}^{2^{j-1}} y(j,\ell) \geq \frac{2^{i-1}}{B}$$

Multiplying the $k+1$st inequality by 1 and each inequality i, $1 \leq i \leq k$, by 2^{k-i} and summing up, we get that:

$$\sum_{j=1}^{k+1} 2^{k-j+1} \cdot \left(\sum_{\ell=1}^{2^{j-1}} y(j,\ell) \right) \geq \frac{1}{B}\left(k 2^{k-1} + 2^k\right)$$

However, this sum is also the sum over all packing constraints. Thus, by an averaging argument since there are 2^k constraints we get that there exists a constraint whose right hand side is at least $\frac{1}{2^k \cdot B}\left(k2^{k-1} + 2^k\right) = \frac{1}{B}\left(1 + \frac{k}{2}\right)$. Since $n = 2^k$ we get the desired bound. □

Lemma 4.5. *There is an instance of the online fractional covering problem with n variables such that any online algorithm is $\Omega(\log n)$-competitive on this instance.*

Proof. Consider the following instance with $n = 2^k$ variables $x_1, x_2, \ldots, x_n$. The first constraint that arrives is $\sum_{i=1}^{n} x_i \geq 1$. If $\sum_{i=1}^{n/2} x_i \geq \sum_{i=n/2+1}^{n} x_i$ then the next constraint that arrives is $\sum_{i=n/2+1}^{n} x_i \geq 1$, otherwise the constraint $\sum_{i=1}^{n/2} x_i \geq 1$ arrives. This process of halving and continuing with the smaller sum goes on until we remain with a single variable. The optimal offline solution satisfying all the constraints is to set the last variable to one. However, for any online algorithm, the sum of the variables in each round that do not appear in the next covering constraint is at least $1/2$. There are $k+1$ rounds and thus the cost of any online algorithm is at least $1 + \frac{k}{2}$, concluding the proof. □

4.4 Two Warm-up Problems

In this Section we demonstrate the use of the online primal-dual framework on two simple examples, a covering problem and a packing problem.

4.4.1 The Online Set Cover Problem

Consider an online version of the offline set cover problem discussed in Section 2.2. In the online version of the problem a subset of the elements X arrive one-by-one in an online fashion. The algorithm has to cover each element upon arrival. The restriction is that sets already chosen to the cover by the online algorithm cannot be "returned".

This online setting exactly fits the online covering setting, since whenever a new element arrives a new constraint is added to the set cover linear formulation. Hence, we can use any of the algorithms presented in Section 4.2 to derive a monotonically increasing fractional solution to the set cover problem.

Getting a randomized integral solution is extremely simple. We may simply use the same randomized rounding algorithm that appears in Section 2.2.2. Note that the algorithm chooses a-priori in random a threshold $\Theta(s) \in [0,1]$ for each set s. The algorithm then chooses the set s to the cover if $x_s \geq \Theta(s)$. Since x_s is monotonically increasing, the online algorithm simply chooses the set s to the cover once $x_s \geq \Theta(s)$. The analysis is straightforward proving that the algorithm produces a solution covering all requested elements with high probability and its expected cost is $O(\log n)$ times the fractional solution. Since the fractional solution is $O(\log m)$-competitive with respect to the optimal solution, we get that the integral algorithm is $O(\log n \log m)$-competitive. In Chapter 6 we show how to obtain a deterministic online algorithm for the set cover problem with the same competitive ratio.

Primal	Dual
Minimize: $\sum_{e \in E} u(e)x(e) + \sum_{r_i} z(r_i)$	Maximize: $\sum_{r_i} \sum_{P \in \mathbb{P}(r_i)} f(r_i, P)$
Subject to:	Subject to:
$\forall r_i \in \mathbb{R}, P \in \mathbb{P}(r_i)$: $\sum_{e \in P} x(e) + z(r_i) \geq 1$	$\forall r_i \in \mathbb{R}$: $\sum_{P \in \mathbb{P}(r_i)} f(r_i, P) \leq 1$
	$\forall e \in E$: $\sum_{r_i \in \mathbb{R}, P \in \mathbb{P}(r_i) \mid e \in P} f(r_i, P) \leq u(e)$

Figure 4.2: The splittable routing problem (Maximize) and its corresponding primal problem.

4.4.2 Online Routing

In this section we give a simple example of an online packing problem. We study the problem of maximizing the throughput of scheduled virtual circuits. In the simplest version of the problem we are given in advance a graph $G = (V, E)$ with capacities $u(e)$ on the edges. A set of requests $r_i = (s_i, t_i)$ $(1 \leq i \leq n)$ arrives in an online fashion. To serve a request, the algorithm chooses a path between s_i and t_i and allocates a bandwidth of one unit on this path. The decisions of the algorithm are irrevocable, and all requests are permanent, meaning that once accepted they stay forever. If the total capacity routed on edge e is $\ell \cdot u(e)$, we say that the *load* on edge e is ℓ. Ideally, the total bandwidth allocated on any edge should not exceed its capacity (load $\ell \leq 1$). The total profit of the algorithm is the number of requests served and its performance is measured with respect to the maximum number of requests that could have be served (offline).

In a fractional version of the problem the allocation is not restricted to an integral bandwidth of either zero or one; instead, we can allocate to each request a fractional bandwidth in the range $[0, 1]$. In addition, the bandwidth allocated to a request can be divided between several paths. This problem is an online version of the maximum multi-commodity flow problem. We describe the problem as a packing problem in Figure 4.2. For $r_i = (s_i, t_i)$, let $\mathbb{P}(r_i)$ be the set of simple paths between s_i and t_i. For each $P \in \mathbb{P}(r_i)$, the variable $f(r_i, P)$ corresponds to the amount of flow (service) given to request r_i on the path P. The first set of constraints guarantees that each client gets at most a fractional flow (bandwidth) of 1. The second set of constraints are the capacity constraints of the edges. In the primal problem we assign a variable $z(r_i)$ to each request r_i and a variable $x(e)$ to each edge in the graph.

This online setting exactly fits our online packing setting, as the new dual variables arrive whenever a new request arrives. However, in each round it may happen that an exponential number of variables arrives. We show in the following that we can still overcome this problem and get an efficient algorithm. We present two algorithms, each having different properties, for the problem. Let $d \leq n$ be the diameter of the graph (the longest simple path between any two nodes). The first algorithm is the following:

Routing Algorithm 1:
When a new request $r_i = (s_i, t_i)$ arrives:

1. if there exists a path $P \in \mathbb{P}(r_i)$ such that $\sum_{e \in P} x(e) < 1$:
   (a) Route the request on P and set $f(r_i, P) \leftarrow 1$.
   (b) Set $z(r_i) \leftarrow 1$.
   (c) For each $e \in P$: $x(e) \leftarrow x(e)(1 + \frac{1}{u(e)}) + \frac{1}{|P| \cdot u(e)}$, where $|P|$ is the length of the path P.

Theorem 4.6. *The algorithm is* 3*-competitive while violating the capacity of each edge by at most* $O(\log d)$ *(i.e. the load on each edge is at most* $O(\log d)$*).*

The proof of Theorem 4.6 is almost identical to the proof of Theorem 4.1 and we leave it as a simple exercise to the reader. The main observation is that there is no need to deal with the exponential number of dual variables. The algorithm only needs to check the condition in Line (1). If, for example, $P(r_i)$ is the set of all simple paths between s_i and t_i then this condition can be checked by computing a shortest path between s_i and t_i.

The above algorithm routes more requests on the edges then the capacity of the edges. We next show a different algorithm that fully respects the edges' capacities.

Routing Algorithm 2:
Initially: $x(e) \leftarrow 0$.
When a new request $r_i = (s_i, t_i, \mathbb{P}(r_i))$ arrives:

1. If there exists a path $P(r_i) \in \mathbb{P}(r_i)$ of length < 1 with respect to $x(e)$:
   (a) Route the request on **any** path $P \in \mathbb{P}(r_i)$ with length < 1.
   (b) $z(r_i) \leftarrow 1$.
   (c) For each edge e in $P(r_i)$:
$$x(e) \leftarrow x(e) \exp\left(\frac{\ln(1+n)}{u(e)}\right) + \frac{1}{n}\left[\exp\left(\frac{\ln(1+n)}{u(e)}\right) - 1\right]$$

Theorem 4.7. *The algorithm is* $O\left(u(\min) \cdot \left[\exp\left(\frac{\ln(1+n)}{u(\min)}\right) - 1\right]\right)$*-competitive and does not violate the capacity constraints. If* $u(\min) \geq \log n$ *then the algorithm is* $O(\log n)$*-competitive.*

Proof. Note first that the function $\left(u(e) \cdot \left[\exp\left(\frac{\ln(1+n)}{u(e)}\right) - 1\right]\right)$ is monotonically decreasing with respect to $u(e)$. Thus, when a request r_i is routed, the increase of the primal cost is at most:

$$1+\sum_{e\in P} u(e)\left(x(e)\left[\exp\left(\frac{\ln(1+n)}{u(e)}\right)-1\right]+\frac{1}{n}\left[\exp\left(\frac{\ln(1+n)}{u(e)}\right)-1\right]\right).$$

This expression is at most $2\left(u(\min)\cdot\left[\exp\left(\frac{\ln(1+n)}{u(\min)}\right)-1\right]\right)+1$. This follows since $z(r_i)$ is set to one, and edges on the path P satisfy $\sum_{e\in P} x(e)\le 1$. Each time a request is routed, the dual profit is 1. Thus, the ratio between the primal and dual solutions is at most that ratio.

Second, observe that the algorithm maintains a feasible primal solution at all times. This follows since $z(r_i)$ is set to one for each request in which the distance between s_i and t_i with respect to $x(e)$-variables is less than 1.

Finally, it remains to prove that the algorithm routes at most $u(e)$ requests on each edge e, and so the dual solution it maintains is feasible. To this end, observe that for each edge e, the value $x(e)$ is the sum of a geometric sequence with initial value $\frac{1}{n}\left[\exp\left(\frac{\ln(1+n)}{u(e)}\right)-1\right]$ and multiplier $\exp\left(\frac{\ln(1+n)}{u(e)}\right)$. Thus, after $u(e)$ requests are routed through edge e, the value $x(e)$ is:

$$\begin{aligned}
x(e) &= \frac{1}{n}\cdot\left(\exp\left(\frac{\ln(1+n)}{u(e)}\right)-1\right)\cdot\frac{\exp\left(\frac{u(e)\ln(1+n)}{u(e)}\right)-1}{\exp\left(\frac{\ln(1+n)}{u(e)}\right)-1}\\
&= \frac{1}{n}\cdot(1+n-1)\ge 1.
\end{aligned}$$

Since the algorithm never routes requests on edges for which $x(e)\ge 1$, we are done.

Finally, it is not hard to verify that when $u(\min)\ge\log n$, then

$$2\left(u(\min)\cdot\left[\exp\left(\frac{\ln(1+n)}{u(\min)}\right)-1\right]\right)+1=O(\log n).$$

□

4.5 Notes

The definitions of the online covering/packing framework along with the basic algorithms in Section 4.2 and the lower bounds in Section 4.3 are based on the work of Buchbinder and Naor [32]. These algorithms draw on ideas from previous algorithms by Alon et al. [3, 4]. The third basic algorithm that incorporate the primal-dual schema into the online algorithm is based on the later work of Bansal, Buchbinder and Naor [14]. The online set cover problem was considered in [3]. There, they gave a deterministic algorithm for the problem that is discussed later in Chapter 6. The second routing algorithm in Section 4.4.2 appeared in [33]. It is basically an alternative description and analysis of a previous algorithm by Awerbuch, Azar and Plotkin [11].

There is a long line of work on generating a near-optimal fractional solution for *offline* covering and packing problems, e.g. [82, 77, 91, 54, 52, 73]. Generating such a solution for the *offline* covering problem with upper bounds on the variables was considered in [51, 53]. All these methods take advantage of the offline nature of the problems. We remark that several of these methods use primal-dual analysis. For example, in [54] some of their algorithms that solves packing and covering linear formulations repeatedly updates primal and dual variables in an unsatisfied primal constraint. Therefore, our approach can be viewed as an adaptation of these methods to the context of online computation.

Chapter 5

Graph Optimization Problems

In this chapter we describe applications of the primal dual approach for a wide range of graph and network optimization problems, focusing on problems that arise in the study of connectivity and cuts in graphs. The first step is to formulate the problem as either an online covering problem or an online packing problem. This enable us to use the generic algorithms in Section 4.2, yielding a fractional solution for the problem. We then transform a known offline rounding method into an online rounding method to obtain an integral solution. This part of the algorithm is problem dependent.

Connectivity and cut problems in graphs are defined on an input graph $G = (V, E)$ (directed or undirected), a cost function $c : E \to \mathbb{R}^+$, and a requirement function f (to be defined later). The goal is to find a minimum cost subgraph that satisfies the requirement function f. Our model is online; that is, the requirement function is not known in advance and it is given "step by step" to the algorithm, while the input graph is known in advance. We consider an online version of network design problems which we call *generalized connectivity*. The requirement function is a set of demands of the form $D = (S, T)$, where S and T are subsets of vertices in the graph such that $S \cap T = \emptyset$. A feasible solution is a set of edges, such that for each demand $D = (S, T)$ there is a path from a vertex in S to a vertex in T. Examples of problems belonging to this class are Steiner trees, generalized Steiner trees, and the group Steiner problem. Less obvious examples are the set cover problem and non-metric facility location.

Cut problems in graphs involve separating sets of vertices from each other. We concentrate on a family of cut problems which we call *generalized cuts*. The requirement function is a set of demands of the form $D = (S, T)$, where S and T are subsets of vertices in the graph such that $S \cap T = \emptyset$. A feasible solution is a set of edges that separates for each demand $D = (S, T)$, any two vertices $s \in S$ and $t \in T$. Examples of problems belonging to this class are the multiway cut problem and the multicut problem (see e.g., [88]).

There is a natural linear programming relaxation for the problems that we are considering. For generalized connectivity problems, a feasible fractional solution associates a fractional weight (capacity) with each edge, such that for each demand $D = (S, T)$ a unit of flow can be sent from S to T, where the flow on each edge does not exceed its weight. For generalized cuts, a feasible fractional solution associates a fractional weight (length) with each edge, which we interpret as inducing a distance function. The constraint is that for each demand $D = (S, T)$, the distance between any two vertices $s \in S$ and $t \in T$ is at least 1. This linear programming relaxation is very useful for computing (offline) an approximate solution for many problems that are special case of the connectivity/cuts problem. Please refer to [88] for more details. We note that fractional solutions have a motivation of their own in certain network design problems and bandwidth allocation problems (see, for example, [82]).

5.1 Formulating the Problem

Let $G = (V, E)$ be a graph (directed or undirected) with cost function $c : E \to \mathbb{R}^+$ associated with the edge set E. Suppose further that there is a weight function (or capacity function) $w : E \to \mathbb{R}^+$ associated with the edge set E. The *cost* of w is defined to be $\sum_{e\in E} c_e w_e$.

Let $A \subset V$ and $B \subset V$ be subsets of V such that $A \cap B = \emptyset$. Let G' be the graph obtained from G by adding a super-source s connected to all vertices in A and a super-sink t connected to all vertices in B. The edges from s to A are directed into A and have infinite weight, and the edges from B to t are directed into t and have infinite weight. We say that there is a flow from A to B of value α if there exists a valid flow function that sends α units of flow from s to t satisfying the capacity function w. The shortest path from A to B is defined to be the shortest path with respect to w from any vertex $u \in A$ to any vertex $v \in B$ (i.e. the minimal distance between any pair of vertices in A and B). A requirement function is a set of demands of the form $D_i = (S_i, T_i)$, $1 \le i \le k$, where $S_i \subset V$, $T_i \subset V$ and $S_i \cap T_i = \emptyset$.

We first define the *generalized connectivity* problem. The input for the problem is a graph $G = (V, E)$ with cost function $c : E \to \mathbb{R}^+$ and a requirement function. A feasible *integral* solution is an assignment of weights (capacities) w from $\{0, 1\}$ to E, such that for each demand $D_i = (S_i, T_i)$, $1 \le i \le k$, there is a flow from S_i to T_i of value at least 1. A feasible *fractional* solution is an assignment of weights (capacities) w from $[0, 1]$ to E, such that for each demand $D_i = (S_i, T_i)$, $1 \le i \le k$, there is a flow from S_i to T_i of value at least 1. We note that the flow constraint has to be satisfied for each demand (S_i, T_i) separately. The cost of a solution is defined to be the cost of w. It is possible to define an LP relaxation for the fractional offline problem. For each demand D_i let $C(D_i)$ be the set of cuts that cut S_i from T_i. The LP formulation is then the following:

$$(P) : \min \sum_{e\in E} c_e w_e$$

Subject to:
For all $1 \le i \le k$ and each cut $C \in C(D_i)$:

$$\sum_{e\in C} w_e \ge 1$$

$$\forall e \in E, \qquad w_e \geq 0$$

We next define the *generalized cuts* problem. The input for this problem is again a graph $G = (V, E)$ with cost function $c : E \to \mathbb{R}^+$ and a requirement function. A feasible *integral* solution is a set of edges $E' \subseteq E$ that separates for each demand $D_i = (S_i, T_i)$ any two vertices $a \in S_i$ and $b \in T_i$. Alternatively, we can think of each edge $e \in E'$ as having weight $w(e) = 1$. Thus, the weight function w induces a distance function on the graph such that the distance between vertices separated by E' is at least 1. A feasible *fractional* solution is an assignment of weights w from $[0, 1]$ to E, such that for each demand $D_i = (S_i, T_i)$, $1 \leq i \leq k$, the distance induced by w between each $a \in S_i$ and $b \in T_i$ is at least 1. The cost of a solution is defined to be the cost of w. Again it is possible to obtain a covering LP formulation for the generalized cuts problem. For each demand D_i let $P(D_i)$ be the set of paths between any two vertices in S_i and T_i. The LP formulation is then the following:

$$(P) : \min \sum_{e \in E} c_e w_e$$

Subject to:
For all $1 \leq i \leq k$ and each cut $P \in P(D_i)$:

$$\sum_{e \in P} w_e \geq 1$$

$$\forall e \in E, \qquad w_e \geq 0$$

In an online setting, the graph $G = (V, E)$ along with the cost function c is known to the algorithm (as well as to the adversary) in advance. The set of requests of the form $D_i = (S_i, T_i)$ is then given one-by-one to the algorithm in an online fashion. Upon arrival of a new demand, the algorithm must satisfy it by increasing the weights of edges in the graph. However, the algorithm is not allowed to decrease the weight of an edge. Thus, previous demands remain satisfied.

Online Algorithm: It is not hard to see that the online setting of the generalized connectivity and the generalized cuts problems perfectly fit the primal-dual framework. In particular, whenever a new demand D_i arrives, the new set of constraints that correspond to all the cuts/paths between S_i and T_i are added to the LP. Although this may be an exponential number of constraints, it is still possible to use the algorithms from Chapter 4 for solving the problem, since it suffices to either determine that a solution is feasible, or find an unsatisfied primal constraint. This can be done easily using a maximum flow computation from the set S_i to the set T_i for the case of generalized connectivity, or a shortest path computations in the case of generalized cuts. Thus, it is possible to obtain online a monotonically increasing fractional solution which is $O(\log m)$-competitive. More precisely, it is possible to improve the factor to $O(\log |C_{\max}|)$, where $C_{\max}$ is the maximum cut in the graph in the case of generalized connectivity. In the case of generalized cuts, the competitive ratio can be improved to $O(\log d)$, where d is the diameter of the graph. We next define the special cases that we consider in the context of generalized connectivity.

5.2 The Group Steiner Problem on Trees

In this section we demonstrate how it is possible to derive a randomized integral solution to a special problem of the generalized connectivity by transforming an offline randomized rounding method into an online randomized rounding method. We demonstrate these ideas using the group steiner problem on trees.

The *group Steiner tree* problem in a rooted tree is defined as follows. We are given a rooted tree $T = (V, E, r)$ with non-negative cost function $c : E \to \mathbb{R}^+$, and groups $g_1, g_2, \ldots g_k \subset V$. Let r denote the root of the tree T. The objective is to find a minimum cost rooted subtree T' that contains at least one vertex from each of the groups g_i, $1 \leq i \leq k$. That is, using the terminology of the generalized connectivity problem, each request is of the form (r, g_i). In the online setting of the group steiner problem the groups arrive one by one and the algorithm has to choose additional edges to its solution upon arrival of a new request, so that the solution contains at least one vertex from the new group. It is easy to verify that the group steiner problem is a special case of the online generalized connectivity problem.

The group Steiner tree problem has an $O(\log n \log k)$-approximation algorithm, where k is the number of groups, and n is the number of leaves in the tree [55]. This approximation is based on a clever randomized rounding technique. In general (i.e., undirected) graphs, the best approximation factor known for the group Steiner problem is $O(\log^2 n \log k)$ by combining [55] with [45]. We next show how to derive an online randomized rounding algorithm for the problem. This method basically imitates the offline randomized rounding method of [55].

The randomized rounding method we propose covers each group with probability $\Omega(1/\log N)$, where N is the maximum size of any group. In addition, its expected cost is at most the cost of the fractional solution. We then run $O(\log k \log N)$ independent trials of this randomized rounding method in parallel, where k is the number of groups asked by the adversary. The algorithm takes to the solution each edge that was selected in any of the trials. Using simple probabilistic analysis we get that our algorithm has a competitive ratio of $O(\log n \log k \log N)$ and each of the groups is covered with probability at least $1 - 1/k$. In order to guarantee that the algorithm produces a feasible solution, we can use the shortest path to a group in case the algorithm fails to cover the group. The cost of this path is certainly a lower bound on the optimal solution, and since this event happens with probability at most $1/k$, it changes the expected competitive ratio of the algorithm by a negligible factor. Since we do not know in advance the value of k we can increase the number of trials gradually by doubling k whenever necessary. Next, we propose an online randomized rounding method and analyze its performance.

Initially, the algorithm starts with an empty cover $\mathbb{C} = \emptyset$. Applying the technique of [55] requires that the fractional weights on a path from the root to any vertex are monotonically decreasing. However, the fractional solution that our algorithm computes may not necessarily satisfy this property. Therefore, we define the weight of each edge to be the maximum flow that can be routed through this edge to any vertex in its subtree. In the following we abuse notation and let w_e denote the flow on edge e, instead of the actual weight of e. Since the flow routed on each edge is at most its weight, we note that this can only decrease the fractional value of the solution serving as our baseline for bounding the competitive analysis.

Consider an iteration in which the fractional weight of some edges is augmented. Since the weight of an edge is the maximum flow that can be routed through it, the fractional weight of an edge can be augmented when the algorithm augments the weights of other edges as well. If the weight of several edges is augmented at the same iteration, the rounding algorithm considers the edges one by one, according to a topological ordering, starting from the edges closer to the root. Let w_e and $w'_e = w_e + \delta_e$ be the weight of edge e before and after the weight augmentation, respectively. Let δ_e be the change in the weight of e. Let $e(p)$ be the edge adjacent to e and closer to the root r. This definition is, of course, only relevant if the edge e is not incident on the root r. The rounding algorithm randomly chooses the edges to the solution by the following scheme.
Consider all edges for which $\delta_e > 0$ in any topological order:

- If $w'_e > 1$, add e to $\mathbb{C}$.
- If e is incident on r, or $w'_{e(p)} > 1$, add e to $\mathbb{C}$ with probability $\delta_e/(1 - w_e)$.
- If $e(p) \in \mathbb{C}$, add e to $\mathbb{C}$ with probability $\delta_e/(w'_{e(p)} - w_e)$.

Note that for each edge e that is not incident on the root, $\delta_e/(w'_{e(p)} - w_e) \le \delta_e/(w'_e - w_e) = 1$, since $w'_e \le w'_{e(p)}$. Thus, the probabilities are well defined. Furthermore, note that $\mathbb{C}$ induces a connected subtree of T. This follows since the edges that were augmented at the same iteration are considered in topological order and each edge may be added to $\mathbb{C}$ only if the path connecting it to the root r is already in $\mathbb{C}$. The following lemma proves a basic important property of the randomized rounding method. The proof that is based on a simple induction is omitted.

Lemma 5.1. *For each edge e, at the end of each iteration, the probability that $e \in \mathbb{C}$ is w'_e. If $w_e > 1$, then $e \in \mathbb{C}$ with probability 1.*

The next lemma follows from linearity of expectation.

Lemma 5.2. *At the end of each iteration, the expected cost of the edges in $\mathbb{C}$ is at most $\sum_{e \in \mathbb{T}} c_e w'_e$, where w'_e is the weight of edge e at the end of the iteration.*

Let N be the maximum size of a group $g = \{\{v_1, v_2, \ldots, v_k\}$. Let w_g be the total flow that can be routed to the vertices in g simultaneously. The next lemma bounds from below the probability that any group g with $w_g > 1$ is covered. The proof is again based on showing that the some properties of the probability distribution that were needed to prove the same result in the offline case are kept the same online. We omit the proof.

Lemma 5.3. *In any iteration, if, for a group $g = \{v_1, v_2, \ldots, v_k\}$, $w_g \geq 1$, then the probability that there exists $v_i \in \mathbb{C}$ $(1 \leq i \leq k)$ is $\Omega(1/\log N)$.*

To conclude, we state the performance of the randomized algorithm for the online group Steiner on trees.

Theorem 5.4. *There is a randomized online algorithm for the group Steiner problem in trees with a competitive ratio of $O(\log^2 n \log k)$.*

5.2.1 The Group Steiner Problem on General Graphs

It is possible to combine our Theorem with the results in [45] similarly to what was done in the offline case. This gives the following result for the online group Steiner in general graphs.

Theorem 5.5. *There is a randomized online algorithm for the group Steiner problem in general graphs with a competitive ratio of $O(\log^3 n \log k)$.*

5.3 Notes

The results in this chapter are based on the work of Alon et al. [4]. The online fractional is analyzed there via a potential function method rather than by the primal-dual approach. The paper contains several other problems that can be solved using the same approach. Online network optimization problems have been studied extensively. The online Steiner problem was considered in [62] who gave an $O(\log n)$-competitive algorithm and showed that in a general metric space this is indeed best possible. Their algorithm greedily connects the new arrived terminal to the current sub-tree via a shortest path. We remark that the analysis of this algorithm has also a primal-dual interpretation. The generalized Steiner problem was considered in [9], where an $O(\log^2 n)$-competitive algorithm is given. This was later improved to an $O(\log n)$-competitive ratio algorithm by [22].

There is a vast literature on efficient (offline) approximation algorithms for problems involving connectivity and cuts. The reader is referred to [61, 88] for more details. In particular, the offline version of the generalized connectivity problem was considered in [36] who gave a poly-logarithmic (offline) approximation to it.

Chapter 6

The Online Set Cover Problem

In Section 4.4.1 we saw a simple randomized $O(\log m \log n)$-competitive algorithm for the online set cover problem. An interesting question is whether it is possible to obtain a deterministic algorithm for the problem with the same competitive ratio. A common approach for obtaining deterministic algorithms is derandomization, which means transforming randomized algorithms into deterministic algorithms. For more details about derandomization methods we refer the reader to [6].

One of the basic methods for derandomization is the so-called method of conditional expectations or pessimistic estimators [6]. In this case one come up with a function that "guides" the deterministic algorithm to a "good" solution. Interestingly, for the online set cover problem it is possible to apply such derandomization methods that actually derandomize the algorithm in an online fashion. This leads to a deterministic $O(\log m \log n)$-competitive algorithm for the problem.

6.1 Obtaining a Deterministic Algorithm

It is quite easy to verify that if the set system is unknown in advance to the algorithm then any deterministic algorithm is $\Omega(n)$-competitive. Thus, we assume that the universe of elements X is known to the algorithm along with the sets $\mathbb{S}$. It is unknown, however, which subset $X' \subseteq X$ of the elements the algorithm would eventually have to cover. Let $c(\mathbb{C}_{OPT})$ denotes the cost of the optimal solution. We design an algorithm that given a value $\alpha \geq c(\mathbb{C}_{OPT})$ produces a feasible solution with cost $O(\alpha \log m \log n)$. In case the algorithm is given a value $\alpha < c(\mathbb{C}_{OPT})$ it may fail.

Our algorithm guesses the value $c(\mathbb{C}_{OPT})$ by doubling. We start by guessing $\alpha = min_{s\in\mathbb{S}} c_s$, and run the algorithm with this value. If the algorithm fails we "forget" about all sets chosen so far to $\mathbb{C}$, update the value of α by doubling it, and continue on. We note that the cost of the sets that we have "forgotten" about can increase the cost of our solution by at most a factor of 2, since the value of α was doubled in each step. In the final iteration the value of α is at most $2c(\mathbb{C}_{OPT})$ and thus our actual cost is at most 4 times the cost of our sub-algorithm. Given a value $\alpha \geq c(\mathbb{C}_{OPT})$ our algorithm ignores all sets of cost exceeding α, and also chooses all sets of cost at most α/m to $\mathbb{C}$.

The algorithm we design is going to use an online algorithm that generates an $O(\log m)$-competitive fractional solution. This online algorithm maintains a monotonically increasing fraction w_s for each set s. Let $w_e = \sum_{s|e\in s} w_s$. Note that the fractional algorithm guarantees that for each element e that was requested $w_e \geq 1$. Initially, our algorithm starts with the empty cover $\mathbb{C} = \emptyset$. Define C to be the set of all elements covered by the members of $\mathbb{C}$. The following potential function is used throughout the algorithm:

$$\Phi = \sum_{e \notin C} n^{2w_e} + n \cdot \exp\left(\frac{1}{2\alpha}\sum_{s\in\mathbb{S}}(c_s\chi_{\mathbb{C}}(s) - 3w_s c_s \log n)\right).$$

The function $\chi_{\mathbb{C}}$ above is the characteristic function of $\mathbb{C}$, that is, $\chi_{\mathbb{C}}(s) = 1$ if $s \in \mathbb{C}$, and $\chi_{\mathbb{C}}(s) = 0$ otherwise. The algorithm is then the following:

Run the algorithm presented in Section 4.2 to produce a monotonically increasing fractional solution. When the weight of some set s is increased:

1. If $s \in \mathbb{C}$ do nothing. Otherwise: add the set s to $\mathbb{C}$ if after adding it the value of the potential function Φ does not exceed its value before the increase of the weight of s.

2. If the value of the potential function before increasing the weight of s exceeds its value after increasing the weight of s and possibly choosing s to $\mathbb{C}$ then return "FAIL".

In the following we analyze the performance of the algorithm in an iteration in which $\alpha \geq c(\mathbb{C}_{OPT})$, and prove that the algorithm never fails in that iteration.

Lemma 6.1. *Consider a step in which the weight of a set s is augmented by the algorithm. Let Φ_{start} and Φ_{end} be the values of the potential function Φ before and after the step, respectively. Then $\Phi_{end} \leq \Phi_{start}$. In particular, when $\alpha \geq c(\mathbb{C}_{OPT})$ the algorithm never fails.*

Proof. Consider first the case in which $s \in \mathbb{C}$. In this case the first term of the potential function is unchanged. The second term of the potential function is decreasing and therefore the claim holds.

The second case is when $s \notin \mathbb{C}$. The proof for this case is probabilistic. We prove that either including s in $\mathbb{C}$ or not including it does not increase the potential function. Let w_s and $w_s + \delta_s$ denote the weight of s before and after the step, respectively. Add set s to $\mathbb{C}$ with probability $1 - n^{-2\delta_s}$. (This is roughly the same as choosing s with probability $\delta_s/2$ and repeating this $4 \log n$ times.)

We first bound the expected value of the first term of the potential function. This is similar to the unweighted case. Consider an element $e \in X$ such that $e \notin C$. If $e \notin s$ then the term that is suitable to this element is unchanged in this step. Otherwise, let the weight of e before the step be w_e and the weight after the step is $w_e + \delta_s$. Before the step, element e contributes to the first term of the potential function the value n^{2w_e}. The probability that we do not choose the set s that contains element e is $n^{-2\delta_s}$. Therefore, the expected contribution of element e to the potential function after the step is at most $n^{-2\delta_s} n^{2(w_e+\delta_s)} = n^{2w_s}$. By linearity of expectation it follows that the expected value of $\sum_{e \notin \mathbb{C}} n^{2w_e}$ after the step is precisely its value before the step.

It remains to bound the expected value of the second term of the potential function. Let

$$T = n \cdot \exp\left(\frac{1}{2\alpha}\sum_{s\in\mathbb{S}}(c_s\chi_{\mathbb{C}}(s) - 3w_s c_s \log n)\right)$$

denote the value of the second term of the potential function before the step, and let T' denote the same term after the weight augmentation and the probabilistic choice of the set s. Recall that $s \notin \mathbb{C}$. Then,

$$\mathbf{Exp}[T'] = T \cdot \exp\left(-\frac{1}{2\alpha}\sum_{s\in\mathbb{S}} 3\delta_s c_s \log n\right) \cdot \mathbf{Exp}\left[\exp\left(\frac{1}{2\alpha} c_s \chi_{\mathbb{C}'}(s)\right)\right]. \tag{6.1}$$

Where $\chi_{\mathbb{C}'}(s) = 1$ is the indicator to the event that the set s is chosen to the cover which happens with probability $1 - n^{-2\delta_s}$. We bound this expectation in the following:

$$\begin{aligned}
\mathbf{Exp} \quad & \left[\exp\left(\frac{1}{2\alpha} c_s \chi_{\mathbb{C}'}(s)\right)\right] \\
= \quad & n^{-2\delta_s} + (1 - n^{-2\delta_s}) \cdot \exp\left(\frac{c_s}{2\alpha}\right) && (6.2) \\
\leq \quad & 1 - 2\delta_s \log n + 2\delta_s \log n \exp\left(\frac{c_s}{2\alpha}\right) && (6.3) \\
= \quad & 1 + 2\delta_s \log n \left(\exp\left(\frac{c_s}{2\alpha}\right) - 1\right) && (6.4) \\
\leq \quad & 1 + 2\delta_s \log n \frac{3c_s}{4\alpha} && (6.5) \\
\leq \quad & \exp\left(\frac{3\delta_s c_s \log n}{2\alpha}\right). && (6.6)
\end{aligned}$$

Here, (6.3) follows since for all $x \geq 0$ and $z \geq 1$, $e^{-x} + (1 - e^{-x}) \cdot z \leq 1 - x + x \cdot z$, (6.5) follows since $e^y - 1 \leq 3y/2$ for all $0 \leq y \leq 1/2$[1], and (6.6) follows since $1 + x \leq e^x$ for all $x \geq 0$. Plugging in (6.1), we conclude that the expected value of the second term after the augmentation step and random choices is at most

$$\mathbf{Exp}[T'] = T \cdot \exp\left(-\frac{1}{2\alpha}\sum_{s\in\mathbb{S}} 3\delta_s c_s \log n\right) \cdot \exp\left(\frac{1}{2\alpha} 3\delta_s c_s \log n\right) \leq T.$$

By linearity of expectation it now follows that $\mathbf{Exp}[\Phi_{\text{end}}] \leq \Phi_{\text{start}}$. Therefore, either choosing s to the cover or not choosing it to the cover does not increase the potential function. We conclude that after each step $\Phi_{\text{end}} \leq \Phi_{\text{start}}$. □

[1]Note that here we use the fact that $\alpha \geq c(\mathbb{C}_{OPT})$, since we ignored all sets with $c_s > \alpha$.

Theorem 6.2. *Throughout the algorithm, the following properties hold.*
(i)Every $e \in X$ of weight $w_e \geq 1$ is covered, that is, $e \in C$.
(ii) $\sum_{s \in \mathbb{C}} c_s = \alpha \cdot O(\log m \log n)$.

Proof. Initially, the value of the potential function Φ is at most $n \cdot n^0 + n < n^2$, and hence it stays smaller than n^2 during the whole algorithm. Therefore, if $w_e \geq 1$ for some e during the process, then $e \in C$, since otherwise the contribution of the term n^{2w_e} itself would be at least n^2. This proves part (i). To prove part (ii), note that by the same argument, throughout the algorithm

$$n \cdot \exp\left(\frac{1}{2\alpha}\sum_{s \in \mathbb{S}} c_s \chi_{\mathbb{C}}(s) - 3w_s c_s \log n\right) < n^2.$$

Therefore,

$$\sum_{s \in \mathbb{S}} c_s \chi_{\mathbb{C}}(s) \leq \sum_{s \in \mathbb{S}} 3w_s c_s \log n + 2\alpha \log n,$$

and the desired result follows from the fact that the fractional solution is $O(\log m)$-competitive. ☐

6.2 Notes

The results in this chapter are based on the work of Alon et al. [3]. The results were not originally in a primal-dual way. Alon et al. also proved that any deterministic algorithm for the online set cover problem is $\Omega\left(\frac{\log n \log m}{\log\log m + \log\log n}\right)$ for many values of m and n. In the unweighted version of the set cover problem all sets are of cost 1 and so the goal is to minimize the number of sets used to cover the elements. Buchbinder and Naor [32] used the improved offline rounding technique of [87] to obtain an $O(\log d \log(n/OPT))$-competitive algorithm, where d is the maximum frequency of an element (i.e., the maximum number of sets an element belong to), n is the number of elements and $\mathrm{O}PT$ is the optimal size of the set cover. We note that we do not fully understand when derandomization methods can be applied in online settings. Using derandomization methods Buchbinder and Naor [32] obtained an alternative routing algorithm that achieves the same competitiveness as the second routing algorithm in Section 4.4.2. Their algorithm is based on the basic algorithms of Section 4.2 along with an online derandomization of the rounding method in [84, 83]. Another de-randomization of an online algorithm was designed by Buchbinder, Jain and Naor [30] for the ad-auctions problem (See also Chapter 11).

Offline derandomization methods have been studied extensively. In particular, the method was used to transform fractional flow into integral flow [84, 83]. There are several other offline derandomization methods and we refer the reader to [6] for more details.

Another online variation of set cover problem was considered in [10]. There, we are also given m sets and n elements that arrive one at a time. However, the goal of the online algorithm is to pick k sets so as to maximize the number of elements that are covered. The algorithm only gets credit for elements that are contained in a set that it selected *before* or *during* the step in which the element arrived. The authors of [10] showed a *randomized* $\Theta(\log m \log \frac{n}{k})$ competitive algorithm for the problem, where the bound is optimal for many values of n, m, and k. An extension of the online set-cover problem was studied in [5]. They considered an admission control problem where the goal is to minimize the number of rejections. They solved it by reducing it to an instance of online set cover with repetitions, in which each element may be needed to be covered several times.

Chapter 7

The Metrical Task System Problem on a Weighted Star

In this chapter we study the metrical task system (MTS) problem on a metric $\mathbb{M}$ defined by a weighted star. The metrical task system is one of the earliest problems studied in the context of online computation. The problem captures many online scenarios. In the MTS problem we are given a finite metric space $\mathbb{M} = (V, d)$, where $|V| = N$. We view the points of $\mathbb{M}$ as states to which the algorithm belongs. The distance between the points of the metric measures the cost of transition between the possible states. We use a "server" notation and say that there is a server moving between the states and serving the requests. Each task (request) r in a metrical task system is associated with a vector $(r(1), r(2), \ldots, r(N))$, where $r(i)$ denotes the cost of serving r in state $i \in V$. In order to serve request r in state i the server has to be in this state. Upon arrival of a new request, the state of the system can first be changed to a new state (paying the transition cost), and only then the request is served (paying for serving the request in the new state). The transition cost between the states is assumed to be a metric. The objective is to minimize the total cost which is the sum of the transition costs and the service costs.

There is also an equivalent continuous time MTS model. In this model the algorithm is allowed to change states at any time t, which is a real number, and not only at integral times. The service cost is generalized in a straight forward way to be an integral instead of a sum. It is well known [28, Sec. 9.1.1] that any continuous time algorithm can be transformed to a discrete time algorithm without increasing the total cost. On the other hand, since the continuous time model is a relaxation of the discrete model it is clear that the optimal cost can only decrease.

We show in this section how to design an optimal online algorithm for the case where the metric space is a weighed star. The idea is to define an alternative MTS model and show that it is "equivalent" up to constant factors to the original model on a weighted star metric. We then show that the basic algorithms presented in Section 4.2 are applicable to the new model. Finally, we show that a randomized algorithm can be obtained by a simple rounding technique.

The leaves of the star are denoted by $\{1, 2, \ldots, N\}$. We present an $O(\log N)$-competitive online algorithm. We are going to charge the algorithm by $2d(i)$ whenever the server moves from state i to another state, say j. Thus, we are not going to charge the algorithm for the cost of moving into state j. This assumption can only add an additive term to the total cost which is independent of the request sequence (we do not charge for the last state change). From now on we abuse notation and let $d(i)$ denote the cost of moving from state i to a different state.

7.1 A Modified Model

We are going to start with the (equivalent) continuous time MTS model. In this model the algorithm is allowed to change states at any time t which is a real number and not only at integral times. We first define a new MTS model and show that on a star metric the cost of an optimal solution can only change by a constant factor. The high level idea of the new model is to cancel the transition cost incurred due to state change and pay only for serving the requests. To balance, we restrict the algorithm and allow it to change its state only if certain conditions are fulfilled. For each state we partition the time interval into phases. We permit the solutions to leave a state i only at the end of a phase (of state i). The first phase of each state starts at time $t = 0$. Phase p of state i starts at time $t_{p-1}(i)$ and ends at the earliest time $t_p(i)$ for which the accumulated cost of service at state i in the interval $[t_{p-1}(i), t_p(i)]$ is exactly $d(i)$.

We are now ready to describe the new MTS model in its full generality. An online algorithm is allowed to leave state i only at the end of a phase (of state i). The algorithm does not pay any transition cost when moving from one state to another. If the algorithm is in state i during phase p then it pays a cost $d(i)$. The algorithm pays the full cost of the phase even if it was in state i only during part of the phase p. This can happen if the algorithm moves to state i from i' in the middle of the pth phase of i (and at the end of a phase of state i'). Given a set of requests $\bar{\sigma}$, let $\mathrm{O}PT_n(\bar{\sigma})$ be the minimum offline cost of serving the set of requests in the new MTS model. Let $\mathrm{O}PT_o(\bar{\sigma})$ be the minimum offline cost of serving the set of requests in the standard MTS model. We prove the following two lemmas:

Lemma 7.1. *Let $\bar{\sigma}$ be a set of requests. Any solution S to $\bar{\sigma}$ in the standard MTS model with cost C can be transformed into a legal solution S' in the new MTS model with cost at most $2C$. In particular, $OPT_n(\bar{\sigma}) \leq 2OPT_o(\bar{\sigma})$.*

Proof. Let $t_1, t_2, \ldots, t_k$ be the times at which solution S changes its state. Let s_i be the state of the algorithm from time t_{i-1} until time t_i. During this time the algorithm pays for the cost for serving the requests in state s_i and then pays for moving out of s_i at t_i. We define a solution S' in the new model that imitates S but changes state only at the end of a phase. Initially, S' starts out from the same state s_1 as solution S. At any time t, if the algorithm is in state j in solution S', it waits until the some $t' \geq t$, when the phase in state j ends and then moves to the state in which solution S is at time t'. Note that if S' is already in the same state as S at time t', then S' does not change its state.

Clearly, S' is a feasible solution in the new MTS model by design as it changes its state only at the end of a phase. Also, it is easily seen that if solution S' is in state i at some time during $[t_{i-1}, t_i]$, then it stays in i at least until time t_i.

We bound the cost of S'. Let W_i be the total cost of serving the requests in state s_i during the time interval $[t_{i-1}, t_i]$. The cost of the solution S during $[t_{i-1}, t_i]$ is, therefore, $W_i + d(s_i)$. By construction, if S' moves to state s_i during $[t_{i-1}, t_i]$, then it leaves s_i no earlier than time t_i. The extra cost of solution S' with respect to S comes from two sources. First, solution S' may leave state s_i after time t_i. Second, solution S' may move to s_i in the middle of a phase (of state s_i), but it still pays the full cost of the phase. However, each of these can only increase the cost of solution S' by at most $d(s_i)$. Recall, that in the new MTS model the solution does not pay for changing the state, and thus its cost is at most $W_i + 2d(s_i)$, which is at most twice the cost incurred by solution S. $\square$

Lemma 7.2. *Let $\bar{\sigma}$ be a set of requests. Any solution S to $\bar{\sigma}$ in the new MTS model with cost C is a legal solution S' in the standard MTS model with cost at most $2C$.*

Proof. We run the solution S in the standard MTS model and upper bound its cost in this model. First, the cost of serving the requests in the standard MTS model is no more than the cost of serving the requests in the new model. Suppose that solution S visits state i during phase p. In this case it pays at least $d(i)$ in the new model, while the cost in the standard model is at most $d(i)$ (the cost could lower if S did not stay in state i during the entire phase).

Second, we claim that the transition cost of solution S in the standard model is no more than the service cost of S in the new model. This follows as S leaves any state i at most once during its phase (at the end of the phase) and hence the cost $d(i)$ of leaving the state can be charged to the service cost of the corresponding phase that just ended (which is also exactly $d(i)$). Thus, the cost of the solution S in the standard MTS model is at most twice its cost in the new MTS model. $\square$

From Lemmas 7.1 and 7.2 a c competitive algorithm in the new MTS model implies a $4c$-competitive algorithm in the standard MTS model, and hence it suffices to consider the new MTS model.

7.2 The Algorithm

We next describe a simple linear programming formulation for the offline problem in the new MTS model. Our online algorithm will generate a fractional solution to this linear program. We later show how to transform this fractional solution to a randomized integral solution. Let $x(i,p)$ be an indicator to the event that the solution is in state i during the pth phase. We relax the solution and allow the algorithm to be at time t in several states as long as the sum of the fractions of the states is at least 1. (The latter constraint is valid since our objective function is minimization.). Let n_i be the number of phases of state i. The linear program is then the following:

$$(P) \qquad \min \qquad \sum_{1=1}^{N} \sum_{p=1}^{n_i} d(i)x(i,p)$$

$$\text{For any time } t\text{:} \sum_{i=1}^{N} \sum_{p \mid t \in [t_{p-1}(i), t_p(i)]} x(i,p) \geq 1 \tag{7.1}$$

It may seem that the linear program contains an unbounded number of constraints. However, it is easy to see that we need only to consider times t which are the end of a phase for some state. It can also be easily verified that given an instance of the MTS problem, any feasible solution in the new MTS model defines a feasible solution to (P) with the same cost. We also observe that a feasible solution to (P) defines a (fractional) solution which is feasible in the new MTS model with the same cost. We should be a bit more careful in the online case, where the constraints of (P) are revealed one-by-one. Upon arrival of a constraint, the algorithm finds a feasible assignment to the (primal) variables that satisfies the constraint. Consider variable $x(i,p)$. In the offline case, we can assume without loss of generality that the value of $x(i,p)$ is determined at the beginning of phase p of state i. However, this is not necessarily true in the online case; thus, we restrict our attention to solutions that assign values to $x(i,p)$ forming a monotonically non-decreasing sequence.

The above formulation of the problem is now a covering linear program and thus it fits the online primal-dual framework. We now can apply the algorithms from Section 4.2 to derive a monotonically increasing fractional solution. The algorithm produces an $O(\log N)$-competitive solution since the number of variables in each covering constraint is exactly N.

Rounding the fractional solution. Rounding the fractional solution for this problem is simple. The algorithm maintains the invariant that it is in state i at time t (in phase p) with probability equal to $x(i,p)$. Suppose that at the end of phase p of state i, $x(i,p) = a$. The distribution mass a is then distributed among the states of the system (including i) by the fractional solution. Let a_j the increase of the fraction associated with state j at that point of time. As $\sum_j a_j = a$, if the algorithm was in state i at the end on phase p, it moves to state j with probability a_j/a. It is easy to verify that the expected cost of the algorithm is exactly the cost of the fractional solution.

7.3 Notes

The results in this chapter are based on the work of Bansal, Buchbinder and Naor [14]. The Metrical Task System (MTS) problem has been studied extensively. The MTS model was originally formulated by Borodin, Linial and Saks [29] who gave tight upper and lower bound of $2N-1$ for any deterministic online algorithm for the problem. They also designed a $2H_N$-competitive randomized algorithm for the uniform metric, and showed a lower bound of H_N for this metric. In fact, our proposed algorithm for the weighed star can be seen in retrospect as a direct generalization of their approach. For the MTS problem on a weighted star, Blum et al. [25] gave a randomized $O(\log^2 N)$-competitive algorithm. For general metrics Bartal et al. [19] designed a randomized $O(\log^5 N)$-competitive algorithm which is based on an algorithm for HST's. Fiat and Mendel [48] improved this bound and designed an $O(\log^2 N \log\log N)$-competitive algorithm for general metrics.

Chapter 8

Generalized Caching

Caching is one of the earliest and most effective techniques of accelerating the performance of computing systems. Thus, vast amounts of work have been invested in the improvement and refinement of caching techniques and algorithms. In the classic two-level caching problem we are given a collection of n pages and a fast memory (cache) which can hold up to k of these pages. At each time step one of the pages is requested. If the requested page is already in the cache then no cost is incurred, otherwise the algorithm must bring the page into the cache, possibly evicting some other page, and a cost of one unit is incurred. This simple model can be extended in two orthogonal directions. First, the cost of bringing a page into the cache may not be uniform for all pages. This version of the problem is called *weighted caching* and it models scenarios in which the cost of fetching a page is not the same due to different locations of the pages (e.g., main memory, disk, Internet). Second, the size of the pages need not be uniform. This is motivated by web caching where pages have varying sizes. Web caching is an extremely useful technique for enhancing the performance of World Wide Web applications. Since fetching a web page or any other information from the internet is usually costly, it is common practice to keep some of the pages closer to the client. This is done, for example, by the web browser itself by keeping some of the pages locally, and also by internet providers that maintain proxy servers for exactly the same purpose.

We study here several models in which pages have non-uniform sizes. The most general setting is called the *General model* in which pages have both non-uniform sizes and non-uniform fetching costs. Two commonly studied special cases are the so-called *Bit model* and *Fault model*. In the Bit model, the cost of fetching a page is proportional to its size, thus minimizing the fetching cost corresponds to minimizing the total traffic in the network. In the Fault model, the fetching cost is uniform for all pages, thus corresponding to the number of times a user has to wait for a page to be retrieved.

In Section 8.1 and Section 8.2 we study the weighted caching problem. Later in Section 8.3 and Section 8.4 we extend the ideas used for the weighted caching problem to the more general problem where pages have both non-uniform sizes and non-uniform fetching costs.

8.1 The Fractional Weighed Caching Problem

In this section we study the weighted caching problem. In the weighted caching problem each page p is associated with a positive fetching cost $c_p \geq 1$, denoting the cost of fetching the page into the cache. A request sequence is a sequence of pages, denoted by $p_1, p_2, \ldots$, where page p_t is requested at time t. The tth request is served by placing page p_t in the cache at time t, for each $t \geq 1$. The objective is to minimize the total cost of fetching pages into the cache.

8.1.1 Linear Programming Relaxation

Consider the following (natural) integer program for the (offline) weighted caching problem. Instead of charging for fetching pages into the cache we charge for evicting them, thus increasing the cost of any algorithm by at most an additive term (fetching the last k pages is "free"). Let $x(p, j)$ be an indicator variable for the event that page p is evicted from the cache between the jth time it is requested and the $(j+1)$st time it is requested. If $x(p, j) = 1$, we can assume without loss of generality that page p is evicted in the first time slot following the jth time it is requested. (As we later discuss, this assumption is not necessarily true in the online case.) For each page p, denote by $t(p, j)$ the time it is requested for the jth time, and denote by $r(p, t)$ the number of times page p is requested until time t (including t). For any time t, let $B(t) = \{p \mid r(p, t) \geq 1\}$ denote the set of pages that were requested until time t (including t). Let p_t be the page that was requested at time t. We need to satisfy the constraint that at any time t, the currently requested page must be present in the cache, i.e. $x(p_t, r(p, t)) = 0$, and that the total space used by pages in $B(t)$ can be at most k. Since one unit of space is already used by p_t, this implies that at most $k-1$ space can be used by pages in $B(t) \setminus \{p_t\}$. Equivalently, pages in $B(t) \setminus \{p_t\}$ with cumulative size at least $|B(t)| - 1 - (k-1) = |B(t)| - k$ must be absent from the cache at time t. This gives the following exact formulation of the problem.

$$\min \quad \sum_{p=1}^{n} \sum_{j=1}^{r(p,t)} c_p \cdot x(p, j)$$

$$\begin{array}{lr} \text{For any time } t\text{:} & \sum_{p \in B(t) \setminus \{p_t\}} x(p, r(p, t)) \geq |B(t)| - k \\ \text{For any } p, t\text{:} & x(p, t) \in \{0, 1\} \end{array}$$

In a fractional solution, we relax $x(p, j)$ to take any value between 0 and 1, and so we get the constraint that for each page p and time t, $0 \leq x(p, t) \leq 1$. In the dual program, there is a variable $y(t)$ for each time t and a variable $z(p, j)$ for each page p and the jth time it is requested. The dual program is the following:

$$\max \quad \sum_{t} (|B(t)| - k)\, y(t) \; - \; \sum_{p=1}^{n} \sum_{j=1}^{r(p,t)} z(p, j)$$

For each page p and the jth time it is requested:

$$\left(\sum_{t=t(p,j)+1}^{t(p,j+1)-1} y(t)\right) - z(p,j) \leq c_p \tag{8.1}$$

$$\text{For any } p,j: \qquad z(p,j) \geq 0$$

$$\text{For all } t: \qquad y(t) \geq 0$$

8.1.2 A Fractional Primal-Dual Algorithm

Our online caching algorithm produces fractional primal and dual solutions to the linear formulation. In the online case, the constraints of LP (corresponding to the requests to pages) are revealed one-by-one. Upon arrival of a constraint, the algorithm finds a feasible assignment to the (primal) variables that satisfies the constraint. Consider variable $x(p,j)$. In the offline case, we can assume without loss of generality that the value of $x(p,j)$ is determined at time $t(p,j)+1$. However, this is not necessarily true in the online case; thus, we stipulate that the values assigned to $x(p,j)$ in the time interval $[t(p,j)+1, t(p,j+1)-1]$ by the online algorithm form a monotonically non-decreasing sequence.

We start with a high level description of the algorithm. Upon arrival of a new constraint at time t, if it is already satisfied, then the algorithm does nothing. Otherwise, the algorithm needs to satisfy the current constraint by increasing some of the primal variables in the constraint. Satisfying the constraint guarantees that there is a enough space in the cache to fetch the new page. To this end, the algorithm starts increasing (continuously) the new dual variable $y(t)$. This, in turn, tightens some of the dual constraints corresponding to primal variables $x(p,j)$ whose current value is 0. Whenever such an event happens, the value of $x(p,j)$ is increased from its initial setting of 0 to $1/k$. Thus, during the time preceding the increase of $x(p,j)$ from 0 to $1/k$, page i cannot be evicted from the cache. This part is somewhat similar to what happens in the Randomized Marking algorithm [47]. Meanwhile, variables $x(p,j)$ which are already set to $1/k$ are increased (continuously) according to an exponential function of the new dual variable $y(t)$. Note that this exponential function is equal to $1/k$ when the constraint is tight. Thus, the algorithm is well defined. When variable $x(p,j)$ reaches 1, the algorithm starts increasing the dual variable $z(p,j)$ at the same rate as $y(t)$. As a result, from this time on, the value of $x(p,j)$ remains 1. The algorithm is presented in a continuous fashion, but it can easily be implemented in a discrete fashion. The algorithm is the following:

Fractional Caching Algorithm: At time t, when page p_t is requested:

- Set the new variable: $x(p_t, r(p_t, t)) \leftarrow 0$. (It can only be increased in times $t' > t$.)
- If the primal constraint corresponding to time t is satisfied, then do nothing.
- Otherwise: increase primal and dual variables, until the primal constraint corresponding to time t is satisfied, as follows:

  1. Increase variable $y(t)$ continuously; for each variable $x(p,j)$ that appears in the (yet unsatisfied) primal constraint that corresponds to time t:
  2. If $x(p,j) = 1$, then increase $z(p,j)$ at the same rate as $y(t)$.
  3. If $x(p,j) = 0$ and

  $$\left(\sum_{t=t(p,j)+1}^{t(p,j+1)-1} y(t)\right) - z(p,j) = c_p,$$

  then set $x(p,j) \leftarrow 1/k$.
  4. If $1/k \leq x(p,j) < 1$, increase $x(p,j)$ according to the following function:

  $$\frac{1}{k} \cdot \exp\left(\frac{1}{c_p}\left[\left(\sum_{t=t(p,j)+1}^{t(p,j+1)-1} y(t)\right) - z(p,j) - c_p\right]\right)$$

Note that the exponential function for $x(p,j)$ contains variables $y(t)$ that correspond to future times. However, these variables are all initialized to 0, so they do not contribute to the value of the function. We also remark that setting the variable $x(p,j)$ in line 3 to value of 1 instead of $\frac{1}{k}$, and removing Line 4 of the algorithm, we get a deterministic k-competitive algorithm for the weighted caching problem. This algorithm is exactly Young's dual-greedy algorithm [90, 42].

The analysis of the primal cost is partitioned into two parts. The first one corresponds to the contribution of the increase of the variables $x(p,j)$ from 0 to $1/k$, and the second part corresponds to the increase of the variables $x(p,j)$ from $1/k$ till (at most) 1, according to the exponential function. Each part is upper bounded separately by the dual solution, yielding the desired result. We now prove the following theorem.

Theorem 8.1. *The algorithm is $O(\log k)$-competitive. Specifically, the algorithm is $2(1 + \ln k)$-competitive.*

Remark 8.2. *It is possible to improve the competitive ratio of the algorithm from $2(1 + \ln k)$ to approximately $\ln k$. To do so simply replace the parameter $\frac{1}{k}$ in Lines 3 and 4 in the algorithm with the value $\frac{1}{k \ln k}$. It is not hard to verify in the proof that this will result an algorithm with competitive ratio $\ln k$ plus lower order terms (e.g. $\ln \ln k$). In the* *rounding phase, however, we loose several constants so we do not make this optimization.*

Proof of Theorem 8.1. First, we note that the primal solution generated by the algorithm is feasible. This follows since, in each iteration, the variables $x(p,j)$ are increased until the new primal constraint is satisfied. Also, each variable $x(p,j)$ is never increased to be greater than 1.

Next, we show that the dual solution that we generate is almost feasible. Whenever $x(p,j)$ increases in some round and reaches 1, the algorithm starts increasing $z(p,j)$ at the same rate as $y(t)$. Therefore, the value of $x(p,j)$ is not going to change anymore, as the exponent of the exponential function will not change any more. Thus, for the dual constraint corresponding to page p and the jth time it is requested, we get that:

$$x(p,j) = \frac{1}{k}\exp\left(\frac{1}{c_p}\left[\left(\sum_{t=t(p,j)+1}^{t(p,j+1)-1} y(t)\right) - z(p,j) - c_p\right]\right) \leq 1.$$

Simplifying, we get that:

$$\left(\sum_{t=t(p,j)+1}^{t(p,j+1)-1} y(t)\right) - z(p,j) \leq c_p(1+\ln k). \tag{8.2}$$

Thus, the dual solution can be made feasible by scaling it down by a factor of $(1+\ln k)$. We now prove that the primal cost is at most twice the dual profit, which means that the primal solution produced by the algorithm is $2(1+\ln k)$-competitive.

We partition the primal cost into two parts. Let C_1 be the contribution to the primal cost from Step (3) of the algorithm, due to the increase of variables $x(p,j)$ from 0 to $1/k$. Let C_2 be the contribution to the primal cost from Step (4) of the algorithm, due to the incremental increases of the variable $x(p,j)$ according to the exponential function from $1/k$ up to at most 1.

Bounding C_1: Let $\tilde{x}(i,j) = \min(x(p,j), \frac{1}{k})$. We bound the term $\sum_{i=1}^{n}\sum_{j=1}^{r(p,t)} c_p\tilde{x}(i,j)$. To do this, we need several observations. First, from design of the algorithm, it follows that if $x(p,j) > 0$, and equivalently if $\tilde{x}(i,j) > 0$, then:

$$\left(\sum_{t=t(p,j)+1}^{t(p,j+1)-1} y(t)\right) - z(p,j) \geq c_p \tag{8.3}$$

We shall refer to (8.3) as primal complementary slackness.

Next, at time t let $B'(t)$ be the set of pages $p \in B(t)$ such that $x(p, r(p,t)) = 1$. In the dual solution, if $y(t)$ is being increased at time t then:

$$\sum_{p \in B(t)\setminus(B'(t)\cup\{p_t\})} \tilde{x}(p, r(p,t)) \leq |B(t)| - k - |B'(t)| \tag{8.4}$$

We shall refer to (8.4) as dual complementary slackness. Inequality (8.4) follows since there are $|B(t)| - 1 - |B'(t)|$ variables in the constraint corresponding to t. By definition for each p, $\tilde{x}(p, r(p,t)) \leq \frac{1}{k}$. Thus, even if for all p, $\tilde{x}(p, r(p,t)) = \frac{1}{k}$, then the sum adds up to $\frac{|B(t)|-1-|B'(t)|}{k} \leq |B(t)|-k-|B'(t)|$. The latter inequality holds since $|B(t)|-|B'(t)| \geq k+1$, since otherwise the constraint at time t is already satisfied and the algorithm stops increasing the variable $y(t)$. Also, it follows from the algorithm that if $z(p,j) > 0$, then:

$$x(p,j) \geq 1 \tag{8.5}$$

We shall refer to (8.5) as the second dual complementary slackness. These primal and dual complementary slackness conditions imply the following.

$$\begin{aligned}
&\sum_{p=1}^{n} \sum_{j=1}^{r(p,t)} c_p \tilde{x}(p,j) \\
&\quad \leq \sum_{p=1}^{n} \sum_{j=1}^{r(p,t)} \left(\left(\sum_{t=t(p,j)+1}^{t(p,j+1)-1} y(t) \right) - z(p,j) \right) \tilde{x}(p,j) && (8.6) \\
&\quad = \sum_{t} \left(\sum_{i \in B(t) \setminus \{p_t\}} \tilde{x}(p, r(p,t)) \right) y(t) - \sum_{p=1}^{n} \sum_{j=1}^{r(p,t)} \tilde{x}(p,j) z(p,j) && (8.7) \\
&\quad \leq \sum_{t} \left(|B(t)| - k \right) y(t) - \sum_{p=1}^{n} \sum_{j=1}^{r(p,t)} z(p,j). && (8.8)
\end{aligned}$$

Inequality (8.6) follows from Inequality (8.3), Equality (8.7) follows by changing the order of summation. To see why Inequality (8.8) holds consider some time t. Consider the derivative of the LHS at time t. By the algorithm behavior (Inequality (8.5)) we increase $z(p,j)$ at the same rate as $y(t)$ only when $x(p, r(p,t)) = 1$ and so $p \in B'(t)$. Thus, the derivative of the LHS is $\sum_{p \in B(t) \setminus (B'(t) \cup \{p_t\})} \tilde{x}(p, r(p,t))$. By Inequality (8.4) this sum is at most $|B(t)| - k - |B'(t)|$ which is exactly the derivative of the RHS of the Inequality. Thus, C_1 is at most the profit of a *feasible* dual solution multiplied by $(1 + \ln k)$.

Bounding C_2: We bound the derivative of the increase of variables $x(p,j)$ in Step (4) by the derivative of the dual profit accrued in the same round. In each round only variables $x(p,j)$ that belong to the new primal constraint (and correspond to the new dual variable $y(t)$) are being increased. However, variables $x(p,j)$ that belong to the new primal constraint but have already reached the value of 1 are not increased anymore and so do not contribute to the primal cost. In the dual program the new variable $y(t)$ is raised with rate 1, and also all the variables $z(p,j)$ that correspond to $x(p,j)$ in the new primal constraint that are already with value 1. It is beneficial for the purpose of analysis to think of the process as increasing a time variable τ, and then raising the variable $y(t)$ and the appropriate variables $z(p,j)$ with rate 1 with respect to the virtual variable τ. Using this notation we get that:

$$
\begin{aligned}
\frac{dC_2}{d\tau} &= \sum_{p\in B(t)\setminus\{p_t\},1/k\le x(p,j)<1} c_p \cdot \frac{dx(p,r(p,t))}{dy(t)} \cdot \frac{dy(t)}{d\tau} \\
&= \sum_{p\in B(t)\setminus\{p_t\},1/k\le x(p,j)<1} x(p,r(p,t)) \qquad (8.9) \\
&\le (|B(t)|-k) - \sum_{p\in B(t)\setminus\{p_t\},x(p,j)=1} 1 \qquad (8.10) \\
&= (|B(t)|-k)\,\frac{dy(t)}{d\tau} - \sum_{p\in B(t)\setminus\{p_t\},x(p,j)=1} \frac{dz(p,j)}{d\tau}
\end{aligned}
$$

Where Equality (8.9) follows since $\frac{dy(t)}{d\tau} = 1$ and also $\frac{dx(p,j)}{dy(t)} = \frac{1}{c_p} \cdot x(p,j)$ for each $1/k \le x(p,j) < 1$. Inequality (8.10) holds since the new primal constraint is unsatisfied yet and thus:

$$
\sum_{p\in B(t)\setminus\{p_t\},1/k\le x(p,j)<1} x(p,r(p,t)) + \sum_{p\in B(t)\setminus\{p_t\},x(p,j)=1} x(p,r(p,t)) < |B(t)| - k
$$

We also remark that by the properties of the algorithm, any variable $x(p,j)$ which is strictly less than $1/k$ is actually equal to 0. Finally, the last term exactly equals the derivative of the dual profit with respect to τ. Therefore, the change in the dual profit is greater than or equal to the change in C_2. Thus, C_2 is at most the profit of a *feasible* dual solution multiplied by $(1 + \ln k)$.

Completing the analysis. It follows that $C_1 + C_2$ is at most twice the profit of a *feasible* dual solution multiplied by $(1 + \ln k)$. Note that the profit of any dual feasible solution is always a lower bound on the optimal solution. Therefore, we conclude by weak duality that the algorithm is $2(1 + \ln k)$-competitive. □

8.1.3 A Fractional Algorithm for the Weighted (h, k)-caching Problem

A common approach to proving better performance of an online caching algorithm is the (h, k)-caching problem where the online algorithm with cache size k is compared to the offline algorithm with cache size h. In this section we show a simple modification of the algorithm that improves the competitive ratio for this case.

The modified algorithm generates a primal solution that is suitable to the online algorithm with cache size k. However, the algorithm generates a dual solution that corresponds to a primal solution that may only use cache size $h \le k$. We will perform a primal-dual analysis and show that primal cost is no more than $O(\log(k/(k - h + 1)))$ times the dual cost $\sum_t (|B(t)| - h)\, y(t) \;-\; \sum_{p=1}^{n} \sum_{j=1}^{r(p,t)} z(p,j)$. Since the dual cost is a lower bound on the offline cost with cache size h, this will imply the desired result. For convenience, let η denote $(k - h + 1)/k$. To avoid trivialities, we assume that $k \ge h$, and that $h > 1$, which implies that $\frac{1}{k} \leq \eta < 1$. Intuitively, η replaces the value $1/k$ in our algorithm. Consider the following modified algorithm:

Fractional Caching Algorithm: At time t, when page p_t is requested:

- Set the new variable: $x(p_t, r(p_t, t)) \leftarrow 0$. (It can only be increased in times $t' > t$.)
- If the primal constraint corresponding to time t is satisfied, then do nothing.
- Otherwise: increase primal and dual variables, until the primal constraint corresponding to time t is satisfied, as follows:
  1. Increase variable $y(t)$ continuously; for each variable $x(p, j)$ that appears in the (yet unsatisfied) primal constraint that corresponds to time t:
  2. If $x(p, j) = 1$, then increase $z(p, j)$ at the same rate as $y(t)$.
  3. If $x(p, j) = 0$ and
  $$\left(\sum_{t=t(p,j)+1}^{t(p,j+1)-1} y(t) \right) - z(p, j) = c_p,$$
  then set $x(p, j) \leftarrow \eta$.
  4. If $\eta \leq x(p, j) < 1$, increase $x(p, j)$ according to the following function:
  $$\eta \cdot \exp\left(\frac{1}{c_p} \left[\left(\sum_{t=t(p,j)+1}^{t(p,j+1)-1} y(t) \right) - z(p, j) - c_p \right] \right)$$

By the description of the algorithm, $x(p, j) \leq 1$ implies that

$$\sum_{t=t(p,j)+1}^{t(p,j+1)-1} y(t) - z(p, j) \leq c_p(1 + \ln(1/\eta))$$

and hence $y(t)$ divided by $(1 + \ln(1/\eta))$ is a feasible dual solution (to the dual of the problem with only h pages). As previously, we relate the primal cost to the dual cost in two parts C_1 and C_2, where C_1 is contribution to the primal cost from Step (3) of the algorithm, due to the increase of variables $x(p, j)$ from 0 to η, and C_2 is the contribution to the primal cost from Step (4) of the algorithm, due to the incremental increases of the variable $x(p, j)$ according to the exponential function.

We first observe that the argument for C_2 follows exactly as for the case when $h = k$. In particular, Δ, the derivative of dual profit with respect to $y(t)$ is equal to $|B(t)| - h$ minus the number of variables $x(p, j)$ that have already reached 1. Moreover, we have that

$$\frac{dx(p, j)}{dy(t)} = \frac{1}{c_p} \cdot x(p, j)$$

and hence the change in primal cost is equal to the sum of $x(p, j)$ that are at least η and strictly less than 1. Since the primal constraint is still unsatisfied, the sum of $x(p, j)$ is at most $|B(t)| - k$ which is at most $|B(t)| - h$ (as $k \geq h$), and hence the sum of $x(p, j)$ that are strict less than 1 is at most Δ. Thus, C_2 is bounded by the dual cost.

Bounding C_1 is also very similar. The only change is in the dual complementary slackness condition. We set the primal variables to η instead of $\frac{1}{k}$. Again, at time t let $B'(t)$ be the set of pages $i \in B(t)$ such that $x(p, r(p, t)) = 1$. In the dual solution, if $y(t)$ is being increased at time t then:

$$\sum_{p \in B(t) \setminus (B'(t) \cup \{p_t\})} \tilde{x}(p, r(p, t)) \leq |B(t)| - h - |B'(t)| \tag{8.11}$$

Inequality (8.11) follows since there are $|B(t)| - 1 - |B'(t)|$ variables in the constraint corresponding to t. By definition for each i, $\tilde{x}(p, r(p, t)) \leq \eta = \frac{k-h+1}{k}$. Thus, even if for all i, $\tilde{x}(p, r(p, t)) = \frac{k-h+1}{k}$, then the sum adds up to $(|B(t)| - 1 - |B'(t)|) \cdot \frac{k-h+1}{k} \leq |B(t)| - h - |B'(t)|$. The latter inequality holds since $|B(t)| - |B'(t)| \geq k + 1$, since otherwise the constraint at time t is already satisfied and the algorithm stops increasing the variable $y(t)$.

Thus, repeating the same analysis as in Theorem 8.1 implies that C_1 is bounded by the dual cost. It follows that $C_1 + C_2$ is at most $2(1 + \ln(1/\eta)) = 2(1 + \ln(k/(k - h + 1)))$ times the optimum solution, which implies that the algorithm is $O(\log(k/(k - h + 1)))$-competitive.

8.2 Randomized Online Algorithm for Weighted Caching

A randomized algorithm is completely specified by a probability distribution on the various configurations (deterministic states) in each state of the algorithm. For the caching problem this corresponds to specifying the distribution on k-tuples of pages that are in the cache. Such a distribution induces another (simpler) distribution $x(p, t)$ on the pages, specifying the probability that a the page in in the cache at time t. Clearly, this map is not a bijection. For example, the distribution $(1/2, 1/2, 1/2, 1/2)$ on four pages A, B, C, D could be induced by the distribution D_1 on two states (A, B) and (C, D) where each state occurs with probability $1/2$ each, or it can be induced by the distribution D_2 where each of six possible states (A, B), $(A, C), \ldots, (C, D)$ occur with probability $1/6$ each.

For the caching problem, the distribution on the pages can be viewed as a probability mass of k units distributed among the n pages, and the "move" of an algorithm simply corresponds to redistributing this mass among the pages. In this view when the algorithm moves ϵ units of mass from page i to page j, it incurs a cost of $\epsilon \cdot (w(i) + w(j))$. We call this the *fractional view* in contrast to working with the probability distribution on states, that we call the *actual* view. We note that a fractional view can easily be obtained from a solution to linear program (LP-Caching), since the variables in the linear program indicate what fraction of a page is already evacuated from the cache. More formally, at time t the probability that page p is in the cache is simply $1 - x(p, r(p, t))$.

Our goal is to generate a randomized algorithm from a fractional view. The main problem in doing so is demonstrated in the following example. Consider the distribution $(1/2, 1/2, 1/2, 1/2)$ on pages A, B, C and D induced by the actual view where cache states (A, B) and (C, D) each occurring with probability 1/2. Pages A and B have weight 1, and pages C and D have a large weight M. Suppose the fractional algorithm moves 1/2 unit of mass from page A to page B leading to the state of $(0, 1, 1/2, 1/2)$. In the fractional view, this algorithm incurs a cost of 1/2. However, it is instructive to see that it is impossible to modify the actual distribution (to be consistent with the fractional distribution) without incurring a cost of $\Theta(M)$. In fact, the only actual distribution consistent with $(0, 1, 1/2, 1/2)$ is to have probability 1/2 on state (B, C) and probability 1/2 on state (B, D). Thus, from the previous cache state (C, D), either C or D must be moved to make room for B, which incurs cost $\Theta(M)$.

To get around this problem we restrict our actual distributions to a certain subclass of distributions (for example in the scenario described above, we do not allow the distribution where states (A, B) and (C, D) have probability half each to correspond to the distribution $(1/2, 1/2, 1/2, 1, 2)$). In particular, we show below how to maintain an online mapping from induced distributions to actual distributions, such that any fractional move with cost c is mapped to a move on actual distributions with cost at most $5c$.

We first round up the page fetching costs to their nearest power of 2 (increasing the competitive ratio by at most a factor of 2). Let $c_1 < c_2 < \ldots < c_\ell$ denote the rounded weights. A page belongs to class i if its rounded weight is c_i. We refer to an individual page as the j-th page of class i. For convenience of analysis, throughout this section we consider the (equivalent) cost version of the problem where we pay $c_i/2$ for both fetching and evicting a class i page.

Recall that in the fractional view of the problem, the algorithm maintains a distribution on the pages with total mass k. Any such distribution P is completely specified by $p_{ij} \in [0, 1]$ such that $\sum_i \sum_j p_{ij} = k$, where p_{ij} is the mass on the j-th page of weight class i. Given two distributions P and P' on pages, let $C_f(P, P')$ denote the cheapest way to move from P to P', where it costs $c_i/2$ to move one unit of mass either into or out of a class i page. For those familiar, C_f is just the transshipment cost of flow from P to P' (we refer the reader to [37] for details about transshipment cost between distributions). Let $\delta_{ij} = p_{ij} - p'_{ij}$. Clearly, $C_f(P, P')$ is at least $\sum_i (c_i/2)(\sum_j |\delta_{ij}|)$ since at least $|\delta_{ij}|$ units of mass either needs to enter or leave page j of class i. Moreover, any greedy algorithm that arbitrarily moves mass out of pages with excess ($\delta_{ij} > 0$) to those with a deficiency ($\delta_{ij} < 0$) has cost $\sum_i (c_i/2)(\sum_j |\delta_{ij}|)$ implying that $C_f(P, P') = \sum_i (c_i/2)(\sum_j |\delta_{ij}|)$.

A randomized algorithm on the other hand needs to work with a distribution on valid cache states. Given two distributions D and D' on the cache states, let $C(D, D')$ denote the cheapest way of moving from D to D' (by definition, this is the cost incurred by the randomized algorithm). Let $\Pi(D)$ denote the distribution induced on the pages by D. We say that P and D are *consistent* if $P = \Pi(D)$. Clearly, $C_f(\Pi(D), \Pi(D')$ is a lower bound on $C(D, D')$.

For the unweighted caching problem, Blum, Burch and Kalai [24] showed that given any P, P' and D such that $\Pi(D) = P$, there exists some D' such that $\Pi(D') = P'$

and $C(D, D') \leq 2C_f(P, P')$. Their procedure is the following. Suppose without loss of generality that P' is obtained from P by removing ϵ units of mass from page a and putting it on page b. Remove page a arbitrarily from ϵ measure of caches that contain a, and add page b to ϵ measure of caches that do not contain b. Now, some caches may have $k+1$ pages (an excess) while some may have $k-1$ pages (a hole). Arbitrarily match the caches with an excess to those with a hole (clearly, the measure of caches with excess is equal to those with a hole). Consider any matched pair; the cache with an excess must contain a page that does not lie in its matched cache, so we simply transfer this page. It can easily be verified that $C(D, D') \leq 2\epsilon$, while the fractional cost $C_f(P, P') = \epsilon$.

However the situation for weighted caching is more involved. Recall our example that shows that there exist P, P' and D consistent with P, such that $C(D, D') \gg C_f(P, P')$ for every D' satisfying $P' = \Pi(D')$. Thus, we cannot work with any arbitrary D that is consistent with P, as in the unweighted case. Interestingly, we get around this problem by carefully restricting the space of distributions D that we are allowed to work with. Formally, we show the following.

Theorem 8.3. *Let the costs c_i be such that $c_{i+1}/c_i \geq 2$ for $1 \leq i \leq \ell - 1$. There is a subclass $\mathcal{D}$ of distributions on cache states, along with a map T from $(\mathcal{D} \times \mathcal{P}) \to \mathcal{D}$ with the following property: Given any two distributions on pages P and P', and given any $D \in \mathcal{D}$ satisfying $\Pi(D) = P$, we can obtain another distribution $D' = T(D, P')$ such that $\Pi(D') = P'$, $D' \in \mathcal{D}$ and $C(D, D') \leq 5C_f(P, P')$.*

The theorem gives us the desired mapping between a distribution P on pages and a distribution D on cache states. Whenever the fractional algorithm moves from state P to P', the randomized algorithm moves from D to $D' = T(D, P')$. Since $\Pi(D') = P'$ and $D' \in \mathcal{D}$, the process can be applied repeatedly.

Proof. Let P be a distribution on pages with total mass k. Let $\mathcal{D}(P)$ denote the set of distributions $D \in \mathcal{D}$ that are consistent with P. Specifying $\mathcal{D}(P)$ for each P suffices to describe $\mathcal{D}$ completely. Each distribution $D \in \mathcal{D}$ is specified by associating a cache state $C(\alpha)$ with each real number α in the interval $[0, 1)$.

Let $k_i = \sum_j p_{ij}$ denote the mass on class i pages as determined by P. Consider the interval $I = [0, k]$, and imagine this interval partitioned into $I_1, \ldots, I_l$ where $I_1 = [0, k_1)$, $I_2 = [k_1, k_1 + k_2)$,$\ldots$, $I_l = [k_1 + \ldots, k_{l-1}, k_1 + \ldots + k_l)$. Consider an $\alpha \in [0, 1)$. Let $T(\alpha)$ denote the set of real numbers $\{\alpha, 1 + \alpha, 2 + \alpha, \ldots, k - 1 + \alpha\}$. For every $D \in \mathcal{D}(P)$, the cache $C(\alpha)$ has n_i pages of c_i where $n_i = |T(\alpha) \cap I_i|$. By construction, each cache $C(\alpha)$ has either $\lfloor k_i \rfloor$ or $\lceil k_i \rceil$ pages of fetching cost c_i, and the expected number of pages of cost c_i is k_i. Consider any arbitrary way of filling the caches $C(\alpha)$, for $0 \leq \alpha < 1$, with pages such that: (i) no $C(\alpha)$ contains two identical pages and (ii) it is consistent with P (i.e. the probability measure of caches that contain page j of class i is exactly p_{ij}). Such a filling always exists since, for example, we can put the first page of class 1 in $C(\alpha)$ corresponding to $\alpha = [0, p_{11})$, the second page of class 1 in $C(\alpha)$ corresponding to $\alpha = [p_{11}, p_{11} + p_{12})$ (where the range of α is considered modulo 1) and so on. Any way of filling $C(\alpha)$'s that satisfies the properties above is a valid element $D \in \mathcal{D}(P)$.

We now describe the transformation T. Suppose we are given some $D \in \mathcal{D}(P)$, and the fractional algorithm changes state from P to P'. By separating the pages for which $p'_{ij} > p_{ij}$ and those for which $p'_{ij} < p_{ij}$ and arbitrarily matching the increases in mass with decreases, we can decompose the move P to P' into the sequence $P = P_0 \to P_1 \to P_2 \to \ldots \to P'$ such that $C_f(P, P') = \sum_{i \geq 0} C_f(P_i, P_{i+1})$ and each move P_i to P_{i+1} is an *exchange* where some infinitesimally small ϵ units of mass is moved from some page p_a to some page p_b. Thus, it suffices to prove the theorem for such exchanges $P \to P'$. Let i be the weight class of page a, and j be that of page b. For this move, the fractional algorithm pays $\epsilon(c_i + c_j)/2$.

We now describe and analyze the move $D \to D'$. We divide the cost into two parts. One due to cache size changes, and the second due to the change in the composition of the cache. We first consider the simpler case when $i = j$. Here, the quantities $k_1, \ldots, k_\ell$ and the intervals $I_1, \ldots, I_\ell$ associated with P remain unchanged, and hence the structure of $C(\alpha)$'s remains unchanged. We essentially apply the argument of Blum et al. [24] to class i pages. The only difference is that we need to verify that their argument works even when caches contain either $\lceil k_i \rceil$ or $\lfloor k_i \rfloor$ class i pages (in [24] all caches have the same size). We arbitrarily remove page a from an ϵ measure of caches that contain a, and arbitrarily add b to an ϵ measure of caches that do not contain b. We say that a cache has a hole if it has one fewer page than it is supposed to, and it has an excess if it has one extra page than it is supposed to. Any cache with a hole has size either $\lfloor k_i \rfloor - 1$ or $\lfloor k_i \rfloor$, and every cache with excess has size either $\lceil k_i \rceil$ or $\lceil k_i \rceil + 1$, and hence is strictly larger. We arbitrarily pair up the caches with a hole to those with excesses, and transfer some page from the larger cache that does not lie in the smaller cache. The cost incurred is at most $2\epsilon c_i/2 + 2\epsilon c_i/2 = 2\epsilon c_i$.

We now consider the case when $i < j$ (the case when $i > j$ is analogous). Consider the intervals $I_1, \ldots, I_\ell$. When we move from P to P' the right boundary of I_i shifts ϵ units to the left, the intervals $I_{i+1}, \ldots, I_{j-1}$ shift to the left by ϵ units, and finally, the left boundary of I_j shifts left by ϵ and its right boundary stays fixed.

We break the analysis into two parts. We first consider the classes h for $i < h < j$. For each such h at most ϵ fraction of caches $C(\alpha)$ must lose a page of cost c_h (as their quota for class h shrinks from $\lceil k_h \rceil$ to $\lfloor k_h \rfloor$ and similarly, at most ϵ fraction of caches must gain a page. Moreover, the fraction of caches that must lose a cost c_h page is exactly equal to the fraction that must gain such a page. We arbitrarily pair these caches. As any cache that must lose a page is strictly larger than a cache that must gain one, for every matched pair of caches, there is some page in the larger cache that does not lie in the smaller cache and hence can be transferred to it. The movement cost incurred per class is at most $2\epsilon(c_h/2)$, and hence the total contribution due to such classes h is $\sum_{i<h<j} \epsilon c_h \leq \epsilon(c_i + c_j)$, as consecutive weights differ by a factor of at least 2.

Finally, we consider the case when $h = i$ (the argument for $h = j$ is analogous). Without loss of generality we assume that $\lceil k_i \rceil = \lceil k_i - \epsilon \rceil$ (otherwise we can split ϵ into at most 2 parts ϵ_1, ϵ_2, and apply the argument separately). Consider the caches $C(\alpha)$ that are supposed to lose a cost c_i page (because k_i becomes $k_i - \epsilon$). We say that these caches have an excess, and note that they all contain exactly $\lceil k_i \rceil$ class i pages. Next, we arbitrarily choose ϵ measure of caches that contain a, and remove a from them. These caches have a hole, and strictly fewer class i pages than caches with excess (a cache with a hole has either $\lfloor k_i \rfloor$ or $\lfloor k_i \rfloor - 1$ pages). We arbitrarily pair caches with an excess to caches with a hole, and transfer some page from the larger cache that does not lie in the smaller cache. The cost incurred is at most $3\epsilon c_i/2$. By an identical argument for class j, the cost incurred is at most $3\epsilon c_j/2$.

The distribution D' obtained satisfies all the conditions required to lie in the set $\mathcal{D}(P')$. Moreover, the total cost incurred in moving from D to D' is $5\epsilon(c_i + c_j)/2$ which is at most 5 times the fractional cost. $\square$

8.3 The Generalized Caching Problem

In this section we extend the main ideas and design an algorithm for the generalized caching problem. In the general caching problem there is a cache of size k and n pages of sizes $w_1 \leq w_2 \leq \ldots \leq w_n$, belonging to $\in [1, k]$. It is not assumed that page sizes are integral and k can be viewed as the ratio between cache size and the smallest page size. For any subset S of pages, let $W(S) = \sum_{p \in S} w_p$ be the sum of the sizes of the pages in S. Page p has a fetching cost of c_p. With this terminology, in the Fault model $c_p = 1$ for each page p, in the Bit model $c_p = w_p$ for each page p, and in the general model c_p and w_p are arbitrary.

8.3.1 LP formulation for general caching

The formulation that we used for the weighted caching can easily be extended to the case of generalized caching. This obvious extension gives us the following integer formulation for the problem.

$$\min \quad \sum_{p=1}^{n} \sum_{j=1}^{r(p,t)} c_p \cdot x(p,j)$$

$$\begin{array}{rc} \text{For any time } t\text{:} & \sum_{p \in B(t) \setminus \{p_t\}} w_p x(p, r(p,t)) \geq W(B(t)) - k \\ \text{For any } p, j\text{:} & x(p,j) \in \{0,1\} \end{array}$$

In a fractional solution, we relax $x(p,j)$ to take any value between 0 and 1. However, there is a fundamental problem with this relaxation. The problem is that this relaxation can have an integrality gap of $\Omega(k)$, and therefore is not suitable for our purposes. For example, suppose the cache size is $k = 2\ell - 1$, and there are two pages of size ℓ, requested alternately. Only one page can be in the cache at any time and hence there is a cache miss in each request. A fractional solution on the other hand can keep almost one unit of each page and then it only needs to fetch an $O(1/k)$ fraction of a page in each request.

To get around this problem, we use an idea introduced by Carr et al. [35] of adding exponentially many *knapsack cover* inequalities. These constraints are redundant in the integer program, but they dramatically reduce the integrality gap of the LP relaxation.

There are two main ideas. First, consider a subset of pages $S \subset B(t)$ such that $p_t \in S$ and $W(S) > k$. The pages in $S \setminus \{p_t\}$ can occupy at most $k - w_{p_t}$ size in the cache at time t. Thus, at least $W(S) - w_{p_t} - (k - w_{p_t}) = W(S) - k$ cumulative size of pages in $S \setminus \{p_t\}$ must be absent from the cache. Hence, we can add the constraint $\sum_{p \in S \setminus \{p_t\}} w_p x(p, r(p,t)) \geq W(S) - k$ for each such set S at time t. The second idea is that for each such constraint, we can truncate the size of a page to be equal to the right hand size of the constraint, i.e. we have $\sum_{p \in S \setminus \{p_t\}} \min(W(S) - k, w_p) x(p, r(p,t)) \geq W(S) - k$. Clearly, truncating the size has no effect on the integer program. Our LP is as follows.

$$\min \quad \sum_{p=1}^{n} \sum_{j=1}^{r(p,t)} c_p \cdot x(p,j)$$

For any time t and any set of requested pages $S \subseteq B(t)$ such that $p_t \in S$ and $W(S) > k$:

$$\sum_{p \in S \setminus \{p_t\}} \min\{W(S) - k, w_p\} x(p, r(p,t)) \geq W(S) - k \tag{8.12}$$

$$\text{For any } p, j: \qquad 0 \leq x(p,j) \leq 1 \tag{8.13}$$

We now note a simple observation about knapsack cover inequalities that will be quite useful.

Observation 8.4. *Given a fractional solution x, if a knapsack cover inequality is violated for a set S at time t, then it is also violated for the set $S' = S \setminus \{p : x(p, r(p,t)) = 1\}$, obtained by omitting pages which are already completely evicted from the cache.*

Proof. Suppose Inequality (8.12) is violated for some S and $x(p, r(p,t)) = 1$ for $p \in S$. First, it must be the case that $\min(W(S) - k, w_p) < W(S) - k$, otherwise (8.12) is trivially satisfied. Suppose we delete p from S. The right hand size decreases by exactly w_p. The left hand side decreases by w_p and possibly more since the term $\min(W(S) - k, w_{p'})$ may decrease for pages $p' \in S$. Thus, Inequality (8.12) is also violated for $S \setminus \{p\}$. The result follows by applying the argument repeatedly. $\square$

Observation 8.4 implies that in any feasible solution to the constraints given by (8.12), it does not help to have $x(p,j) > 1$. Hence, it can be assumed that $x(p,j) \leq 1$ without loss of generality. Thus, we can drop the upper bounds on $x(p,j)$ and simplify the LP formulation to:

$$\min \quad \sum_{p=1}^{n} \sum_{j=1}^{r(p,t)} c_p \cdot x(p,j) \qquad \text{(LP-Caching)}$$

For any time t and any set of requested pages $S \subseteq B(t)$ such that $p_t \in S$ and $W(S) > k$:

$$\sum_{p \in S \setminus \{p_t\}} \min\{W(S) - k, w_p\} x(p, r(p,t)) \geq W(S) - k \tag{8.14}$$

$$\text{For any } p, j\text{: } 0 \leq x(p,j) \tag{8.15}$$

In the dual program, there is a variable $y(t, S)$ for each time t and set $S \subseteq B(t)$ such that $p_t \in S$ and $W(S) > k$. The dual program is the following:

$$\max \quad \sum_t \sum_{S \subseteq B(t), p_t \in S} (W(S) - k)\, y(t, S)$$

For each page p and the jth time it is requested:

$$\sum_{t=t(p,j)+1}^{t(p,j+1)-1} \sum_{S \mid p \in S} \min\{W(S) - k, w_p\} y(t, S) \leq c_p \tag{8.16}$$

We will sometimes denote $\min\{W(S) - k, w_p\}$ by $\tilde{w}_p^S$.

8.3.2 Computing a Competitive Fractional Solution

Our online caching algorithm that produces fractional primal and dual solutions to LP-Caching is a lot similar to the weighted caching case. In the online case, the constraints of LP-Caching are revealed one-by-one. At any time t, exponentially many new linear knapsack-cover constraints are revealed to the online algorithm. The goal is to produce a feasible assignment to the (primal) variables that satisfies all the constraints. Since there are exponentially many constraints, this process may not run in polynomial-time. However, we show later that for our purposes we can make the algorithm run in polynomial time.

Upon arrival of the new constraints at time t, if all constraints are already satisfied, then the algorithm does nothing. Otherwise, the algorithm needs to satisfy all the current constraints by increasing some of the primal variables. We call a set S minimal if $x(p, r(p, t)) < 1$ for each $p \in S$. By Observation 8.4, it suffices to consider primal constraints corresponding to minimal sets. Satisfying all the constraints at time t guarantees that there is enough space (fractionally) in the cache to fetch the new page.

To this end, the algorithm arbitrarily picks an unsatisfied primal constraint corresponding to some minimal set S and starts increasing continuously its corresponding dual variable $y(t, S)$. This, in turn, tightens some of the dual constraints corresponding to primal variables $x(p, j)$ whose current value is 0. Whenever such an event happens, the value of $x(p, j)$ is increased from its initial setting of 0 to $1/k$. Meanwhile, variables $x(p, j)$ which are already set to $1/k$ are increased (continuously) according to an exponential function of the new dual variable $y(t, S)$. When variable $x(p, j)$ reaches 1, the set S is no longer minimal, and page p is dropped from S. As a result, from this time on, the value of $x(p, j)$ remains 1. When this primal constraint is satisfied the algorithm continues on to the next infeasible primal constraint.

Since there are exponentially many primal constraints in each iteration this process may not be polynomial. However, the rounding process we design in Section 8.4 does not need the solution to satisfy all primal constraints. Specifically, for each model we show that there exists a (different) value $\gamma > 1$ such that the algorithm needs to guarantee that at time t the primal constraint of the set $S = \{p \mid x(p, r(p, t)) < \frac{1}{\gamma}\}$ is satisfied. Thus, the algorithm may actually consider only that set[1]. Fortunately, the requirement of the online primal-dual framework that variables can only increase monotonically makes this task simple. In particular, as the primal variables increase some pages reach $1/\gamma$ and "leave" the set S. The algorithm then tries to satisfy the set S' that contains the rest of the pages. Since pages can only leave S, this process may continue for at most n rounds. For simplicity, we describe the algorithm that satisfies all the constraints. The algorithm is presented in a continuous fashion, but it can easily be implemented in a discrete fashion. The algorithm is the following:

Fractional Caching Algorithm: At time t, when page p_t is requested:

- Set the new variable: $x(p_t, r(p_t, t)) \leftarrow 0$. (It can only be increased at times $t' > t$.)
- Until **all** the primal constraint corresponding to time t are satisfied do the following:
- Assume that the primal constraint of a minimal set S is not satisfied.

  1. Increase variable $y(t, S)$ continuously; for each variable $x(p, j)$ such that $p \in S \setminus \{p_t\}$:
  2. If $x(p, j) = 1$, then remove p from S, i.e. $S \leftarrow S \setminus \{p\}$.
  3. If $x(p, j) = 0$ and $\displaystyle\sum_{t=t(p,j)+1}^{t(p,j+1)-1} \sum_{S:p\in S} \tilde{w}_p^S y(t, S) = c_p$, then $x(p, j) \leftarrow 1/k$.
  4. If $1/k \leq x(p, j) < 1$, increase $x(p, j)$ according to the following function:

  $$\frac{1}{k} \cdot \exp\left(\frac{1}{c_p}\left[\left(\sum_{t=t(p,j)+1}^{t(p,j+1)-1} \sum_{S:p\in S} \tilde{w}_p^S y(t, S)\right) - c_p\right]\right)$$

  where $\tilde{w}_p^S$ denotes $\min\{W(S) - k, w_p\}$.

Theorem 8.5. *The algorithm is $O(\log k)$-competitive.*

Proof. The proof of the theorem is in the same lines of the proof of Theorem 8.1. First, we note that the primal solution generated by the algorithm is feasible. This follows since, in each iteration, the variables $x(p, j)$ are increased until all new primal constraints are satisfied. Also, each variable $x(p, j)$ is never increased to be greater than 1.

Next, we show that the dual solution that we generate is feasible up to an $O(\log k)$ factor. Whenever $x(p, j)$ reaches 1, the variables $y(t, S)$ for sets S containing p do not increase anymore, and hence the value of $x(p, j)$ does not change any more. Thus, for the dual constraint corresponding to page p and the jth time it is requested, we get that:

[1] In general, for knapsack cover constraints in an offline setting, all possible subsets may be needed since it is not clear apriori which set S will have this property, nor can it be expressed as a linear or even a convex program. See [35] for more details.

$$x(p,j) = \frac{1}{k} \cdot \exp\left(\frac{1}{c_p}\left[\left(\sum_{t=t(p,j)+1}^{t(p,j+1)-1} \sum_{S:p\in S} \tilde{w}_p^S y(t,S)\right) - c_p\right]\right) \leq 1$$

where $\tilde{w}_p^S = \min\{W(S)-k, w_p\}$. Simplifying, we get that:

$$\sum_{t=t(p,j)+1}^{t(p,j+1)-1} \sum_{S \mid p\in S} \min\{W(S)-k, w_p\} y(t,S) \leq c_p(1+\ln k).$$

Thus, the dual solution can be made feasible by scaling it down by a factor of $(1+\ln k)$. We now prove that the primal cost is at most twice the dual profit, which means that the primal solution produced is $O(\log k)$-competitive.

We partition the primal cost into two parts, C_1 and C_2. Let C_1 be the contribution to the primal cost from Step (3) of the algorithm, due to the increase of variables $x(p,j)$ from 0 to $1/k$. Let C_2 be the contribution to the primal cost from Step (4) of the algorithm, due to the incremental increases of the variable $x(p,j)$ according to the exponential function.

Bounding C_1 Let $\tilde{x}(p,j) = \min(x(p,j), \frac{1}{k})$. We bound the term $\sum_{p=1}^{n} \sum_{j=1}^{r(p,t)} c_p \tilde{x}(p,j)$. To do this, we need two observations. First, from design of the algorithm, it follows that if $x(p,j) > 0$, and equivalently if $\tilde{x}(p,j) > 0$, then:

$$\sum_{t=t(p,j)+1}^{t(p,j+1)-1} \sum_{S \mid p\in S} \min\{W(S)-k, w_p\} y(t,S) \geq c_p \tag{8.17}$$

We shall refer to (8.17) as primal complementary slackness.

Next, in the dual solution if $y(t,S) > 0$, then:

$$\sum_{p\in S\setminus\{p_t\}} \min\{W(S)-k, w_p\} \tilde{x}(p, r(p,t)) \leq W(S)-k \tag{8.18}$$

We shall refer to (8.18) as dual complementary slackness. To see why (8.18) holds, consider the following two cases depending on whether $W(S) \geq k+1$ or not. Recall that $\tilde{x}(p, r(p,t)) \leq 1/k$ for all pages. If $W(S) \geq k+1$ then:

$$\begin{aligned} \sum_{p\in S\setminus\{p_t\}} \frac{1}{k} \cdot \min\{W(S)-k, w_p\} &\leq \frac{1}{k} \cdot \sum_{p\in S\setminus\{p_t\}} w_p \\ &= \frac{W(S)-w(p_t)}{k} \leq \frac{W(S)-1}{k} \leq W(S)-k. \end{aligned}$$

If $W(S) < k+1$, then the set S contains at most k pages. In this case we get that:

$$\begin{aligned} \sum_{p\in S\setminus\{p_t\}} \frac{1}{k} \cdot \min\{W(S)-k, w_p\} &\leq \frac{1}{k} \cdot \sum_{p\in S\setminus\{p_t\}} (W(S)-k) \\ &\leq \frac{k-1}{k} \cdot (W(S)-k) \leq W(S)-k. \end{aligned}$$

The last inequality follows since $W(S) \geq k$. These primal and dual complementary slackness conditions imply the following.

$$
\begin{aligned}
& \sum_{p=1}^{n} \sum_{j=1}^{r(p,t)} c_p \tilde{x}(p,j) && (8.19)\\
\leq \; & \sum_{p=1}^{n} \sum_{j=1}^{r(p,t)} \left(\sum_{t=t(p,j)+1}^{t(p,j+1)-1} \sum_{S \mid p \in S} \tilde{w}_p^S y(t,S) \right) \tilde{x}(p,j) && (8.20)\\
= \; & \sum_{t} \sum_{S \subseteq B(t), p_t \in S} \left(\sum_{p \in S \setminus \{p_t\}} \tilde{w}_p^S \tilde{x}(p, r(p,t)) \right) y(t,S) && (8.21)\\
\leq \; & \sum_{t} \sum_{S \subseteq B(t), p_t \in S} \left(W(S) - k\right) y(t,S). && (8.22)
\end{aligned}
$$

Inequality (8.20) follows from Inequality (8.17), Equality (8.21) follows by changing the order of summation, and Inequality (8.22) follows from Inequality (8.18). Thus, C_1 is at most the profit of a *feasible* dual solution multiplied by $(1 + \ln k)$.

Bounding C_2 We bound the derivative of the primal cost of variables $x(p,j)$ in Step (4) by the derivative of the dual profit accrued in the same round. Variables $x(p,j)$ that have already reached the value of 1 do not contribute anymore to the primal cost. The derivative of a variable $x(p,j)$, $1/k \leq x(p,j) < 1$, as a function of $y(t)$ is:

$$\frac{dx(p,j)}{dy(t,S)} = \frac{\min\{W(S) - k, w_p\}}{c_p} \cdot x(p,j). \tag{8.23}$$

Therefore, the derivative of the primal is at most:

$$
\begin{aligned}
\frac{dX}{dy(t,S)} &= \sum_{p \in S \setminus \{p_t\} : x(p, r(p,t)) < 1} \tilde{w}_p^S x(p, r(p,t))\\
&\leq W(S) - k = \frac{dY}{dy(t,S)}.
\end{aligned}
$$

The inequality in the second step above follows since the primal constraint of the set S is unsatisfied yet. Thus, C_2 is at most the profit of a feasible dual solution multiplied by $(1 + \ln k)$.

Completing the analysis It follows that $C_1 + C_2$ is at most twice the profit of a *feasible* dual solution multiplied by $(1 + \ln k)$. Note that the profit of any dual feasible solution is always a lower bound on the optimal solution. Therefore, we conclude by weak duality that the algorithm is $O(\log k)$-competitive. □

8.4 Rounding the Fractional Solution Online

In this section we show how to obtain a randomized online algorithm from the fractional solution generated previously. The ideas here generalize the ideas in the simpler weighted caching case. For convenience of analysis, throughout this section we consider the (equivalent) cost version of the problem where we pay c_p for both fetching and evicting a page p. This assumption can change the cost of the fractional solution by at most a factor of two. At any time step, the LP solution $x_1, \ldots, x_n$, where we denote by $x_p \triangleq x(p, r(p,t))$, specifies the probability that each of the pages is absent from the cache. However, to obtain an actual randomized algorithm we need to specify a probability distribution over the various cache states that is consistent with the LP solution. That is, we need to simulate the moves of the LP over the set of pages by consistent moves over the actual cache states. We adopt the following approach to do this simulation.

Let $\gamma \geq 1$ be a parameter and set $y_p = \min(\gamma x_p, 1)$. Let μ be a distribution on subsets of pages. We say that μ is consistent with y (or γ-consistent with x) if μ induces the distribution y on the page set. That is,

$$\forall p: \qquad \sum_D A_p^D \cdot \mu(D) = y_p, \tag{8.24}$$

where, for a set of pages D, $A_p^D = 1$ if $p \in D$ and 0 otherwise. We will view μ as a distribution over the complement of the cache states. To be a meaningful simulation, it suffices to require the following.

1. *Size Property:* For any set D with $\mu(D) > 0$, the sum of the sizes of the pages in D is at least $W(B(t)) - k$. That is, D corresponds to the complement of a valid cache.
2. *Bounded Cost Property:* If y changes to y' while incurring a fractional cost of d, the distribution μ can be changed to another distribution μ' which is consistent with y', while incurring a (possibly amortized) cost of at most βd, where $\beta > 0$.

It is easy to see that if x_p changes by ϵ, then y_p changes by at most $\gamma\epsilon$. Hence, given a fractional algorithm with competitive ratio c, the existence of a simulation with the above properties implies an actual randomized online algorithm with competitive ratio $\gamma\beta c$. We provide three different simulation procedures for the Bit model, General model, and the Fault Model. These are organized in increasing order of complexity.

8.4.1 The Bit Model

In this section we will show how to obtain an $O(\log k)$-competitive randomized algorithm for the general caching problem in the Bit model. Let $U \triangleq \lfloor \log_2 k \rfloor$. For $i = 0$ to U, we define the size class $S(i)$ to be the set of pages of sizes between 2^i and less than size 2^{i+1}. Formally, $S(i) = \{p \mid 2^i \leq w_p < 2^{i+1}\}$. Let $x_1, \ldots, x_n$ be the LP solution at the current time step. Recall that it satisfies the knapsack cover inequalities for all subsets. For each page p let $y_p = \min\{1, 3x_p\}$ (i.e. $\gamma = 3$).

Definition 8.1 (Balanced subsets). *We say that a subset of pages D is balanced with respect to y if:*

1. *If $y_p = 1$ then p is evicted in all cache states, i.e. $A_p^D = 1$ for all D with $\mu(D) > 0$.*
2. *The following holds for all $0 \le j \le U$:*

$$\left\lfloor \sum_{i=j}^{U} \sum_{p \in S(i)} y_p \right\rfloor \le \sum_{i=j}^{U} \sum_{p \in S(i)} A_p^D \le \left\lceil \sum_{i=j}^{U} \sum_{p \in S(i)} y_p \right\rceil . \tag{8.25}$$

We first show that the size property follows from the requirement that sets are balanced.

Lemma 8.6. *Let x and y be defined as above. Then, for any subset D which is balanced with respect to y, the sum of the sizes of all the pages in D is at least $W(B(t)) - k$.*

We first prove a simple mathematical claim.

Claim 8.7. *Let $x_1, x_2, \ldots, x_n$ and $y_1, y_2, \ldots, y_n$ be two sequences of non-negative real numbers and let $0 = a_0 \le a_1 \le a_2 \le \ldots, \le a_n$ be a non-decreasing sequence of positive numbers. If for every $1 \le j \le n$: $\sum_{i=j}^{n} x_i \ge -1 + (\sum_{i=j}^{n} y_i)$, then: $\sum_{i=1}^{n} a_i x_i \ge -a_n + \sum_{i=1}^{n} a_i y_i$.*

Proof. For every j, $1 \le j \le n$, multiply the jth inequality by $(a_j - a_{j-1})$ (which is non negative), yielding:

$$(a_j - a_{j-1}) \sum_{i=j}^{n} x_i \ge -(a_j - a_{j-1}) + (a_j - a_{j-1}) \sum_{i=j}^{n} y_i .$$

Summing up over all the inequalities yields the desired result. □

Proof. (Lemma 8.6). For the proof it suffices to use the LHS of condition (8.25) (i.e., the lower bound). Let $S' \subseteq S$ be the set of pages with $y_p < 1$, and let $S'(i) = S' \cap S(i)$ be the class i pages in S'. Since $A_p^D = 1$ whenever $y_p = 1$, condition (8.25) implies that for every $0 \le j \le U$:

$$\sum_{i=j}^{U} \sum_{p \in S'(i)} A_p^D \ge \left\lfloor \sum_{i=j}^{U} \sum_{p \in S'(i)} y_p \right\rfloor \ge \left(\sum_{i=j}^{U} \sum_{p \in S'(i)} y_p \right) - 1. \tag{8.26}$$

The sum of the sizes of the pages in D is $\sum_{p \in S} w_p A_p^D$. Since $A_p^D = 1$ for $p \in S \setminus S'$, it suffices to show that $\sum_{p \in S'} w_p A_p^D \ge W(S') - k$ for the proof. Consider the following:

$$\begin{aligned} & \sum_{p \in S'} w_p A_p^D \ge \sum_{p \in S'} \min\{w_p, W(S') - k\} A_p^D \\ = \; & \sum_{i=0}^{U} \sum_{p \in S'(i)} \min\{w_p, W(S') - k\} A_p^D \\ \ge \; & \frac{1}{2} \sum_{i=0}^{U} \sum_{p \in S'(i)} \min\{2w_p, W(S') - k\} A_p^D \end{aligned}$$

$$
\begin{aligned}
&\geq \frac{1}{2}\sum_{i=0}^{U}\min\{2^{i+1}, W(S')-k\}\sum_{p\in S'(i)} A_p^D && (8.27)\\
&\geq -\frac{1}{2}\min\{2^{U+1}, W(S')-k\}\\
&\quad +\frac{1}{2}\sum_{i=0}^{U}\min\{2^{i+1}, W(S')-k\}\sum_{p\in S'(i)} y_p && (8.28)\\
&\geq -\frac{1}{2}(W(S')-k)\\
&\quad +\frac{1}{2}\sum_{i=0}^{U}\sum_{p\in S'(i)}\min\{w_p, W(S')-k\}y_p && (8.29)\\
&\geq -\frac{1}{2}(W(S')-k)+\frac{3}{2}(W(S')-k)\geq W(S')-k. && (8.30)
\end{aligned}
$$

Here, Inequality (8.27) follows since $w_p \geq 2^i$ for each $p \in S'(i)$. Inequality (8.28) follows by applying Claim 8.7 with $a_i = \min\{2^{i+1}, W(S')-k\}$, $x_i = \sum_{p\in S'(i)} A_p^D$ and $y_i = \sum_{p\in S'(i)} y_p$, and observing that (8.26) implies that the conditions of the claim are satisfied. Inequality (8.29) follows since $w_p < 2^{i+1}$, and finally Inequality (8.30) follows since by the LP knapsack constraints, and the fact that $y_p = 3x_p$ for each $p \in S'$:

$$
\begin{aligned}
&\sum_{i=0}^{U}\sum_{p\in S'(i)}\min\{w_p, W(S')-k\}y_p\\
&= 3\sum_{p\in S'}\min\{w_p, W(S')-k\}x_p \geq 3(W(S')-k).
\end{aligned}
$$

□

We show how to maintain the bounded cost property using both the LHS and RHS of condition (8.25).

Lemma 8.8. *Let μ be any distribution on balanced sets that is consistent with y. Then the cost property holds with $\beta = 10$. That is, if y changes to y' while incurring a fractional cost of d, then the distribution μ can be modified to another distribution μ' over balanced sets such that μ' is consistent with y' and the cost incurred while modifying μ to μ' is at most $10d$.*

Proof. By considering each page separately, it suffices to show that the property holds whenever y_p increases or decreases for some page p. Assume first that the weight y_p of page p for $p \in S(i)$ is increased by ϵ. The argument when y_p is decreased is analogous. Page p belongs to $S(i)$, and so $w_p \geq 2^i$. Thus, the fractional cost is at least $\epsilon 2^i$.

We construct μ' as follows. To ensure the consistency with y',i.e., Equation (8.24), we add page p to ϵ measure of the sets D that do not contain p. Since this is the Bit model, this incurs a cost of at most $2^{i+1}\epsilon$. However this may violate condition (8.25) for classes $j \leq i$. We iteratively fix condition (8.25) starting with class i. Consider class i. Let $s = \lceil \sum_{j=i}^{U}\sum_{p\in S(j)} y_p \rceil$ and suppose first that $\lceil \sum_{j=i}^{U}\sum_{p\in S(j)} y'_p \rceil$ remains equal to

s. Then in μ', let ϵ' be the measure of sets that have $s+1$ pages in classes i or higher. Note that $\epsilon' \leq \epsilon$. Consider the sets with $s-1$ pages in classes i or higher and arbitrarily choose ϵ' measure of these (this is possible since $s = \lceil \sum_{j=i}^{U} \sum_{p \in S(j)} y'_p \rceil$). Arbitrarily pair the sets with $s+1$ pages to those with $s-1$ pages. Consider any pair of sets (D, D'). Since μ' satisfies condition (8.25) is for class $i+1$, the number of pages in D and D' that lie in classes $i+1$ or higher differ by at most 1. Hence, $D \setminus D'$ contains some class i page. We move this page from D to D'. Note that (8.25) is satisfied for i after this procedure. Now, consider the case when $\lceil \sum_{j=i}^{U} \sum_{p \in S(j)} y'_p \rceil$ increases to $s+1$. Note that in this case, the condition (8.25) is be violated for class i for at most $\epsilon' \leq \epsilon$ of sets that have precisely $s-1$ pages in classes i or higher. We arbitrarily pair the classes with $s-1$ to pages to those with $s+1$ pages and apply the argument above. The total cost incurred in this step is at most $(2\epsilon') \cdot 2^{i+1} \leq 2^{i+2}\epsilon$.

After applying the above procedure to fix class i, condition (8.25) might be violated for class $i-1$ for at most ϵ measure of sets. We apply the matching procedure sequentially to $i-1$ and lower classes incurring an additional cost of $\sum_{j=0}^{i-1} 2\epsilon \cdot 2^{j+1} < 4\epsilon 2^i$. Thus the total cost incurred is at most $10\epsilon 2^i$. $\square$

Theorem 8.9. *There is an $O(\log k)$-competitive algorithm for the general caching problem in the Bit model.*

8.4.2 The General Cost Model

In this section we study the General cost model and show how to obtain an $O(\log^2 k)$-competitive randomized caching algorithm for this model. Let $U \triangleq \lfloor \log_2 k \rfloor$. Let $C = \lfloor \log_2 C_{\max} \rfloor$. For $i = 0$ to U, and $j = 0$ to C, we define $S(i,j)$ to be the set of pages of sizes at least 2^i and less than 2^{i+1}, and fetching cost between 2^j and less than 2^{j+1}. Formally, $S(i,j) = \{p \mid 2^i \leq w_p < 2^{i+1} \text{ and } 2^j \leq c_p < 2^{j+1}\}$. Let $x_1, \ldots, x_n$ be the LP solution at the current time step that satisfies the knapsack cover inequalities for all subsets. Let $\gamma = U+3$. Thus, for each page p, $y_p = \min\{1, (U+3) \cdot x_p\} = O(\log k) \cdot x_p$.

Definition 8.2. *A set D of pages is balanced with respect to y if the following two conditions hold:*

1. *If $y_p = 1$ then p is evicted in all cache states, i.e. $A_p^D = 1$ for all D with $\mu(D) > 0$.*
2. *For each size class $0 \leq i \leq U$, it holds that for each $0 \leq j \leq \lfloor \log C_{\max} \rfloor$:*

$$\left\lfloor \sum_{z=j}^{C} \sum_{p \in S(i,z)} y_p \right\rfloor \leq \sum_{z=j}^{C} \sum_{p \in S(i,z)} A_p^D \leq \left\lceil \sum_{z=j}^{C} \sum_{p \in S(i,z)} y_p \right\rceil . \tag{8.31}$$

We first show that the size property follows from the requirement that the sets are balanced.

Lemma 8.10. *Let x and y be defined as above. Then, for any subset D that is balanced with respect to y, the sum of sizes of all pages in D is at least $W(B(t)) - k$.*

Proof. For the proof it suffices to use the LHS of condition (8.31) (i.e., the lower bound). Let S' denote the subset of pages with $y_p < 1$. As $y_p = 1$ whenever $A_p^D = 1$, it suffices to show that $\sum_{p \in S'} w_p A_p^D \geq W(S') - k$. Moreover, condition (8.31) implies that for any $0 \leq i \leq U$:

$$\sum_{z=0}^{C} \sum_{p \in S'(i,z)} A_p^D \geq \lfloor \sum_{z=0}^{C} \sum_{p \in S'(i,z)} y_p \rfloor \geq -1 + \sum_{z=0}^{C} \sum_{p \in S'(i,z)} y_p. \tag{8.32}$$

Thus, the total size of pages from S' that are in D can be lower bounded as follows:

$$\begin{aligned}
& \sum_{p \in S'} w_p A_p^D \geq \sum_{p \in S'} \min\{w_p, W(S') - k\} A_p^D \\
= \; & \sum_{i=0}^{U} \sum_{j=0}^{C} \sum_{p \in S'(i,j)} \min\{w_p, W(S') - k\} A_p^D \\
\geq \; & \frac{1}{2} \sum_{i=0}^{U} \sum_{j=0}^{C} \sum_{p \in S'(i,j)} \min\{2w_p, W(S') - k\} A_p^D \\
\geq \; & \frac{1}{2} \sum_{i=0}^{U} \min\{2^{i+1}, W(S') - k\} \sum_{j=0}^{C} \sum_{p \in S'(i,j)} A_p^D && (8.33) \\
\geq \; & \frac{1}{2} \sum_{i=0}^{U} \min\{2^{i+1}, W(S') - k\} \left(-1 + \sum_{j=0}^{C} \sum_{p \in S'(i,j)} y_p \right) && (8.34) \\
\geq \; & -\frac{U+1}{2} (W(S') - k) \\
& + \frac{1}{2} \sum_{i=0}^{U} \sum_{j=0}^{C} \sum_{p \in S'(i,j)} \min\{w_p, W(S') - k\} y_p && (8.35) \\
\geq \; & -\frac{U+1}{2} (W(S') - k) + \frac{U+3}{2} (W(S') - k) \\
= \; & W(S') - k. && (8.36)
\end{aligned}$$

Inequality (8.33) follows since $w_p \geq 2^i$ for each page $p \in S'(i,j)$, and Inequality (8.34) follows from (8.32). Inequality (8.35) follows since $w_p \leq 2^{i+1}$ for each page $p \in S'(i,j)$. Finally, Inequality (8.36) follows by the knapsack constraints:

$$\begin{aligned}
& \sum_{i=0}^{U} \sum_{j=0}^{C} \sum_{p \in S'(i,j)} \min\{w_p, W(S') - k\} y_p \\
= \; & \sum_{p \in S'} \min\{w_p, W(S') - k\} y_p \\
= \; & (U+3) \sum_{p \in S'} \min\{w_p, W(S') - k\} x_p \\
\geq \; & (U+3)(W(S') - k).
\end{aligned}$$

Here we use the fact that $y_p = (U+3) x_p$ for $p \in S'$. ☐

We now show how to maintain the bounded cost property with $\beta = 10$. For this we need to use both the LHS and RHS of condition (8.31), and we use an argument similar to the one used in the proof of Lemma 8.8.

Lemma 8.11. *Give any distribution μ over balanced sets that is consistent with y. If y changes to y' incurring a fractional cost of d, then the distribution μ can be modified to another distribution μ' over balanced sets consistent with y' such that total cost incurred is at most $10d$.*

Proof. Suppose that y_p increases by ϵ and p lies in the class $S(i,j)$. Note that the balance condition (8.31) holds for every size class different from i, and moreover for size class i the condition also holds for all cost classes higher than j. We apply the procedure used in Lemma 8.8 to size class i. Note that applying this procedure does not have any effect on size classes different from i, and we can thus iteratively balance cost classes starting from j down to 0 in size class i. To bound the cost, observe that the analysis in the proof of Lemma 8.8 only used the fact that the cost of the classes are geometrically decreasing. Thus, a similar analysis implies that the cost incurred is no more than $10\epsilon \cdot 2^j$. ☐

We conclude with the next theorem:

Theorem 8.12. *There is an $O(\log^2 k)$-competitive algorithm for the caching problem in the General model.*

8.4.3 The Fault Model

In this section we study the Fault model and show how to obtain an $O(\log k)$-competitive randomized caching algorithm for this model. Note that an $O(\log^2 k)$-competitive algorithm follows directly from the result for the General model. Recall that in the proofs for the Bit model and the General model we crucially used the fact that the cost in the different classes is geometrically decreasing. However, this is not the case for the Fault model, making the proof significantly more involved and requiring the use of a potential function so as to perform an amortized analysis.

We sort the n pages with respect to their size, i.e., $w_1 \leq w_2 \leq \ldots \leq w_n$. Let $x_1, \ldots, x_n$ be the LP solution at the current time step that satisfies the knapsack cover inequalities for all subsets. For each page p, let $y_p = \min\{1, 15 \cdot x_p\}$. Let S' denote the set of pages with $y_p < 1$. During the execution of the algorithm we maintain a grouping $\mathcal{G}$ of pages in S' into groups $G(i)$, $1 \leq i \leq \ell$. Each group $G(i)$ contains a sequence of consecutive pages in S'. As the pages are ordered in non-decreasing order with respect to size, for any i the largest page size in group $G(i)$ is at most the smallest page size in $G(i+1)$.

Definition 8.3 (Good Grouping)**.** *A grouping $\mathcal{G}$ of pages in S' is called* good *if it satisfies the following properties.*

1. *For each i, $1 \leq i \leq \ell$, we have $\sum_{p \in S(i)} y_p \leq 12$.*

2. *If* $\sum_{p \in S'} y_p \geq 3$*, then for each group* i*,* $1 \leq i \leq \ell$*, we have* $\sum_{p \in G(i)} y_p \geq 3$*. If* $\sum_{p \in S'} y_p < 3$*, then there is exactly one group* $G(1)$ *containing all the pages in* S'*.*

We define $\sum_{p \in G(i)} y_p$ to be the *weight* of group $G(i)$.

Definition 8.4 (Balanced Set)**.** *Given a good grouping* G*, a set of pages* D *is called balanced if the following two properties hold.*

1. *If* $y_p = 1$*, then* $A_p^D = 1$*.*

2. *For each* i*, the number of pages* $|D \cap G(i)| = \sum_{p \in G(i)} A_p^D$ *satisfies*

$$\left\lfloor \sum_{p \in G(i)} y_p \right\rfloor \leq \sum_{p \in G(i)} A_p^D \leq \left\lceil \sum_{p \in G(i)} y_p \right\rceil . \tag{8.37}$$

The simulation procedure works as follows. At any time the algorithm maintains a good grouping $\mathcal{G}$ of the pages. It also maintains a probability distribution μ on balanced sets D which is consistent with y. At each step of the algorithm, as the value of y changes, the algorithm modifies the distribution μ to be consistent with y. Additionally, as y changes, the grouping $\mathcal{G}$ may also possibly change (so as to remain good), in which case a previously balanced set need not remain balanced anymore. In such a case, we also modify μ since only balanced sets can belong to the support of μ.

We first show that the size property holds for balanced sets D, and then show how to update $\mathcal{G}$ and μ as y changes, such that the cost property holds with $\beta = O(1)$ in an amortized sense.

Lemma 8.13. *Let* y *be as defined above and let* $\mathcal{G}$ *be a good grouping with respect to* y*. Then any balanced set* D *with respect to* $\mathcal{G}$ *has size at least* $W(S) - k$*.*

Proof. Let S' be the set of pages p for which $y_p < 1$. As D is balanced, each page with $y_p = 1$ belongs to D and hence it suffices to show that $\sum_{p \in S'} w_p A_p^D \geq W(S') - k$. If $W(S') - k \leq 0$, then we are already done. Henceforth we assume that $W(S') - k > 0$.

The linear program constraint for the set S' implies that $\sum_{p \in S'} \min\{w_p, W(S') - k\} x_p \geq W(S') - k$. This implies that $\sum_{p \in S'} x_p \geq 1$ and so $\sum_{p \in S'} y_p \geq 15$. Hence by the second condition for a good grouping, each group $G(i)$ has weight at least 3.

For each group $G(i)$ let $w_i(\min)$ and $w_i(\max)$ denote the smallest and largest page size in $G(i)$. Recall that for each i, we have that $w_i(\min) \leq w_i(\max) \leq w_{i+1}(\min)$. (Define $w_{\ell+1}(\min) = w_\ell(\max)$.) Let $m_i = \min(w_i(\min), W(S') - k)$ for $i = 1, \ldots, \ell + 1$. We lower bound the total size of pages in $D \cap S'$ as follows.

$$\begin{aligned} \sum_{p \in S'} w_p A_p^D &\geq \sum_{p \in S'} \min\{w_p, W(S') - k\} A_p^D \\ &= \sum_{i=1}^{\ell} \sum_{p \in G(i)} \min\{w_p, W(S') - k\} A_p^D \end{aligned}$$

$$
\begin{aligned}
&\geq \sum_{i=1}^{\ell} m_i \sum_{p \in G(i)} A_p^D \geq \sum_{i=1}^{\ell} m_i(-1 + \sum_{p \in G(i)} y_p) \\
&\geq \frac{2}{3} \sum_{i=1}^{\ell} m_i \sum_{p \in G(i)} y_p && (8.38) \\
&= \frac{2}{3} \left(\sum_{i=1}^{\ell} m_{i+1} \sum_{p \in G(i)} y_p \right) \\
&\quad -\frac{2}{3} \left(\sum_{i=1}^{\ell} (m_{i+1} - m_i) \sum_{p \in G(i)} y_p \right) \\
&\geq \frac{2}{3} \left(\sum_{i=1}^{\ell} m_{i+1} \sum_{p \in G(i)} y_p \right) - 8 \left(\sum_{i=1}^{\ell} (m_{i+1} - m_i) \right) && (8.39) \\
&= \frac{2}{3} \left(\sum_{i=1}^{\ell} m_{i+1} \sum_{p \in G(i)} y_p \right) - 8m_{\ell+1} + 8m_1 \\
&\geq \frac{2}{3} \left(\sum_{i=1}^{\ell} \sum_{p \in G(i)} \min\{w_p, W(S') - k\} y_p \right) \\
&\quad - 8(W(S') - k) && (8.40) \\
&\geq 2(W(S') - k).
\end{aligned}
$$

Here inequality (8.38) follows since D is balanced, and hence for each $1 \leq i \leq \ell$,

$$\sum_{p \in G(i)} A_p^D \geq \left\lfloor \sum_{p \in G(i)} y_p \right\rfloor \geq -1 + \sum_{p \in G(i)} y_p,$$

and by observing that $\mathcal{G}$ is good and hence $\sum_{p \in G(i)} y_p \geq 3$ for each $1 \leq i \leq \ell$ and thus

$$-1 + \sum_{p \in G(i)} y_p \geq \frac{2}{3} \left(\sum_{p \in G(i)} y_p \right).$$

Inequality (8.39) follows since $m_{i+1} - m_i \geq 0$ for each $1 \leq i \leq \ell$, and since $\mathcal{G}$ is good, for each $1 \leq i \leq \ell$ we have that $\sum_{p \in G(i)} y_p \leq 12$: Finally, Inequality (8.40) follows by considering the knapsack cover inequality for the set S' and observing that $y_p = 15x_p$ for each $p \in S'$:

$$
\begin{aligned}
& \sum_{i=1}^{\ell} \sum_{p \in G(i)} \min\{w_p, W(S) - k\} y_p \\
= & \sum_{p \in S'} \min\{w_p, W(S') - k\} 15x_p \geq 15(W(S') - k).
\end{aligned}
$$

$\square$

Lemma 8.14. *As the solution y changes over time we can maintain a good grouping $\mathcal{G}$ and a consistent distribution on balanced sets with amortized cost at most a constant times the fractional cost.*

Proof. The online fractional algorithm has the following dynamics. After a page p is requested variable y_p can only increase (the page is gradually evicted). This process stops when page p is requested again and y_p is set to zero. Whenever y_p changes, we need to modify the distribution μ on balanced sets D to remain consistent. Moreover, a change in y_p may change the structure of the groups. This happens if either the weight of $G(i)$ exceeds 12, or if it falls below 3, or if y_p becomes 1 and leaves the group $G(i)$ (recall that groups only contain pages q with $y_q < 1$). We view a change in y_p as a sequence of steps where y_p changes by an infinitesimally small amount ϵ. Thus at each step exactly one of the following events happens.

Event 1: Variable $y_p < 1$ of page p increases or decreases by ϵ.

Event 2: The weight of group $G(i)$ reaches 12 units.

Event 3: The weight of group $G(i)$ drops to 3 units.

Event 4: The value of y_p for page p reaches 1 and p leaves the set $S(i)$.

We prove that in all cases the amortized cost of the online algorithm is at most $O(1)$ times the fractional cost. For amortization we use the following potential function:

$$\Phi = 13 \sum_{p \in S'} y_p + 11 \sum_{i=1}^{\ell} \left| 6 - \sum_{p \in G(i)} y_p \right| .$$

In each possible event let C_{on} be the total cost of the online algorithm. Let C_f be the fractional cost, and let $\Delta\Phi$ be the change in the potential function. We show that in each of the events:

$$\Delta C_{on} + \Delta\Phi \leq 405 \Delta C_f \tag{8.41}$$

Since Φ is always positive, this will imply the desired result.

Event 1 Assume first that y_p such that $p \in G(i)$ is increased by ϵ. If y_p increases by ϵ it must be that x_p is increased by at least $\frac{\epsilon}{15}$. Thus, in the fault model the fractional cost is at least $\frac{\epsilon}{15}$.

To maintain consistency, we add p to ϵ measure of the sets D that do not contain p. However this might make some of these sets unbalanced by violating (8.37). Suppose first that $s = \lceil \sum_{p \in G(i)} y_p \rceil$ does not change when y_p is increased by ϵ. In this case, we match the sets with $s+1$ pages in $G(i)$ (the measure of these is at most ϵ) arbitrarily with sets contains $s-1$ pages, and transfer some page from the larger set (that does not lie in the smaller set) to the smaller set. An analogous argument works when s increases as y_p is increased. Note that after this step, the sets become balanced.

The total online cost is 3ϵ. Moreover, the potential change $\Delta\Phi$ is at most $13\epsilon + 11\epsilon = 24\epsilon$ and hence (8.41) holds. An analogous argument works if y_p is decreased (in fact it is even easier since the potential only decreases).

Event 2 Consider an event in which the total weight of a group $G(i)$ reaches 12 units. In this case we split $G(i)$ into two sets such that their weight is as close to 6 as possible. Suppose one set is of size $6 + x$ and the other is of size $6 - x$ where $0 \leq x \leq 1/2$. Let $\Phi(s)$ and $\Phi(e)$ denote the potential function before and after the change respectively. The contribution of the first term does not change. The second term corresponding to $G(i)$ initially is at least $11(12-6) = 66$ and the final contribution is $11(|6-(6-x)|+|6-(6+x)|) = 22x \leq 11$. Thus $\Delta\Phi = \Phi(e) - \Phi(s) = 11 - 66 \leq -55$.

Next, we redistribute the pages in the original group $G(i)$ among the sets D such that they are balanced with respect to the two new groups. Observe that in the worst case, each set D might need to remove all the 12 pages it previously had and bring in at most $\lceil 6 + x \rceil + \lceil 6 - x \rceil \leq 13$ new pages. Since the measure of sets is D, the total cost incurred is at most 25. Again, (8.41) holds as the fractional cost C_f is 0 and the decrease in potential more than offsets the cost C_{on}.

Event 3 Consider the event when the weight of a group $G(i)$ decreases to 3 units. If $G(i)$ is the only group (i.e. $\ell = 1$) then all properties of a good grouping still hold. Otherwise, we merge $G(i)$ with one of its neighbors (either $G(i-1)$ or $G(i+1)$). If $G(i)$ has a neighbor with weight at most 9, then we merge $G(i)$ with this neighbor. Note that before the merge, each balanced set D has exactly 3 pages from $G(i)$ and hence it also remains balanced after the merge. Also, since $|6-3| + |6-x| \geq |6-(x+3)|$ for all $3 \leq x \leq 9$, and hence the potential function does not increase in this case. Thus (8.41) holds trivially.

Now suppose that all neighbors of $G(i)$ have weight greater 9. Consider any such neighbor and let $x > 9$ be its weight. We merge $G(i)$ with this neighbor to obtain a group with weight $3 + x$ which lies in the range $(12, 15]$. Then as in the handling of Event 8.4.3, we split this group into two groups with as close weight as possible. Since the weight is at most 15, the cost of balancing the sets D is at most $16 + 15 = 31$ (using argument similar to that in Event 8.4.3). We now consider the change in potential. The only change is due to second terms corresponding to $G(i)$ and its neighbor (the first term does not matter since total weight of pages in S' does not change upon merging or splitting). Before the merge, the contribution was $11 \cdot 3 + 11 \cdot (x - 6) = 11x - 33 \geq 66$. After the merge (and the split) the maximum value of the potential is obtained for the case when the size of the merged group is 15 which upon splitting leads to sets of size $7 + y$ and $8 - y$ where $y \leq 0.5$, in which case its value is $11(1 + y + 2 - y) = 33$. Thus, the potential function decreases by at least 33 while the online cost is at most 31, and hence (8.41) holds.

Event 4 Suppose some y_p increases to 1 and exits the group $G(i)$. Note that if $y_p = 1$, then all balanced sets D contain p. Thus, removing p from $G(i)$ keeps the sets balanced.

Let us first assume that the weight of $G(i)$ does not fall below 3 when p is removed. In this case, the groups and the balanced sets remain unchanged. Thus the online algorithm incurs zero cost. The first term of the potential decreases by 13, and the second term increases by at most 11, and hence (8.41) holds. Now consider the case when the weight of $G(i)$ falls below 3. We apply an argument similar to that for Event 8.4.3. If $G(i)$ can be merged with some neighbor without weight exceeding 12, then we do so. This merge may cause some sets D to become imbalanced. However, this imbalance is no more than one page and can be fixed by transferring one page from each set to another appropriate set. The total cost incurred in this case is at most 2. We now consider the change in potential. The first term decreases by 13. For the second term, the original group $G(i)$ contributes function $11(6-(3+x)) = 11(3-x)$, with $x < 1$ and its neighbor contributes $11(|6-z|)$ where $3 \leq z \leq 9$ is its weight. After the merge, the second term corresponding to the merged group contributes $11(|6-(z+2+x)|)$ which is at most $11(|6-z|+(2+x))$. Overall, $\Delta\Phi \leq -13+11(2+x)-11(3-x) = 22x-24 < -2$. Thus (8.41) holds.

If we need to split the merged set, we note that the above analysis, showing that (8.41) holds, is also valid when $9 \leq z \leq 12$. Next, when this merged set is split, we can apply the analysis in Event 8.4.3, and then the potential function decreases by at least 33 units, while the cost incurred is at most 31, and hence (8.41) holds. □

We conclude with the next theorem:

Theorem 8.15. *There is an $O(\log k)$-competitive algorithm for the caching problem in the Fault model.*

8.5 Notes

The results in this chapter are based on the work of Bansal, Buchbinder and Naor [14, 15]. The weighted caching problem was studied in [14], while the more general setting where pages have both sizes and fetching costs was studied in [15]. Transforming a fractional view to an actual view has been considered previously by [19, 24]. Blum et al. [24] showed that for the unweighted caching problem it is possible to transform online a fractional view to an actual view such that the expected cost incurred is at most twice the cost of the fractional view.

The unweighted paging problem is very well understood. In their seminal paper, Sleator and Tarjan [86] showed that any deterministic algorithm is at least k-competitive, and also showed that LRU (Least Recently Used) is exactly k-competitive. They also considered the more general (h,k)-paging problem where the online algorithm with cache size k is compared to the offline algorithm with cache size h. They showed that any deterministic algorithm is at least $k/(k-h+1)$-competitive, and that LRU is exactly $k/(k-h+1)$-competitive. When randomization is allowed, Fiat et al. [47] designed the Randomized Marking algorithm which is $2H_k$-competitive against an oblivious adversary, where H_k is the k-th Harmonic number. They also showed that any randomized algorithm is at least H_k-competitive. Subsequently, McGeoch and Sleator [79] gave a matching H_k-competitive algorithm, and Achlioptas, Chrobak and Noga [1] gave another H_k-competitive algorithm that is easier to state and analyze. For (h,k)-paging, Young [89] gave a $2\ln(k/(k-h))$-competitive algorithm (ignoring lower order terms) and showed that any algorithm is at least $\ln(k/(k-h))$-competitive. There has been extensive work on paging along several other directions, and we refer the reader to the excellent text by Borodin and El-Yaniv [28] for further details.

For weighted paging, a tight k-competitive deterministic algorithm follows from the more general work of Chrobak et al. [39] for k-server problem on trees (see below). Subsequently, Young [90] gave a tight $k/(k-h+1)$-competitive deterministic algorithm for the more general (h,k)-paging problem. The randomized competitiveness of the weighted paging problem was not clear until the work of Bansal, Buchbinder and Naor [14]. Irani [65] gave an $O(\log k)$-competitive algorithm for the two weight case, i.e. when each page weight is either 1 or some fixed $M > 1$. In another direction, Blum, Furst and Tomkins [25] gave an $O(\log^2 k)$-competitive algorithm for the case of $n = k+1$ pages. Later, Fiat and Mendel [49] gave an improved $O(\log k)$ competitive algorithm for the case of $n = k+c$ pages, where c is a constant. For large n however, no $o(k)$-competitive algorithm was known even for the case of three distinct weights.

Paging can be viewed as a special case of the much more general and challenging k-server problem. In this problem, there are k servers located on points in an n-point metric space. At each time step a request is placed at one of the points and the algorithm must move one of the servers to this point to serve the request. The goal is to minimize the overall distance traveled by the servers. The unweighted paging problem is exactly the k-server problem on a uniform metric space. The weighted paging problem is identical (up to an additive constant) to the k-server problem on the metric space in which the distance between any two pages a and b is $(w(a)+w(b))/2$, where $w(\cdot)$ denotes the page weights.

The k-server problem has a fascinating history and substantial progress has been made on deterministic algorithms for the problem. It is known that any deterministic algorithm must be at least k-competitive on any metric space with more than k points. Fiat, Rabani and Ravid [50] gave the first algorithm for which the competitive ratio was only a function of k. Their algorithm was $O((k!)^3)$-competitive. After a series of results, a breakthrough was achieved by Koutsoupias and Papadimitriou [74] who gave an almost tight $2k-1$ competitive algorithm. This is still the best known bound (both for deterministic and randomized algorithms) for general metric spaces. The tight guarantee of k is also known for many special cases. In particular, Chrobak et al. [39] gave a k-competitive algorithm for trees. We refer the reader to [28] for more details on the k-server problem.

Nevertheless, randomized algorithms for the k-server problem remain poorly understood. No lower bound better than $\ln k$ is known for any metric space. Moreover, from the work of Bartal, Bollobas and Mendel [18] and Bartal, Linial, Mendel and Naor [20], it follows that no metric space with more than k points can admit a $o(\log k/\log\log k)$-competitive algorithm. A widely believed conjecture is that $O(\log k)$-competitive algorithms exist for every metric space. In a breakthrough result, Bartal, Blum, Burch and

Tomkins [19] gave a poly-log(N) competitive algorithm for the metrical task system problem (see definitions in the following paragraph) that implies a poly-log(k)-competitive algorithm for the k-server on a space with $k + c$ points, where c is a constant independent of k. This guarantee was improved by Fiat and Mendel [49] to $O(\log^2 k \log\log k)$. However, for n much larger than k, no algorithms with sublinear competitive ratio are known except for very few special cases. Besides paging and weighted paging with two weights, a poly-logarithmic competitive algorithm is known for a special subclass of certain well-separated spaces [85]. Csaba and Lodha [43] gave an $O(n^{2/3})$ competitive algorithm, which is $o(k)$ competitive for $n = o(k^{3/2})$, for n uniformly spaced points on a line[2].

General caching where the page sizes are also non-uniform is substantially harder. In contrast to uniform page size caching, even the offline version of the problem is NP-hard, as it captures the knapsack problem[3]. Following a sequence of results [64, 2, 42], Bar-Noy et al. [16] gave a 4-approximation for the problem based on the local-ratio technique. This is currently the best known approximation for (offline) general caching. For the online case it is known that LRU is $(k+1)$-competitive for the Bit model and also for the Fault model [64], where k denotes the ratio between cache size and the size of the smallest page. Later on, Cao and Irani [34] and Young [92] gave a $(k + 1)$-competitive algorithm for the General model based on a generalization of the Greedy-Dual algorithm of Young [90]. An alternate proof of this result was obtained by [42]. When randomization is allowed, Irani [64] designed an $O(\log^2 k)$-competitive algorithm for both Fault and Bit models. These algorithms are very complicated and are based on an approach combining offline algorithms with the Randomized Marking algorithm. For the General model, no $o(k)$ randomized algorithms are known. There has been extensive work on caching in other directions, and we refer the reader for further details to the excellent text by Borodin and El-Yaniv [28] and to the survey by Irani [63] on paging.

[2]A generalization of this result was considered by Bartal and Mendel [21], who proposed a $\Delta^{1-\epsilon}$polylog$k$ competitive algorithm for bounded growth metrics with diameter Δ. Unfortunately, their result seems to have a serious error [M. Mendel, personal communication].

[3]It remains NP-hard for the Bit model. For the Fault model it is open whether the problem is polynomially solvable [64].

Chapter 9

Load Balancing on Unrelated Machines

In this chapter we show how to use the primal dual method to design an optimal algorithm for the problem of online load balancing on unrelated machines. In this setting we are given a set of m machines $\mathbb{S} = \{s_1, s_2, \ldots, s_m\}$. For each job $r_i \in \mathbb{R}$ and machine s_j, there is an arbitrary load $p(i,j)$ of processing job the job on that machine. The load on each machine is the sum of $p(i,j)$ of jobs that are processed on that machine. Our goal is to distribute the jobs between the m machines so as to minimize the maximum load on a machine. In the online setting the jobs arrive one-by-one and need to be assigned to a machine upon arrival. The assignment of a job cannot be changed.

9.1 LP formulation and Algorithm

The first idea we need is the idea of guessing the value of the optimum. That is, our online algorithm is going to "guess" the value of the maximum load Λ^* that is needed in order to process all jobs. This will be done by starting from value $\alpha = \min_{j=1}^{m}\{p(1,j)\}$ and doubling our guess whenever needed until $\alpha \geq \Lambda^*$. We design an algorithm that never assigns more than $\alpha \cdot O(\log m)$ units of load on any machine. The algorithm guarantees success in assigning all jobs when it is given a value $\alpha \geq \Lambda^*$. When it is given a value $\alpha < \Lambda^*$ it may return a failure. It is not hard to see that due to this doubling process the competitive ratio is multiplied by 4.

Given a guess α we define the normalized load of job r_i on machine s_j to be $\tilde{p}(i,j) = \frac{p(i,j)}{\alpha}$. When job j arrives we can only consider machines for which $\tilde{p}(i,j) \leq 1$, since when $\alpha \geq \Lambda^*$ the job can only be processed by the optimal solution on such machines. If there is no such machine then our guess of Λ^* is wrong and the algorithm can return failure. Next, we formulate the problem as a packing linear program. For each job r_i, let $\mathbb{S}(r_i)$ be the set of machines for which $\tilde{p}(i,j) \leq 1$. We have a variable $y(i,j)$ indicating that job r_i is assigned to machine s_j. The objective function is to maximize the number of jobs assigned to the machines. The formulation appears as the dual program (maximization)

Primal		Dual	
Minimize:	$\sum_{s_j\in\mathbb{S}} x(j) + \sum_{r_i\in\mathbb{R}} z(i)$	Maximize:	$\sum_{r_i\in\mathbb{R}}\sum_{s_j\in\mathbb{S}(r_i)} y(i,j)$
Subject to:		Subject to:	
$\forall r_i\in\mathbb{R}, s_j\in\mathbb{S}(r_i)$:	$\tilde{p}(i,j)x(j)+z(i)\geq 1$	$\forall r_i\in\mathbb{R}$:	$\sum_{s_j\in\mathbb{S}(r_i)} y(i,j)\leq 1$
		$\forall s_j\in\mathbb{S}$:	$\sum_{r_i\in\mathbb{R}, s_j\in\mathbb{S}(r_i)} \tilde{p}(i,j)y(i,j)\leq 1$

Figure 9.1: A primal dual pair for the load balancing problem on unrelated machines.

in Figure 9.1 along with its corresponding primal program.

Let $N = |\mathbb{R}|$ be the number of jobs. The important observation is that when we guess a value $\alpha \geq \Lambda^*$ then it is possible to process all jobs on the machines without exceeding the load. This means that the value of the optimal solution to the dual LP is exactly N. On the negative side it means that there is no primal LP solution that has value strictly less than N. We are now ready for our algorithm:

Balance Algorithm: Initially: $x(j) \leftarrow \frac{1}{2m}$. When a new job r_i arrives:

1. If there is no machine j such that $\tilde{p}(i,j) \leq 1$, or there exists a machine with $x(j) > 1$ return "failure". Otherwise:
   (a) Let $s_\ell \in \mathbb{S}(r_i)$ be a machine with minimal value of $\tilde{p}(i,\ell)x(\ell)$.
   (b) Assign request r_i to machine s_ℓ and set $y(i,\ell) \leftarrow 1$.
   (c) Set $z(i) \leftarrow 1 - \tilde{p}(i,\ell)x(\ell)$.
   (d) $x(\ell) \leftarrow x(\ell)(1 + \frac{\tilde{p}(i,\ell)}{2})$.

Theorem 9.1. *If there exists a feasible solution to the LP that processes* **all** *jobs, then the algorithm processes* **all** *jobs with load $O(\log m)$.*

Proof. To prove the theorem we prove the following claims:

1. The load on each machine is at most $O(\log m)$.
2. If the algorithm returns "failure" in line 1 then there is a primal feasible solution that is strictly smaller than $|\mathbb{R}| = N$.

Proof of (1): To prove the first part of the claim note that the algorithm never assigns a job on machines with $x(j) > 1$ (otherwise, it already fails in Line 1. Also, since for each job and each machine, $\tilde{p}(i,j) \leq 1$ then $x(j) \leq 3/2$. Let $\mathbb{R}(s_j)$ be the set of jobs that the algorithm routed on machine s_j. We, thus, have the following inequalities:

$$\frac{1}{2m}\exp\left(\ln\frac{3}{2}\sum_{r_i\in\mathbb{R}(s_j)}\tilde{p}(i,j)\right) \leq \frac{1}{2m}\prod_{r_i\in\mathbb{R}(s_j)}(\frac{3}{2})^{\tilde{p}(i,j)} \leq \frac{1}{2m}\prod_{r_i\in\mathbb{R}(s_j)}\left(1+\frac{\tilde{p}(i,j)}{2}\right) = x(j) \leq 3/2.$$

Simplifying, we get that:

$$\sum_{r_i \in \mathbb{R}(s_j)} \tilde{p}(i,j) \leq \frac{\ln 3m}{\ln \frac{3}{2}} = O(\log m).$$

Proof of (2): First note that our algorithm produces a feasible primal solution. This follows since for each job we set $z(i) \leftarrow 1 - \tilde{p}(i,\ell)x(\ell)$, where ℓ is the machine that minimizes the value $\tilde{p}(i,\ell)x(\ell)$. Thus, we satisfy all the new primal constraints that arrived in the current iteration. Since $x(i)$ are only increasing all previous constraints remain feasible. Next, observe that whenever we assign a job to a machine the change in the primal cost is $1 - \tilde{p}(i,\ell)x(\ell) + \frac{\tilde{p}(i,\ell)x(\ell)}{2} = 1 - \frac{\tilde{p}(i,\ell)x(\ell)}{2}$. Note, however, that the change in $x(\ell)$ due to the assignment of job r_i is exactly $\frac{\tilde{p}(i,\ell)x(\ell)}{2}$. Let $x(j)_{\text{init}}$ be the initial value of $x(j)$ (which is $1/2m$). Thus, by this observation at any time during the execution of the algorithm the cost of the primal solution is:

$$\begin{aligned} P &= \sum_{j=1}^{m} x(j)_{\text{init}} + N - \sum_{j=1}^{m} (x(j) - x(j)_{\text{init}}) \\ &= 2\sum_{j=1}^{m} x(j)_{\text{init}} + N - \sum_{j=1}^{m} x(j) = 1 + N - \sum_{j=1}^{m} x(j) \end{aligned}$$

where N is the number of jobs. Assume now that there exists some $x(j) > 1$. This means that we have a primal solution with cost strictly less than N, meaning that there can be no dual solution with profit exactly N, concluding the proof. □

9.2 Notes

The results in this chapter are based on the work of Buchbinder and Naor [33]. The algorithm described in this chapter and its analysis are actually a primal-dual view of a previous algorithm by Aspnes et al. [8].

Many load balancing models were studied in the literature. Perhaps the simplest model is the identical machines model in which there are m identical machines. n jobs arrive online each with load p_i and the assignment of a job cannot be changed. For this model Graham [60] proved that the simple greedy heuristic that assigns the next task to the least loaded machine is $2 - 1/n$-competitive. Another model that was studied is the restricted assignment model. In this model each job can be assigned only to a subset of the machines (and not to all of them as in the identical machines model). For this model [13] analyzed the performance of the same greedy strategy, proving that it is $O(\log m)$-competitive. They also showed that this factor is optimal for this model. More refined performance measures of the greedy strategy were later studied in [57, 33]. For further discussion about load balancing in many models we refer the reader to [12].

Chapter 10

Routing

In this chapter we study routing problems. We already discussed two simple routing algorithms and a primal dual approach for solving routing problems in Section 4.4.2. In this chapter we will design more complex routing algorithms taking into account other objective functions.

Routing and call admission problems in various models have been studied extensively in both offline and online settings. Consider a network modeled by a graph $G = (V, E)$ ($|V| = n$, $|E| = m$), which can be either directed or undirected. The edges in the graph have capacities, denoted by $u : E \to \mathbb{N}$, which provide an upper bound on the sum of the demands of the routes that can be packed into an edge. The set of routing requests is $\mathbb{R}$ and, for simplicity, each request $r_i \in \mathbb{R}$ is associated with a bandwidth demand of one unit[1] between a source vertex s_i and a target vertex t_i. In order to serve a request r_i one should allocate bandwidth for the request on paths that connect the source vertex s_i to the target vertex t_i. There are several common ways by which this can be done. The setting in which each request should be served via a single path is referred to as *unsplittable routing*. A less restrictive setting in which each request can be served via multiple routes is called *splittable routing*. We associate each request with a set of allowed paths (routes) $\mathbb{P}(r_i)$, capturing the *fixed routes* model, in which requests can only be served via a *unique* given path, as a special case. Let $b(r_i)$ be the sum of all bandwidth allocations assigned to request r_i on all paths $P \in \mathbb{P}(r_i)$. The *total bandwidth* of a routing solution is the total bandwidth allocated to all the requests. A *feasible* routing solution is an allocation of bandwidth to requests that does not violate any of the edge capacities. When the routing solution is infeasible, the *load* on an edge is the total bandwidth allocated to it divided by its capacity. The load of a routing solution is the maximum load taken over all edges.

An important parameter that is used in our analysis is U, which is defined to be the minimum value by which the capacities in the network need to be multiplied so as to obtain a feasible splittable solution that routes all requests. When routes are fixed, U reduces to the maximum, taken over all edges, of the number of routes that pass through

[1]The results in this chapter can be extended, with obvious limitations, to handle scenarios in which requests have different bandwidth demands.

an edge divided by its capacity.

One issue separating routing models is whether requests have to be fully served or not. *All-or-nothing* routing means that a request has to be allocated a total bandwidth (splittable or unsplittable) of one unit. Other models relax this requirement and allow the routing algorithm to allocate requests less than one unit of bandwidth.

Routing algorithms are designed to achieve several natural goals. One goal is to maximize the *utility* of the network which is the total bandwidth allocated to all requests. In a somewhat dual setting, the routing algorithm is not allowed to reject any of the requests, in which case the goal is to minimize the maximum load. Another important routing goal is *fairness*. An accepted notion of fairness is *max-min* fairness. To define a fair routing solution, we consider the bandwidth allocation to the requests ($b(r_i)$ to request r_i) as a vector in which the entries (allocations) are sorted from small to large. This vector is called a *bandwidth vector*. A max-min fair routing solution is then an allocation of bandwidth to requests which defines a lexicographically maximal bandwidth vector. An intuitive way of viewing a max-min fair solution is that one cannot increase the bandwidth allocation to a request r_i without decreasing the bandwidth allocated to requests that have received at most the bandwidth given to r_i.

An even more general fairness measure studied in the literature as well is the notion of coordinate-wise competitive solution. A routing solution is called γ_c-*coordinate-wise competitive* if for every i, the ith coordinate of the bandwidth vector is at least $1/\gamma_c$ times the ith coordinate in any feasible routing solution. The beauty of this definition is that a γ_c-coordinate-wise competitive routing approximates *all* possible routings. In particular, it approximates the max-min fair routing, as well as the routing solution that maximizes the total bandwidth allocated, achieving, in some sense, a solution which is the "best of all worlds".

Two parameters are of particular interest in routing problems. The first one is the amount of *bandwidth* that the algorithm routes with respect to an optimal routing, and the second one is the maximum *load* on the edges. A (c_1, c_2)-competitive routing algorithm routes at least $1/c_1$ of the maximum possible bandwidth, while guaranteeing that the load on each edge is at most c_2. With this notation in mind we re-examine the first algorithm in Section 4.4.2 and conclude that it is $(3, O(\log n))$-competitive. This algorithm is actually a bicriteria competitive algorithm that routes a constant fraction of the optimal number of requests while incurring a load of $O(\log n)$.

It turns out that getting a uni-criteria competitive algorithm (i.e. an $(1, O(\log n))$-competitive algorithm) is a crucial non-trivial step for getting better routing solutions for many routing goals[2]. In particular, we will show that given such an algorithm it is easy to design an algorithm that achieves fair routing. A simple example shows that such a result is optimal for an online algorithm. In addition, the generic algorithm we design here generates an unsplittable all-or-nothing routing; however, to allow the use of the algorithm in a wide variety of routing models, its performance is compared to a

[2] Note that we can easily transform a (c_1, c_2)-competitive algorithm to a $(c_1 \cdot c_2, 1)$-competitive algorithm by scaling down all allocated bandwidth. However, obtaining a $(1, c_1 \cdot c_2)$-competitive factor is problematic, since requests should be allocated bandwidth of at most 1.

splittable optimal routing which is allowed to allocate to each request (total) bandwidth in the interval $[0, 1]$. It turns out that this stronger performance allows the use of the generic algorithm as a "building block" for the design of online routing solutions for several models and objectives, yielding improved bounds. Here we will only show one such application for getting a fair routing.

10.1 A Generic Routing Algorithm

In this section we design a generic online routing algorithm which is based on the primal-dual approach. The algorithm generates in an online fashion an unsplittable all-or-nothing routing which is $(1, O(\log n))$-competitive with respect to all splittable routings. To achieve these stronger bounds it is not enough to maintain a single primal solution, leading us to maintain simultaneously several primal solutions that will be used throughout to make clever routing decisions. We will use the same primal-dual pair that was used in Section 4.4.2. The primal-dual pair appears in Figure 4.2.

The algorithm decomposes the graph $G = (V, E)$ into graphs $G_0, G_1, \ldots, G_k$. For each j, the vertices of G_j are the same as in G. The edges of G_j are all edges in G having capacity at least m^j. The capacity of each edge in the jth copy, G_j, is then set to $u(e, j) \leftarrow \min\{u(e), m^{j+2}\}$. Let G_k be the last copy of the graph which is non-empty (i.e. the maximum capacity in G, $u(\max) \leq m^k$). The algorithm maintains a primal solution in each copy of the graph. We denote by $x(e, j)$ and $z(r_i, j)$ the primal variables corresponding to the jth copy. Let $u(\min, j)$ be the minimal edge capacity in the jth copy (which is at least m^j).

Routing Algorithm: Initially, $\forall j$: $x(e,j) \leftarrow \frac{u(\min,j)}{m \cdot u(e,j)}$.
When new request $r_i = (s_i, t_i, \mathbb{P}(r_i))$ arrives:

1. Consider all copies of the graph from G_k to G_0. In each copy G_j:
   (a) Let $P(r_i,j) \in \mathbb{P}(r_i,j)$ be the shortest path with respect to $x(e,j)$ and let α be the length of $P(r_i,j)$.
   (b) If $\alpha < 1$:
      i. Route the request on $P(r_i,j)$.
      ii. For each edge e in $P(r_i,j)$:
      $x(e,j) \leftarrow x(e,j)(1 + \frac{1}{u(e,j)})$.
      iii. $z(r_i,j) \leftarrow 1 - \alpha$.
   (c) Else ($\alpha > 1$):
      i. If the total bandwidth routed in this step in G_j is less than $u(\min,j)$, and the current request can be routed in G_j, route the request in an arbitrary feasible path $P \in \mathbb{P}(r_i,j)$.
   (d) If the request is routed - finish.
2. Reject requests that are rejected from all copies.

The analysis of the algorithm is done via the following claims.

Lemma 10.1. *Let N_j be the total number of requests that are introduced to the jth copy. Let M be the maximum total bandwidth of any feasible splittable routing in G_j (out of N_j). Then, the algorithm accepts at least M requests in G_j, and the load on each edge in G_j is $O(\log n)$.*

Proof. First, observe that when the algorithm decides to route a request in Step (1b), the total primal cost maintained in the jth copy increases by $(1-\alpha)+\sum_{e\in P(r_i,j)} x(e,j) = 1$. When a request is rejected from the jth copy, the primal cost in the jth copy does not change. Second, observe that the primal solution maintained in each copy is feasible with respect to the requests introduced to this copy. This follows, since, if the shortest path in $\mathbb{P}(r_i,j)$ is already at least 1, the constraints of the new request are all satisfied. If the shortest path is of length $\alpha < 1$, then the algorithm updates $z(r_i,j)$ to be $1-\alpha$ to make the current new set of constraints feasible. All previous constraints remain feasible. Finally, note that the total initial primal cost in the jth copy is $\sum_{e\in E} u(e,j)\frac{u(\min,j)}{m\cdot u(e,j)} = u(\min,j)$.

Assume to the contrary that the algorithm routes in Step (1b) a total bandwidth $M' < M - u(\min,j)$. This immediately implies that we have a feasible primal solution with cost strictly less than $M - u(\min,j) + u(\min,j) = M$, contradicting the fact that we have a feasible dual solution with profit M (out of N_j). This means that $M - M'$ is at most $u(\min,j)$ and thus, at least $M - M'$ (or zero, if this value is negative) are routed in G_j in Step (1c), proving the first part of the claim. We next prove the second part of the claim. The initial value of each $x(e,j)$ is $\frac{u(\min,j)}{m\cdot u(e,j)}$. Each time a new request is routed on edge e in Step (1b), $x(e,j)$ is multiplied by $1 + 1/u(e,j) \leq 2$. The algorithm never routes requests in Step (1b) on edges with $x(e,j) > 1$, thus, $x(e,j) \leq 2$. Let $\mathbb{R}(e)$ be the set of requests that are routed on an edge e. We get that:

$$
\begin{aligned}
\frac{u(\min,j)}{m \cdot u(e,j)} \exp\left(\frac{\ln 2}{u(e,j)} \sum_{r_i \in \mathbb{R}(e)} 1\right) &= \frac{u(\min)}{m \cdot u(e,j)} \prod_{r_i \in \mathbb{R}(e)} 2^{1/u(e,j)} \\
&\leq \frac{u(\min,j)}{m \cdot u(e,j)} \prod_{r_i \in \mathbb{R}(e)} \left(1 + \frac{1}{u(e,j)}\right) \leq 2
\end{aligned}
$$

Simplifying, the total bandwidth that the algorithm routes on edge e in Step (c) is $u(e,j) \cdot O\left(\log \frac{m \cdot u(e,j)}{u(\min,j)}\right)$. Since $u(e,j) \leq m^2 u(\min,j)$, this expression is equal to $u(e,j)O(\log n)$. The total bandwidth that the algorithm routes on edge e in the jth copy in step (d) is at most $u(\min,j)$, thus completing the proof. $\square$

Theorem 10.2. *The routing algorithm is* $(1, O(\log n))$*-competitive with respect to all splittable routing solutions.*

Proof. Let M be the maximal bandwidth that can be routed splittably in G. We start by proving that the total bandwidth that the algorithm routes is at least M. As we assumed, there exists a feasible solution that routes a total bandwidth M without violating the constraints. Let $P_1^*, P_2^*, \ldots, P_\ell^*$ be the routes that are used by this feasible solution and let $b_1^*, b_2^*, \ldots, b_\ell^*$ be the bandwidth allocated on each of the paths. Note that each request can be served via multiple routes and get a total bandwidth in the interval $[0,1]$.

For each of the ℓ routing paths, let $w_j \leftarrow \min_{e \in P_j^*} u(e)$, i.e., w_j is the capacity of the minimal capacity edge that is used in path P_j^*. We divide the paths into separate groups $M_0, M_1, \ldots, M_k$. Group M_i consists of all paths for which $m^i \leq w_j \leq m^{i+1}$. Let $|M_i|$ be the total bandwidth of all paths in M_i. Note that $\sum_{i=1}^{k} |M_i| = M$. We prove by reverse induction that the total bandwidth allocated by the algorithm in levels G_j to G_k is at least $|M_j| + |M_{j+1}| + \ldots, +|M_k|$. By this claim the total bandwidth that the algorithm allocates in levels G_0 to G_k is at least M, proving the first part of the theorem.
Induction Base: For $j = k$ the graph G_k consists of edges with capacity at least m^k. In this graph the capacities of all the edges are the same as their capacities in the graph G. M_k consists of paths such that the minimal capacity of an edge on P_j^* is at least m^k. Thus, all the paths in M_k exist in the graph G_k. By Lemma 10.1, the algorithm routes a total bandwidth of at least $|M_k|$ out of N (all the requests).
Induction Step: Let G_j be any level $j < k$. Let $M'_{j+1}, M'_{j+2}, \ldots, M'_k$ be the groups of requests that were routed by the algorithm in $G_{j+1}, \ldots, G_k$. By the induction hypothesis: $|M'_{j+1}| + |M'_{j+2}| + \ldots + |M'_k| \geq |M_{j+1}| + |M_{j+2}| + \ldots + |M_k|$.

We consider the set of paths S in $(M_j \cup M_{j+1} \cup \ldots \cup M_k)$ that do not belong to requests that are routed by the routing algorithm in G_k to G_{j+1}. Let $|S|$ be the total bandwidth allocated by the feasible solution on these paths. These paths all belong to requests that are presented to the graph G_j (i.e. counted as part of N_j). We claim that it is possible to route in G_j a total bandwidth of at least $\min\{|M_j|, |S|\}$ out of the requests that are given to the algorithm on level j. In order to prove this, we prove that if we take any part of the total flow of the paths in S with total bandwidth of at most $|M_j|$, it is possible to route this flow on the graph G_j without violating the capacities.

Set S consists of paths in groups of at least M_j, thus each path $P_j^* \in S$ contains only edges with capacity at least m^j. G_j contains all the edges whose capacity is at least m^j, so the path P_j^* exists in the graph G_j. Each path in M_j also contains an edge with capacity at most m^{j+1}. This means that the total bandwidth allocated by the feasible solution on all paths that belong to M_j is at most m^{j+2}. The capacity of the edges in G_j is restricted to $\min\{u(e), m^{j+2}\}$. Thus, if we take part of the total flow of the paths in S with a total bandwidth of at most $|M_j| \le m^{j+2}$, it is possible to route this flow on the graph G_j without violating the capacities.

By the above claim and Lemma 10.1, the total bandwidth of requests that are routed in G_j is at least $\min\{|M_j|, |S|\}$. If $|S| \ge |M_j|$ then a total bandwidth of at least $|M_j|$ will be routed in G_j and since $|M'_{j+1}| + |M'_{j+2}| + \ldots + |M'_k| \ge |M_{j+1}| + |M_{j+2}| + \ldots + |M_k|$ the induction hypothesis holds. If $|S| < |M_j|$ then at least $|S| \ge (|M_j| + |M_{j+1}| + \ldots + |M_k|) - (|M'_{j+1}| + |M'_{j+2}| + \ldots + |M'_k|)$ will be routed in G_j and the induction hypothesis holds again.

To prove the second part of the theorem consider an edge e with capacity $m^j \le u(e) < m^{j+1}$. Its capacity in levels $\ell > j$ is zero. Therefore, no requests are routed through e in G_ℓ for $\ell > j$. The capacity of e in levels $j, j-1, j-2$ is $u(e)$ and in levels $j-3, j-2, \ldots 0$ the edge capacity drops to $m^{j-1}, m^{j-2}, \ldots m^2$. Thus, the total capacity of the copies of the edge e in all levels is at most four times its capacity. In each level G_j the ratio between the maximal and the minimal edge is at most m^2. Thus, the total bandwidth of requests that are routed on edge e in each level is at most $u(e,j)O(\log n)$. Therefore, the total number of requests routed on the edge in all levels is at most $O(\log n)$ times the sum of capacities of the edge in all levels, which remains $O(\log n)$ times the capacity of edge e in G. $\square$

10.2 Achieving Coordinate-wise Competitive Allocation

In this Section we show application of the generic algorithm so as to achieve a fair allocation. We design an almost optimal online algorithm for achieving a coordinate-wise routing solution. In this setting the algorithm should output an unsplittable routing and assign bandwidth $b \in [0,1]$ to each request. We design an $O(\frac{1}{\epsilon} \log n \log U (\log\log U)^{1+\epsilon})$-competitive algorithm for any $\epsilon > 0$ and prove an almost matching lower bound of $\Omega(\log n \log U + \log U \log\log U)$ even when splittable routing is allowed. The algorithm is quite simple. It considers copies of the graph referred to as levels. In levels $\ell = 0, 1, 2, \ldots$ we multiply all edge capacities by 2^ℓ.

Algorithm: When a request $r_i = (s_i, t_i, \mathbb{P}(r_i))$ arrives:

1. Run the routing algorithm on levels $\ell = 0, 1, 2, \ldots$ in an increasing order.
2. Route the request in the lowest level ℓ in which the routing algorithm accepts the request.
3. Assign bandwidth of $\frac{\epsilon}{c \log n 2^{\ell}(1+\ell)(\mathbb{H}(1+\ell))^{1+\epsilon}}$ to the request, where c is a constant and $\mathbb{H}(\cdot)$ is the harmonic number.

Theorem 10.3. *The algorithm is* $O(\frac{1}{\epsilon} \log n \log U (\log \log U)^{1+\epsilon})$*-coordinate-wise competitive.*

Proof. We first prove a useful lemma that will help us in the analysis:

Lemma 10.4. *If it is possible to route M (out of N) requests in a splittable way when multiplying the capacities of the edges in G by 2^k, then the algorithm:*

- *Routes at least M requests using levels 1 to k.*
- *The total number of requests that are routed on edge e in level k is at most $2^k u(e) O(\log n)$.*

Proof. Let M' be the group of requests that were routed in levels 1 to $k-1$. If $|M'| \geq |M|$, then we are done. Otherwise, it is possible to route in level k a total bandwidth of at least $|M| - |M'|$ out of the requests that were rejected by levels 1 to $k-1$. Thus, by Theorem 10.2, the algorithm routes at least $|M| - |M'|$ requests in level k, and we are done. The second claim is immediate from Theorem 10.2. $\square$

To prove that the algorithm is γ_c-coordinate-wise competitive, we need to show that any coordinate i in the bandwidth vector of the solution we generate is at least $1/\gamma_c$ of this coordinate in any other feasible solution. To prove this, we show that if there exists a solution that assigns the ith coordinate in the bandwidth vector (i.e. the ith "poorest" request) bandwidth b, then our algorithm assigns to at least $N-i+1$ requests bandwidth $b \cdot \Omega(\frac{\epsilon}{\log m \log U (\log \log U)^{1+\epsilon}})$. Assume there exists a feasible solution that assigns bandwidth b to the ith coordinate in the bandwidth vector, then it must assign bandwidth of at least b to at least $N-i+1$ coordinates. This means that there exists a feasible solution that routes at least $N-i+1$ requests when we multiply the capacities in the graph G by $1/b$. Thus, by Lemma 10.4, at least $N-i+1$ requests will be routed in the first k levels where $2^k \leq 2/b$. All these requests are, therefore, assigned bandwidth of at least $\frac{b \cdot \epsilon}{2c \log m (1+k)(\mathbb{H}(1+k))^{1+\epsilon}} = b \cdot \Omega(\frac{\epsilon}{\log m \log U (\log \log U)^{1+\epsilon}})$. The last equation follows since by Lemma 10.4 and the definition of U all the requests will be routed until level k, such that $2^k \leq 2U$.

It is left to prove that the online algorithm does not violate the capacity of any of the edges. By Lemma 10.4, the number of requests routed on an edge e in level k is at most $2^k u(e) O(\log n)$. Thus, when we choose a large enough constant c, the total bandwidth assigned to each edge e is at most:

$$\sum_{j=0}^{\infty} 2^j u(e) O(\log n) \frac{\epsilon}{c \log n 2^j (1+j)(\mathbb{H}(1+j))^{1+\epsilon}} \leq \frac{u(e)}{c'} \sum_{j=1}^{\infty} \frac{\epsilon}{j(\mathbb{H}(j))^{1+\epsilon}} \leq u(e).$$

□

Lower bounds: In [58] a lower bound of (approximately) $\Omega(\log n + \log U \log\log U)$ was proved. We improve the lower bound and prove an almost matching lower bound for the problem.

Lemma 10.5. *Any deterministic algorithm (splittable or unsplittable) is* $\Omega(\log n \log U)$*-coordinate-wise competitive.*

Proof. Let $G = (V, E)$ be a directed line with n nodes. The first U requests can be routed either from node 1 towards node $n/2$ or from node $n/2 + 1$ towards node n. In order to be able to produce such requests, the graph contains an additional source node that is connected to nodes 1 and node $n/2 + 1$. In addition, the graph has a sink node that has directed edges incoming from nodes $n/2$ and n.

After the first U requests the adversary continues to produce requests either in the left half of the line, or the right half, depending on which half contains more than half of the bandwidth. For instance, if it is the left half, then the adversary introduces U new requests that can be routed either from node 1 to node $n/4$ or from node $n/4+1$ to node $n/2$. The adversary continues on with this strategy $\log n$ rounds. We note that in order to be able to produce such a sequence of requests we only add $O(n)$ new source/target vertices to the graph.

Next, consider an optimal solution that routes at most U requests on each edge. With this choice of paths there is a feasible bandwidth allocation that allocates $\log n$ requests bandwidth 1. There is also a feasible solution that allocates bandwidth 1/2 to $2 \log n$ requests. In general, there is a feasible solution that allocates $i \log n$ requests bandwidth of value $1/i$, where $1 \leq i \leq U$. This means that an algorithm that is γ_c coordinate-wise competitive must allocate at least $\log n$ requests bandwidth of more than $1/\gamma_c$. In general, for any $1 \leq i \leq U$, the algorithm must give at least $i \log n$ requests bandwidth of at least $1/i\gamma_c$. By this observation, the total bandwidth allocated by any γ_c coordinate-wise online algorithm must be at least $\sum_{i=1}^{U} \frac{\log n}{i\gamma_c} \geq \frac{\log n \log U}{\gamma_c}$. By the adversary's strategy, there exists an edge such that at least half of the total bandwidth is routed on that edge. Since the total bandwidth allocated to this edge is at most 1, we get that $\gamma_c \geq \frac{\log n \log U}{2} = \Omega(\log n \log U)$. □

10.3 Notes

The results in this chapter are based on the work of Buchbinder and Naor [32]. In this work they designed, using the generic algorithm, several routing algorithms that achieve certain routing goals in several models. They also studied other models such as the fixed routes model and another model that allows the algorithm to allocate weights to requests instead of actual bandwidths (see [33]).

Routing algorithms have been studied extensively. In [8, 11] two different (but similar in spirit) online routing algorithms were suggested. The objective in [11] is maximizing the total throughput, while the algorithm in [8] minimizes the load. Both these algorithms can be viewed within the primal-dual framework (see [33]).

The notion of *all-or-nothing* routing was defined in [38]. The elegant notion of *max-min fairness* was considered in many settings [23, 66]. The general framework of prefix and coordinate-wise competitiveness was suggested in [72]. More properties of these measures (in the offline case) were studied later on in [75, 56]. For example, in [75] they proved that any fixed route instance has an allocation that is $O(\log U)$-coordinate-wise competitive. Goel et al. [58] studied the problem of achieving coordinate-wise competitiveness online. They designed an algorithm which is $O(\frac{1}{\epsilon}\log^2 n(\log U)^{1+\epsilon})$-coordinate-wise competitive for any $\epsilon > 0$. In a relaxed setting where the algorithm is allowed to assign weights instead of allocating bandwidth directly, [58] designed an algorithm which is $O(\log^2 n \log U)$-coordinate-wise competitive.

Chapter 11

Maximizing Ad-auctions revenue

Maximizing the revenue of a seller in an auction has received much attention recently and studied in many models and settings. In particular, the way search engine companies such as MSN, Google and Yahoo! maximize their revenue out of selling ad-auctions has been studied extensively. In the search engine environment, advertisers link their ads to (search) keywords and provide a bid on the amount paid each time a user clicks on their ad. When users send queries to search engines, along with the (algorithmic) search results returned for each query, the search engine displays funded ads corresponding to *ad-auctions*. The ads are instantly sold, or allocated, to interested advertisers (*buyers*). The total revenue out of this fast growing market is currently billions of dollars. Thus, algorithmic ideas that can improve the allocation of the ads, even by a small percentage, are crucial.

The ad-auctions problem is modeled as a generalization of online bipartite matching. There is a set I of n buyers, each buyer i ($1 \leq i \leq n$) has a known daily budget of $B(i)$. We consider an online setting in which m products arrive one-by-one in an online fashion. Let M denote the set of all the products. Upon arrival of a product j, each buyer provides a bid $b(i,j)$ for buying item j. The online algorithm can *allocate* (or *sell*) the product to any one of the buyers. We distinguish between *integral* and *fractional* allocations. In an integral allocation, a product can only be allocated to a single buyer. In a fractional allocation, products can be fractionally allocated to several buyers, however, for each product, the sum of the fractions allocated to buyers cannot exceed 1. The revenue received from each buyer is defined to be the minimum between the sum of the costs of the products allocated to a buyer (times the fraction allocated) and the total budget of the buyer. That is, buyers can never be charged by more than their total budget. The objective is to maximize the total revenue of the seller. Let $R_{\max} = \max_{i \in I, j \in M}\{\frac{b(i,j)}{B(i)}\}$ be the maximum ratio between a bid of any buyer and its total budget.

A linear programming formulation of the fractional (offline) ad-auctions problem appears in Figure 11.1. Let $y(i,j)$ denote the fraction of product j allocated to buyer i. The objective function is maximizing the total revenue. The first set of constraints guarantees that the sum of the fractions of each product is at most 1. The second set

Dual (Packing)	Primal (Covering)
Maximize: $\sum_{j=1}^{m}\sum_{i=1}^{n} b(i,j)y(i,j)$	Minimize : $\sum_{i=1}^{n} B(i)x(i) + \sum_{j=1}^{m} z(j)$
Subject to:	Subject to:
For each $1 \le j \le m$: $\sum_{i=1}^{n} y(i,j) \le 1$	For each (i,j): $b(i,j)x(i) + z(j) \ge b(i,j)$
For each $1 \le i \le n$: $\sum_{j=1}^{m} b(i,j)y(i,j) \le B(i)$	For each i,j: $x(i), z(j) \ge 0$
For each i,j: $y(i,j) \ge 0$	

Figure 11.1: The fractional ad-auctions problem (the dual) and the corresponding primal problem

of constraints guarantees that each buyer does not spend more than its budget. In the primal problem there is a variable $x(i)$ for each buyer i and a variable $z(j)$ for each product j. For all pairs (i,j) the constraint $b(i,j)x(i) + z(j) \ge b(i,j)$ needs to be satisfied.

11.1 The Basic Algorithm

The basic algorithm for the online ad-auctions produces primal and dual solutions to the linear programs in Figure 11.1.

Allocation Algorithm: Initially $\forall i \;\; x(i) \leftarrow 0$.
Upon arrival of a new product j allocate the product to the buyer i that maximizes $b(i,j)(1-x(i))$. If $x(i) \ge 1$ then do nothing. Otherwise:

1. Charge the buyer the minimum between $b(i,j)$ and its remaining budget and set $y(i,j) \leftarrow 1$
2. $z(j) \leftarrow b(i,j)(1-x(i))$
3. $x(i) \leftarrow x(i)\left(1 + \frac{b(i,j)}{B(i)}\right) + \frac{b(i,j)}{(c-1)\cdot B(i)}$ (c is determined later).

Theorem 11.1. *The allocation algorithm is* $(1-1/c)(1-R_{\max})$*-competitive, where* $c = (1+R_{\max})^{\frac{1}{R_{\max}}}$. *When* $R_{\max} \to 0$ *the competitive ratio tends to* $(1-1/e)$.

Proof. Let P and D be the values of the primal and dual solution during the run of the algorithm. We prove three simple claims:

1. The algorithm produces a primal feasible solution.
2. In each iteration $\Delta P \le (1 + \frac{1}{c-1}) \cdot \Delta D$, where ΔP and ΔD are the changes in the values of the primal and dual objective functions.
3. The algorithm produces an almost feasible dual solution.

Proof of (1)**:** Consider a primal constraint corresponding to buyer i and product j. If $x(i) \ge 1$ then the primal constraint is satisfied. Otherwise, the algorithm allocates the product to the buyer i' for which $b(i',j)(1-x(i'))$ is maximized. Setting $z(j) = b(i',j)(1-x(i'))$ guarantees that the constraint is satisfied for all (i,j). Subsequent increases of the variables $x(i)$'s cannot make the solution infeasible.

Proof of (2)**:** Whenever the algorithm updates the primal and dual solutions, the change in the dual profit is $b(i,j)$. (Note that even if the remaining budget of buyer i, to which product j is allocated, is less than its bid $b(i,j)$, variable $y(i,j)$ is still set to 1.) The change in the primal cost is:

$$B(i)\Delta x(i) + z(j) = \left(b(i,j)x(i) + \frac{b(i,j)}{c-1}\right) + b(i,j)(1-x(i)) = b(i,j)\left(1+\frac{1}{c-1}\right).$$

Proof of (3)**:** The algorithm never updates the dual solution for buyers satisfying $x(i) \geq 1$. We prove that for any buyer i, when $\sum_{j\in M} b(i,j)y(i,j) \geq B(i)$, then $x(i) \geq 1$. This is done by proving that:

$$x(i) \geq \frac{1}{c-1}\left(c^{\frac{\sum_{j\in M} b(i,j)y(i,j)}{B(i)}} - 1\right). \tag{11.1}$$

Thus, whenever $\sum_{j\in M} b(i,j)y(i,j) \geq B(i)$, we get that $x(i) \geq 1$. We prove (11.1) by induction on the (relevant) iterations of the algorithm. Initially, this assumption is trivially true. We are only concerned with iterations in which a product, say k, is sold to buyer i. In such an iteration we get that:

$$\begin{aligned}
x(i)_{\text{end}} &= x(i)_{\text{start}} \cdot \left(1+\frac{b(i,k)}{B(i)}\right) + \frac{b(i,k)}{(c-1)\cdot B(i)} \\
&\geq \frac{1}{c-1}\left[c^{\frac{\sum_{j\in M\setminus\{k\}} b(i,j)y(i,j)}{B(i)}} - 1\right]\cdot\left(1+\frac{b(i,k)}{B(i)}\right) + \frac{b(i,k)}{(c-1)\cdot B(i)} \qquad (11.2)\\
&= \frac{1}{c-1}\left[c^{\frac{\sum_{j\in M\setminus\{k\}} b(i,j)y(i,j)}{B(i)}} \cdot \left(1+\frac{b(i,k)}{B(i)}\right) - 1\right] \\
&\geq \frac{1}{c-1}\left[c^{\frac{\sum_{j\in M\setminus\{k\}} b(i,j)y(i,j)}{B(i)}} \cdot c^{\left(\frac{b(i,k)}{B(i)}\right)} - 1\right] = \frac{1}{c-1}\left[c^{\frac{\sum_{j\in M} b(i,j)y(i,j)}{B(i)}}. - 1\right] \qquad (11.3)
\end{aligned}$$

Inequality (11.2) follows from the induction hypothesis, and Inequality (11.3) follows since, for any $0 \leq x \leq y \leq 1$, $\frac{\ln(1+x)}{x} \geq \frac{\ln(1+y)}{y}$. Note that when $\frac{b(i,k)}{B(i)} = R_{\max}$ then Inequality 11.3 holds with equality. This is the reason why we chose the value c to be $(1+R_{\max})^{\frac{1}{R_{\max}}}$. Thus, it follows that whenever the sum of charges to a buyer exceeds the budget, we stop charging this buyer. Hence, there can be at most one iteration in which a buyer is charged by less than $b(i,j)$. Therefore, for each buyer i: $\sum_{j\in M} b(i,j)y(i,j) \leq B(i) + \max_{j\in M}\{b(i,j)\}$, and thus the profit extracted from buyer i is at least:

$$\left[\sum_{j\in M} b(i,j)y(i,j)\right]\frac{B(i)}{B(i)+\max_{j\in M}\{b(i,j)\}} \geq \left[\sum_{j\in M} b(i,j)y(i,j)\right](1-R_{\max}).$$

Dual (Packing)	
Maximize:	$\sum_{j=1}^{m}\sum_{i=1}^{n}\sum_{\ell=1}^{k} b(i,j,\ell)y(i,j,\ell)$
Subject to:	
$\forall 1 \le j \le m,\ 1 \le k \le \ell$:	$\sum_{i=1}^{n} y(i,j,k) \le 1$
$\forall 1 \le i \le n$:	$\sum_{j=1}^{m}\sum_{k=1}^{\ell} b(i,j,k)y(i,j,k) \le B(i)$
$\forall 1 \le j \le m, 1 \le i \le n$:	$\sum_{k=1}^{\ell} y(i,j,k) \le 1$

Primal (Covering)	
Minimize :	$\sum_{i=1}^{n} B(i)x(i) + \sum_{j=1}^{m}\sum_{k=1}^{\ell} z(j,k) + \sum_{i=1}^{n}\sum_{j=1}^{m} s(i,j)$
Subject to:	
$\forall i,j,k$:	$b(i,j,k)x(i) + z(j,k) + s(i,j) \ge b(i,j,k)$

Figure 11.2: The fractional multi-slot problem (the dual) and the corresponding primal problem

By the second claim the dual value it at least $1 - 1/c$ times the primal value, and thus (by weak duality) we conclude that the competitive ratio of the algorithm is $(1 - 1/c)\,(1 - R_{\max})$. $\square$

11.2 Multiple Slots

In this section we show how to extend the algorithm in a very elegant way to sell different advertisement slots in each round. Suppose there are ℓ slots to which ad-auctions can be allocated and suppose that buyers are allowed to provide bids on keywords which are slot dependent. Denote the bid of buyer i on keyword j and slot k by $b(i,j,k)$. The restriction is that an (integral) allocation of a keyword to two different slots cannot be sold to the same buyer. The linear programming formulation of the problem is in Figure 11.2. Note that the algorithm does not update the variables $z(\cdot)$ and $s(\cdot)$ explicitly. These variables are only used for the purpose of analysis. The algorithm for the online ad-auctions problem is as follows.

Dual (Packing)		Primal (Covering)	
Maximize: $\sum_{i=1}^{n}\sum_{k=1}^{\ell} b(i,j,k)\,(1-x(i))\,y(i,j,k)$		Minimize : $\sum_{i=1}^{n} s(i,j)+\sum_{k=1}^{\ell} z(j,k)$	
Subject to:		Subject to:	
$\forall 1\le k\le \ell$:	$\sum_{i=1}^{n} y(i,j,k)\le 1$	$\forall(i,k)$:	$s(i,j)+z(j,k)\ge b(i,j,k)\,(1-x(i))$
$\forall 1\le i\le n$:	$\sum_{k=1}^{\ell} y(i,j,k)\le 1$	$\forall i,k$:	$s(i,j),z(j,k)\ge 0$
$\forall i,k$:	$y(i,j,k)\ge 0$		

Figure 11.3: The matching problem solved for product j. Here $x(i)$, $1\le i\le n$, is a constant.

Allocation Algorithm: Initially, $\forall i,\quad x(i)\leftarrow 0$. Upon arrival of a new product j:

1. Generate a bipartite graph H: n buyers on one side and ℓ slots on the other side. Edge $(i,k)\in H$ has weight $b(i,j,k)(1-x(i))$.
2. Find a maximum weight (integral) matching in H, i.e., an assignment to the variables $y(i,j,k)$.
3. Charge buyer i the minimum between $\sum_{k=1}^{\ell} b(i,j,k)y(i,j,k)$ and its remaining budget.
4. For each buyer i, if there exists slot k for which $y(i,j,k)>0$:

$$x(i)\leftarrow x(i)\left(1+\frac{b(i,j,k)y(i,j,k)}{B(i)}\right)+\frac{b(i,j,k)y(i,j,k)}{(c-1)\cdot B(i)}$$

Theorem 11.2. *The algorithm is* $(1-1/c)\,(1-R_{\max})$*-competitive, where* c *tends to* e *when* $R_{\max}\to 0$.

Proof. We prove three simple claims:

1. The algorithm produces a primal feasible solution.
2. In each iteration, $\Delta P\le\left(1+\frac{1}{c-1}\right)\cdot\Delta D$.
3. The algorithm produces an almost feasible dual solution.

To prove the claims, we crucially use the fact that a maximum weight (integral) matching in H can be computed via a primal-dual algorithm. The primal and dual matching programs are in Figure 11.3. The algorithm outputs an optimal primal and dual solutions satisfying:

$$\sum_{i=1}^{n}\sum_{k=1}^{\ell} b(i,j,k)\,(1-x(i))\,y(i,j,k)=\sum_{i=1}^{n} s(i,j)+\sum_{k=1}^{\ell} z(j,k).$$

Proof of (1)**:** Recall that the primal constraint in the linear program of the multiple slot problem (see Figure 11.2) is:

$$\forall i,j,k:\quad b(i,j,k)x(i)+z(j,k)+s(i,j)\ge b(i,j,k).$$

Since $z(j,k) + s(i,j) \geq b(i,j,k)\left((1 - x(i)\right)$, the above constraint is satisfied.

Proof of (2)**:** When the jth product arrives,

$$\begin{aligned}
\Delta P &= \sum_{i=1}^{n} z(j,i) + \sum_{k=1}^{\ell} s(j,i) + \sum_{i=1}^{n} B(i)\Delta x(i) \\
&= \sum_{i=1}^{n}\sum_{k=1}^{\ell} b(i,j,k)\left(1 - x(i)\right) y(i,j,k) \\
&\quad + \sum_{i=1}^{n}\sum_{k=1}^{\ell} B(i)\left(\frac{b(i,j,k)x(i)y(i,j,k)}{B(i)} + \frac{b(i,j,k)y(i,j,k)}{(c-1)\cdot B(i)}\right) \\
&= \sum_{i=1}^{n}\sum_{k=1}^{\ell} b(i,j,k)y(i,j,k)\left(1 + \frac{1}{c-1}\right).
\end{aligned}$$

Since $\Delta D = \sum_{i=1}^{n}\sum_{k=1}^{\ell} b(i,j,k)y(i,j,k)$, the claim follows.

Proof of (3)**:** The algorithm never updates the dual solution for buyers satisfying $x(i) \geq 1$. We prove that for any buyer i, when $\sum_{j=1}^{m}\sum_{k=1}^{\ell} b(i,j,k)y(i,j,k) \geq B(i)$, then $x(i) \geq 1$. This is done by showing that

$$x(i) \geq \frac{1}{c-1}\left(c^{\frac{\sum_{j=1}^{m}\sum_{k=1}^{\ell} b(i,j,k)y(i,j,k)}{B(i)}} - 1\right). \tag{11.4}$$

Thus, whenever $\sum_{j=1}^{m}\sum_{k=1}^{\ell} b(i,j,k)y(i,j,k) \geq B(i)$, we get that $x(i) \geq 1$. We prove (11.4) by induction on the (relevant) iterations of the algorithm. Initially, this assumption is trivially true. We are only concerned about iterations in which the kth slot of product t is sold to buyer i. In such an iteration we get that:

$$\begin{aligned}
x(i)_{\text{end}} &= x(i)_{\text{start}} \cdot \left(1 + \frac{b(i,t,k)}{B(i)}\right) + \frac{b(i,t,k)}{(c-1)\cdot B(i)} \\
&\geq \frac{1}{c-1}\left[c^{\frac{\sum_{j\in M\setminus\{t\}}\sum_{k=1}^{\ell} b(i,j,k)y(i,j,k)}{B(i)}} - 1\right]\cdot\left(1 + \frac{b(i,t,k)}{B(i)}\right) \\
&\quad + \frac{b(i,t,k)}{(c-1)\cdot B(i)} \qquad (11.5) \\
&= \frac{1}{c-1}\left[c^{\frac{\sum_{j\in M\setminus\{t\}}\sum_{k=1}^{\ell} b(i,j,k)y(i,j,k)}{B(i)}}\cdot\left(1 + \frac{b(i,t,k)}{B(i)}\right) - 1\right] \\
&\geq \frac{1}{c-1}\left[c^{\frac{\sum_{j\in J\setminus\{t\}}\sum_{k=1}^{\ell} b(i,j,k)y(i,j,k)}{B(i)}}\cdot c^{\left(\frac{b(i,t,k)}{B(i)}\right)} - 1\right] \qquad (11.6) \\
&= \frac{1}{c-1}\left[c^{\frac{\sum_{j\in M}\sum_{k=1}^{\ell} b(i,j,k)y(i,j,k)}{B(i)}} - 1\right].
\end{aligned}$$

Inequality (11.5) follows from the induction hypothesis and Inequality (11.6) follows since, for any $0 \leq x \leq y \leq 1$, $\frac{\ln(1+x)}{x} \geq \frac{\ln(1+y)}{y}$.

By the above, it follows that whenever the sum of the charges to a buyer is more than its budget, we stop charging this buyer. Thus, there can be at most one iteration in which we charge the buyer by less than $b(i,j,k)$. Therefore, for each buyer i: $\sum_{j\in M}\sum_{k=1}^{\ell} b(i,j,k)y(i,j,k) \leq B(i)+\max_{j\in M,k}\{b(i,j,k)\}$, and thus the profit extracted from buyer i is at least:

$$\left[\sum_{j\in M} b(i,j,k)y(i,j,k)\right]\frac{B(i)}{B(i)+\max_{j\in M,k}\{b(i,j,k)\}} \geq \left[\sum_{j\in M}\sum_{k=1}^{\ell} b(i,j,k)y(i,j,k)\right](1-R_{\max}).$$

By the second claim the profit of the dual it at least $1-1/c$ times the cost of the primal, and thus, by weak duality theorem we conclude that the competitive ratio of the algorithm is $(1-1/c)\,(1-R_{\max})$. □

11.3 Incorporating Stochastic information

In this Section we improve the worst case competitive ratio when additional stochastic information is available. We assume that stochastically or from historical experience we know that a bidder i is likely to spend a good fraction of her budget. We want to tweak the algorithm so that the algorithm's worst case performance improves. As we tweak the algorithm it is likely that the bidder may spend more or less fraction of his budget. So we propose to tweak the algorithm gradually until some steady state is reached, i.e., no more tweaking is required. Suppose at the steady state buyer i is likely to spend a good fraction of his budget. Let $0 \leq g_i \leq 1$ be a lower bound on the fraction of the budget buyer i is going to spend. We show that having this additional information allows us to improve the worst case competitive ratio to $1-\frac{1-g}{e^{1-g}}$, where $g=\min_{i\in I}\{g_i\}$ is the minimal fraction of budget extracted from a buyer.

The main idea behind the algorithm is that if a buyer is known to have spent at least g_i fraction of his budget, then it means that the primal variable $x(i)$ will be large at the end. Thus, the value of $z(j)$ can be made smaller. This, in turn, gives us additional "money" that can be used to increase $x(i)$ faster. The main issue is to determine the value of $x(i)$ once the buyer has spent g_i fraction of his budget. This value is denoted by $x_s(i)$ and we choose it so that after the buyer has spent g_i fraction of its budget, $x(i)=x_s(i)$, and after having extracting all of its budget, $x(i)=1$. In addition, we need the change in the primal cost to be the same with respect to the dual profit in iterations where we sell the product to a buyer i who has not yet spent the threshold of g_i of his budget. The optimal choice of $x_s(i)$ turns out to be $\frac{g_i}{c^{1-g_i}-(1-g_i)}$, and the growth function of the primal variable $x(i)$, as a function of the fraction of the budget spent, should be linear until the buyer has spent a g_i fraction of his budget, and exponential from that point on. The modified algorithm is the following:

Allocation Algorithm: Initially $\forall i \;\; x(i) \leftarrow 0$. Upon arrival of a new product j Allocate the product to the buyer i that maximizes $b(i,j)(1 - \max\{x(i), x_s(i)\})$, where $x_s(i) = \frac{g_i}{c^{1-g_i} - (1-g_i)}$. If $x(i) \geq 1$ then do nothing. Otherwise:

1. Charge the buyer the minimum between $b(i,j)$ and its remaining budget
2. $z(j) \leftarrow b(i,j)(1 - \max\{x(i), x_s(i)\})$
3. $x(i) \leftarrow x(i) + \max\{x(i), x_s(i)\}\frac{b(i,j)}{B(i)} + \frac{b(i,j)}{B(i)}\frac{1-g_i}{c^{1-g_i}-(1-g_i)}$ (c is determined later).

Theorem 11.3. *If each buyer spends at least g_i fraction of its budget, then the algorithm is: $\left(1 - \frac{1-g}{c^{1-g}}\right)(1 - R_{\max})$-competitive, where $c = (1 + R_{\max})^{\frac{1}{R_{\max}}}$.*

Proof. We first prove a more general claim regarding the final value of $x(i)$. During the execution of the algorithm we increase the value of primal variables $x(i)$. For buyer i, let $x(i, \text{end})$ be the final (highest) value of $x(i)$ (upon termination). By our assumption, buyer i extracted at least g_i fraction of its budget. Whenever we charge a buyer i for an item and $x(i) < x_s(i)$, the algorithm updates:

$$x(i) \leftarrow x(i) + \frac{b(i,j)}{B(i)}\left(x_s(i) + \frac{1-g_i}{c^{1-g_i} - (1-g_i)}\right).$$

Thus, the final value of $x(i)$ is:

$$x(i,\text{end}) \geq g_i \cdot \left(x_s(i) + \frac{1-g_i}{c^{1-g_i} - (1-g_i)}\right) = g_i \cdot x_s(i) + (1-g_i)x_s(i) = x_s(i) \tag{11.7}$$

We next prove three simple claims:

- The algorithm produces a primal feasible solution.
- In each iteration, $\Delta P \leq (1 + \frac{1-g}{c^{1-g}-(1-g)}) \cdot \Delta D$.
- The algorithm produces an almost feasible dual solution.

Proof of (1)**:** Consider a primal constraint of buyer i and any item j. In order to make this constraint feasible, we need to set $z(j) \geq \max\{0, b(i,j)(1 - x(i,\text{end}))\}$. By Equation 11.7, $x(i,\text{end}) \geq x_s(i)$. Thus, when item j arrives, setting $z(j)$ to be $b(i,j)(1 - \max\{x(i), x_s(i)\}) \geq b(i,j)(1 - x(i,\text{end}))$ suffices to satisfy the constraint. Since the algorithm chooses the buyer i that maximizes this value, and sets $z(j)$ according to this maximal value, we get that the constraint corresponding to any buyer i and item j is satisfied.

Proof of (2)**:** Whenever the algorithm updates the primal and dual solutions the change in the dual profit is $b(i,j)$. (Note that even if the remaining budget of buyer i to which product j is allocated is less than its bid $b(i,j)$, variable $y(i,j)$ is still set to 1.) The change in the primal cost is:

$$\begin{aligned}
B(i)\Delta x(i) + z(j) &= B(i)\cdot\left(\frac{b(i,j)\max\{x(i),x_s(i)\}}{B(i)} + \frac{b(i,j)}{B(i)}\frac{1-g_i}{c^{1-g_i}-(1-g_i)}\right) \\
&+ b(i,j)(1-\max\{x(i),x_s(i)\}) \\
&= b(i,j)\left(1+\frac{1-g_i}{c^{1-g_i}-(1-g_i)}\right) \le b(i,j)\left(1+\frac{1-g}{c^{1-g}-(1-g)}\right)
\end{aligned}$$

Proof of (3)**:** The algorithm never updates the dual solution for buyers satisfying $x(i)\ge 1$. We prove that for any buyer i, when $\sum_{j\in M} b(i,j)y(i,j)\ge B(i)$, then $x(i)\ge 1$. This is done by proving that if the buyer i extracted g'_i fraction of its budget (i.e. $\sum_{j\in M} b(i,j)y(i,j) = g'_i\cdot B(i)$) then:

$$x(i) \ge \begin{cases} g'_i\left[x_s(i) + \frac{1-g_i}{c^{1-g_i}-(1-g_i)}\right] & \text{if } g'_i \le g_i \\ x_s(i)c^{g'_i-g_i} + \frac{1-g_i}{c^{1-g_i}-(1-g_i)}\left[c^{g'_i-g_i}-1\right] & \text{if } g'_i > g_i \end{cases} \tag{11.8}$$

It is easy to check that when $g'_i = g_i$, the two are the same and equal to $x_s(i)$. Thus, if the claim is correct, then whenever buyer i extracts all his budget we get that:

$$\begin{aligned}
x(i) &\ge x_s(i)c^{1-g_i} + \frac{1-g_i}{c^{1-g_i}-(1-g_i)}\left[c^{1-g_i}-1\right] \\
&= \frac{g_i}{c^{1-g_i}-(1-g_i)}c^{1-g_i} + \frac{1-g_i}{c^{1-g_i}-(1-g_i)}\left[c^{1-g_i}-1\right] = 1
\end{aligned}$$

We prove Inequality (11.8) by induction on the (relevant) iterations of the algorithm. Initially, this assumption is trivially true. We are only concerned about iterations in which an item, say k, is sold to buyer i. Let g'_i be the fraction of the budget buyer i spent before the current allocation, and let $g''_i = g'_i + \frac{b(i,j)}{B(i)}$ be the fraction of the budget buyer i spends after the current allocation. In iterations in which $x(i) < x_s(i)$, we get by Equality 11.7 that $g'_i < g_i$, and thus:

$$\begin{aligned}
x(i)_{\text{end}} &= x(i)_{\text{start}} + x_s(i)\frac{b(i,k)}{B(i)} + \frac{b(i,k)}{B(i)}\frac{1-g_i}{c^{1-g_i}-(1-g_i)} \\
&\ge g'_i\left[x_s(i) + \frac{1-g_i}{c^{1-g_i}-(1-g_i)}\right] + x_s(i)\frac{b(i,k)}{B(i)} + \frac{b(i,k)}{B(i)}\frac{1-g_i}{c^{1-g_i}-(1-g_i)} \qquad (11.9) \\
&= g''_i\left[x_s(i) + \frac{1-g_i}{c^{1-g_i}-(1-g_i)}\right],
\end{aligned}$$

where Inequality (11.9) follows by the induction hypothesis. We also remark here that if the budget extracted from buyer i before the iteration is less than g_i, and the budget extracted after the iteration is strictly more than g_i, then it is possible to divide the cost of the item $b(i,j)$ into two costs $b(i,j)_1 + b(i,j)_2 = b(i,j)$, such that the budget extracted after virtually selling $b(i,j)_1$ is exactly g_i. We virtually sell both items to buyer i and change $x(i)$ in two iterations. It is easy to verify that the change of $x(i)$ is the same as if this was done in a single iteration.

In iterations in which $x(i) \geq x_s(i)$ we get by Equality (11.7) that $g_i' \geq g_i$ and so:

$$\begin{aligned}
x(i)_{\text{end}} &= x(i)_{\text{start}} \left(1 + \frac{b(i,k)}{B(i)}\right) + \frac{b(i,k)}{B(i)} \frac{1-g_i}{c^{1-g_i} - (1-g_i)} \\
&\geq \left[x_s(i) c^{g_i' - g_i} + \frac{1-g_i}{c^{1-g_i} - (1-g_i)} \left[c^{g_i'-g_i} - 1\right]\right] \left(1 + \frac{b(i,k)}{B(i)}\right) \\
&\quad + \frac{b(i,k)}{B(i)} \frac{1-g_i}{c^{1-g_i} - (1-g_i)} \qquad (11.10) \\
&= x_s(i) c^{g_i'-g_i} \left(1 + \frac{b(i,k)}{B(i)}\right) + \frac{1-g_i}{c^{1-g_i} - (1-g_i)} \left(c^{g_i'-g_i} \left(1 + \frac{b(i,k)}{B(i)}\right) - 1\right) \\
&\geq x_s(i) c^{g_i'-g_i} \cdot c^{\left(\frac{b(i,k)}{B(i)}\right)} + \frac{1-g_i}{c^{1-g_i} - (1-g_i)} \left(c^{g_i'-g_i} \cdot c^{\left(\frac{b(i,k)}{B(i)}\right)} - 1\right) \qquad (11.11) \\
&= x_s(i) c^{g_i''-g_i} + \frac{1-g_i}{c^{1-g_i} - (1-g_i)} \left[c^{g_i''-g_i} - 1\right],
\end{aligned}$$

where Inequality (11.10) follows from the induction hypothesis, and Inequality (11.11) follows since for any $0 \leq x \leq y \leq 1$, $\frac{\ln(1+x)}{x} \geq \frac{\ln(1+y)}{y}$.

By the above, it follows that whenever the sum of the charges to a buyer is more than its budget, we stop charging this buyer. Thus, there can be at most one iteration in which we charge the buyer by less than $b(i,j)$. Therefore, for each buyer i: $\sum_{j \in M} b(i,j) y(i,j) \leq B(i) + \max_{j \in M}\{b(i,j)\}$, and thus the profit extracted from buyer i is at least:

$$\left[\sum_{j \in M} b(i,j) y(i,j)\right] \frac{B(i)}{B(i) + \max_{j \in M}\{b(i,j)\}} \geq \left[\sum_{j \in M} b(i,j) y(i,j)\right] (1 - R_{\max}).$$

By the second claim the profit of the dual it at least $1 - \frac{1-g_i}{c^{1-g_i}} \geq 1 - \frac{1-g}{c^{1-g}}$ times the cost of the primal, and thus, by weak duality theorem we conclude that the competitive ratio of the algorithm is $(1 - R_{\max})(1 - \frac{1-g}{c^{1-g}})$. □

11.4 Notes

The results in this chapter are based on the work of Buchbinder, Jain and Naor [30]. The paper has further extensions and variants of the algorithm to other scenarios. Maximizing the revenue of a seller in both offline and online settings has been studied extensively in many different models, e.g., [80, 7, 78, 27, 26]. Mehta *et al.* [80] also proposed a simple deterministic $(1 - 1/e)$-competitive algorithm. Their analysis uses a new notion of trade-off revealing LP. The work of [80] builds on online bipartite matching [71] and online b-matching [68]. The online b-matching problem is a special case of the online ad-auctions problem in which all buyers have a budget of b dollars, and the bids are either 0 or 1. In [68] a deterministic algorithm is given for b-matching with competitive ratio tending to $(1 - 1/e)$ (from below) as b grows.

The work of Buchbinder, Jain and Naor [30] also includes $(1 - 1/e)$-competitive algorithms for other problems as well. For instance, the description and analysis of the ski rental problem in Chapter 3 is taken from their work. The tight randomized upper bound on the ski rental problem was originally obtained in a non-primal dual approach by Karlin et al. [70].

Chapter 12

Dynamic TCP-Acknowledgement Problem

In this chapter we consider the dynamic TCP acknowledgement problem. The dynamic TCP acknowledgment problem is the following. A stream of packets arrives at a destination from a source. The source needs to get an acknowledgement for each of the packets, however, it is possible to acknowledge several packets by a single acknowledgement message. This can save on communication overhead, but requires delaying the acknowledgement of certain messages (which is in general undesirable). Thus, the objective function is to minimize the number of acknowledgement messages sent along with the sum of latencies of the packets.

Let M be the set of packets. For each packet $j \in M$, let $t(j)$ be the time of arrival at the destination. Assume now that packets can only arrive in discrete times of $\frac{1}{d}$. We later take $d \to \infty$ so this assumption is not limiting. With the time discretization assumption, we can formulate the TCP acknowledgement problem as a covering linear program which appears in Figure 12.1. In this formulation we have a variable x_t for each discrete time t which is set to 1 if the algorithm sends an acknowledgement message at t. For each packet j and time $t \geq t(j)$, we have a variable $z(j,t)$ which is set to 1 if packet j is delayed between time t and time $t + \frac{1}{d}$. By this formulation, our objective is minimizing $\sum_{t \in T} x_t + \sum_{j \in M} \sum_{t | t \geq t(j)} \frac{1}{d} z(j,t)$. For each j and $\{t | t \geq t(j)\}$, we require that $\sum_{k=t(j)}^{t} x_k + z(j,t) \geq 1$. This guarantees that either the packet is delayed between

Dual (Packing)	Primal (Covering)
Maximize: $\sum_{j \in M} \sum_{t \mid t \geq t(j)} y(j,t)$	Minimize : $\sum_{t \in T} x_t + \sum_{j \in M} \sum_{t \mid t \geq t(j)} \frac{1}{d} z(j,t)$
Subject to:	Subject to:
For each $t \in T$: $\sum_{j \mid t \geq t(j)} \sum_{t' \geq t} y(j,t') \leq 1$	For each j, $t \mid t \geq t(j)$: $\quad \sum_{k=t(j)}^{k=t} x_k + z(j,t) \geq 1$
For each j, $t \mid t \geq t(j)$: $\quad y(j,t) \leq \frac{1}{d}$	

Figure 12.1: The fractional TCP problem (the primal) and the corresponding dual problem

time t and time $t + \frac{1}{d}$, or an acknowledgement message was sent to the source since the arrival time of the packet. The dual packing problem has variables $y(j,t)$ for each packet j and $t \geq t(j)$.

12.1 The Algorithm

Based on the covering LP formulation of the dynamic TCP-acknowledgment problem, we design a simple primal-dual based algorithm for the problem. The algorithm is similar in spirit to the online algorithm presented for the ski rental in Chapter 3.

Initially, $\forall k \quad x_k \leftarrow 0$.
At each discrete time t (iteration), consider each of the packets j for which $\sum_{k=t(j)}^{k=t} x_k < 1$.
For each such packet j do the following update:

1. $z(j,t) \leftarrow 1 - \sum_{k=t(j)}^{k=t} x_k$
2. $x_t \leftarrow x_t + \frac{1}{d}\sum_{k=t(j)}^{k=t} x_k + \frac{1}{(c-1)\cdot d}$ (c is determined later).
3. $y(j,t) \leftarrow \frac{1}{d}$.

The analysis is not very difficult: First, the primal solution we produce is feasible. This follows since we update for each unsatisfied packet $z(j,t) \leftarrow 1 - \sum_{k=t(j)}^{k=t} x_k$ in each time t.

The second observation is that for each packet j and time t in which we updated, the change in the dual profit is $\frac{1}{d}$, while the change in our primal cost is:

$$\left(1 - \sum_{k=t(j)}^{k=t} x_k\right)\frac{1}{d} + \frac{1}{d}\left(\sum_{k=t(j)}^{k=t} x_k + \frac{1}{c-1}\right) = \frac{1}{d}\left(1 + \frac{1}{c-1}\right).$$

Finally, we want to choose the parameter c such that the dual solution we produce is feasible. Consider a time t and a corresponding dual constraint $\sum_{j \mid t \geq t(j)} \sum_{t' \geq t} y(j,t') \leq 1$. We want to guarantee that after d updates of $y(j,t')$ that "belongs" to the constraint, all packets that have arrived prior to t are satisfied, and therefore there are no more updates of $y(j,t')$ belonging to the constraint. We prove that after d such updates, $\sum_{k \geq t} x_k \geq 1$, and so all packets that have arrived until time t are satisfied.

We prove by induction on the updates that $\sum_{k \geq t} x_k \geq \frac{(1+1/d)^q - 1}{c-1}$, where q is the number of updates. Before the first update, the claim trivially holds. Consider an update of $y(j,t')$ (at time t') such that $t' \geq t$ and packet j arrived at time $\leq t$. By the algorithm we get that:

$$x_{t'} \leftarrow x_{t'} + \frac{1}{d}\sum_{k=t(j)}^{k=t'} x_k + \frac{1}{(c-1)\cdot d} \geq x_{t'} + \frac{1}{d}\sum_{k=t}^{k=t'} x_k + \frac{1}{(c-1)\cdot d}.$$

Therefore, $\sum_{k \geq t} x_k$ satisfies:

$$(1+1/d)\sum_{k \geq t} x_k + \frac{1}{(c-1)\cdot d} \geq (1+1/d)\,\frac{(1+1/d)^{q-1}-1}{c-1} + \frac{1}{(c-1)\cdot d} = \frac{(1+1/d)^q - 1}{c-1},$$

where the inequality follows by the induction hypothesis. Thus, choosing $c = (1+1/d)^d$ suffices, and when $d \to \infty$ we get a $(1-1/e)$ competitive algorithm.

In order to get a randomized integral solution we arrange the variables x_t on the infinite line. We choose a random number $p \in_R [0,1]$. We then send an acknowledgement message at each time segment x_t that falls in $p+k$ for some integer value k. We remark that we need the random choices to be correlated. It can be verified that our expected cost is the same as the cost of our fractional algorithm, completing the analysis.

12.2 Notes

The results in this chapter are based on the work of Buchbinder, Jain and Naor [30]. The TCP acknowledgment problem was introduced by Dooly, Goldman and Scott [44] who gave a 2-competitive algorithm for the problem. This bound was later improved by [69] to a randomized $(1-1/e)$-competitive algorithm. Our algorithm is an alternative primal-dual view of this algorithm. Buchbinder et al. [31] studied an online inventory problem that is a variant of the classical joint replenishment problem (JRP) that has been studied extensively over the years. This inventory problem is actually a generalization of the dynamic TCP-acknowledgment. They designed a deterministic 3-competitive algorithm for the problem which is also based on a primal-dual approach.

Chapter 13

The Bounded Allocation Problem: Beating $1 - 1/e$

In this chapter we consider a special case of the ad-auctions problem that was studied in chapter 11 called the *allocation problem.* In the allocation problem, a seller is interested in selling products to a group of buyers, where buyer i has budget $B(i)$. The seller introduces the products one-by-one and sets a fixed price $b(j)$ for each product j[1]. Each buyer then announces to the seller (upon arrival of a product) whether it is interested in buying the current product for the set price. The seller then decides (instantly) to which of the interested buyers to sell the product. There is a lower bound example that shows that without further assumptions any algorithm for the problem has competitive ratio of at most $1 - 1/e$. However, in many realistic settings we may assume that for each product j the set of interested buyers is much smaller than the total number of buyers. The question is whether we can take advantage of this fact to improve on the competitiveness of the algorithm. We answer this question in the affirmative for the allocation problem. The main interesting idea we demonstrate here is a non-intuitive fractional algorithm for the problem.

For each product j let $S(j)$ be the set of interested buyers. We assume that there is an upper bound d such that for each product j, $|S(j)| \leq d$. We are interested in the case in which $d \ll n$, where n is the total number of buyers. We design an online

[1] In the ad-auctions problem considered in chapter 11 the price is not fixed for all buyers

	Lower Bound	Upper Bound		Lower Bound	Upper Bound
$d = 2$	0.75	0.75	$d = 10$	0.662	0.651
$d = 3$	0.704	0.704	$d = 20$	0.648	0.641
$d = 5$	0.686	0.672	$d \to \infty$	0.6321...	0.6321...

Table 13.1: Summary of upper and lower bounds on the competitive ratio for certain values of d.

Dual (Packing)	Primal (Covering)
Maximize: $\sum_{j=1}^{m} \sum_{i \in S(j)} b(j)y(i,j)$	Minimize : $\sum_{i=1}^{n} B(i)x(i) + \sum_{j=1}^{m} z(j)$
Subject to:	Subject to:
For each $1 \le j \le m$: $\sum_{i \in S(j)} y(i,j) \le 1$	For each $j, i \in S(j)$: $b(j)x(i) + z(j) \ge b(j)$
For each $1 \le i \le n$: $\sum_{j \mid i \in S(j)} b(j)y(i,j) \le B(i)$	For each i, j: $x(i), z(j) \ge 0$
For each i, j: $y(i,j) \ge 0$	

Figure 13.1: The fractional allocation problem (the dual) and the corresponding primal problem

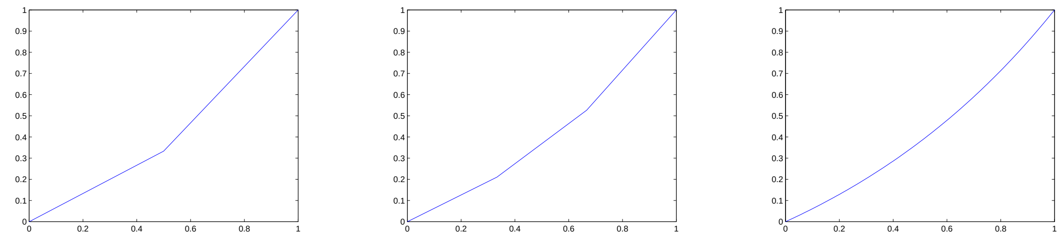
algorithm with competitive ratio $C(d) = 1 - \frac{d-1}{d\left(1+\frac{1}{d-1}\right)^{d-1}}$. This factor is strictly better than $1-1/e$ for any value of d, and approaches $(1-1/e)$ from above as d goes to infinity. We also prove lower bounds for the problem that indicate that the competitive factor of the online algorithm is quite tight. The improved bounds for certain values of d are shown in Figure 13.1.

13.1 The algorithm

The first step is to cast the problem as a linear program using the same formulation as in Chapter 11. Let $y(i,j)$ be an indicator to the event that item j was allocated to buyer i. Then the offline problem can be cast as the dual linear formulation in Figure 13.1

The main interesting idea for obtaining the improved competitive factor is in the production of a fractional solution to the problem. A fractional solution for the problem allows the algorithm to sell each product in fractions to several buyers. This problem has a motivation of its own in case products can be divided between buyers. An example of a divisible product is the allocation of bandwidth in a communication network. The fractional algorithm we design that generates a fractional solution in an online fashion is somewhat counter-intuitive. In particular, a newly arrived product is not split equally between buyers who have spent the least fraction of their budget. Such an algorithm is referred to as a "water level" algorithm and it is not hard to verify that it does not improve upon the $(1-1/e)$ worst case ratio, even for small values of d. Rather, the idea is to split the product between several buyers that have **approximately** spent the same fraction of their total budget.

The idea is to divide the buyers into *levels* according to the fraction of the budget that they have already spent. For $0 \le k \le d$, let $L(k)$ be the set of buyers that have spent at least a fraction of $\frac{k}{d}$ and less than a fraction of $\frac{k+1}{d}$ of their budget (buyers in level d have exhausted their budget). We refer to each $L(k)$ as level k and say that it is *nonempty* if it contains buyers. The formal description of the algorithm is the following:



$d = 2$ $\qquad$ $d = 3$ $\qquad$ $d \to \infty$

Figure 13.2: The function f_d for $d = 2$ and $d = 3$. The y middle value for $d = 2$ is $1/3$. The y middle values for $d = 3$ are $4/19$ and $10/19$.

Allocation Algorithm: Upon arrival of a new product j, allocate the product to the buyers according to the following rules:

- Allocate the product equally and continuously between interested buyers in the lowest non empty level that contain buyers from $S(j)$. If, during the allocation some of the buyers have moved to a higher level, then continue to allocate the product equally only among the buyers in the lowest level.
- If all interested buyers in the lowest level have moved to a higher level, then allocate the remaining fraction of the product equally and continuously between the buyers in the new lowest level. If all interested buyers have exhausted their budget, then stop allocating the remaining fraction of the product.

The idea behind the analysis is to find the best tradeoff function, f_d, that relates the value of each primal variable to the value of its corresponding dual constraint. It turns out that the best function is a piecewise linear function that consists of d linear segments. As d grows, the function approximates the exponential function $f_{(d=\infty)}(x) = \frac{e^x-1}{e-1}$. In order to define the linear pieces we define a geometric sequence a_t $(1 \le t \le d)$ inductively as follows:

$$a_1 = \frac{1}{d\left(1+\frac{1}{d-1}\right)^{d-1} - (d-1)}, \qquad \ldots \qquad , a_t = a_1 \cdot \left(1 + \frac{1}{d-1}\right)^{t-1}.$$

The sequence a_t is a geometric sequence and we only consider the first d elements in the sequence. The potential function f_d is defined for any $0 \le j \le d$ to be $f_d(\frac{j}{d}) \triangleq \sum_{t=1}^{j} a_t$. A simple calculation yields the following, for any j, $1 \le j \le d$:

$$f_d\left(\frac{j}{d}\right) = \sum_{i=1}^{j} a_i = a_1 \cdot \frac{\left(1+\frac{1}{d-1}\right)^j - 1}{(1+\frac{1}{d-1}) - 1} = a_1 \cdot \left[d\left(1 + \frac{1}{d-1}\right)^{j-1} - (d-1)\right].$$

In particular, setting $j = d$, we get $f_d\left(\frac{d}{d}\right) = 1$. This piecewise linear approximation allows us to analyze more accurately the algorithm and obtain better competitive factors.

The function f_d for $d = 2$, $d = 3$, and for d tending to infinity appears in Figure 13.2. Next, we use the potential function to prove that the allocation algorithm has the desired competitive factor.

Theorem 13.1. *The allocation algorithm is $C(d)$-competitive with respect to the optimal offline fractional solution, where: $C(d) = 1 - \frac{d-1}{d\left(1+\frac{1}{d-1}\right)^{d-1}}$.*

Proof. Let $Y(j)$ denote the total profit of the algorithm (the dual packing) in the jth iteration. In each iteration we maintain a corresponding feasible primal solution whose value is denoted by $X(j)$. Upon arrival of a new product we update both primal and dual programs. The dual (packing) program is updated by adding a new constraint corresponding to the new product which has arrived, and by adding a new term $b(j)y(i, j)$ to each constraint of an interested buyer. The primal program is updated by adding a new variable $z(j)$ for the new product and a constraint of the form $b(j)x(i)+z(j) \geq b(j)$ for each buyer who is interested in the new product.

Initially, the dual and primal programs are empty. In the jth iteration, the change in values of the primal and dual solutions is denoted by $\Delta X(j)$ and $\Delta Y(j)$, correspondingly. We prove that in each iteration:

$$\Delta X(j) \leq \frac{1}{C(d)} \cdot \Delta Y(j)$$

The primal solution is an assignment of values to the variables $x(i)$ and $z(j)$. Since these values are not used by the allocation algorithm, we can set them using future knowledge. For each buyer i, let $t(i)$ $(0 \leq t \leq d)$ be the largest level i to which this buyer belongs during the algorithm. Thus, buyer i spent overall at least $t(i)/d$ fraction of his budget. The variable $x(i)$ grows as a function of the fraction of money that buyer i spent, which in fact depends on the corresponding dual constraint. Specifically, for buyer i:

$$x(i) = \begin{cases} f_d\left(\frac{1}{B(i)} \sum_{j \mid i \in S(j)} b(j)y(i,j)\right) & \text{if } \frac{1}{B(i)} \sum_{j \mid i \in S(j)} b(j)y(i,j) \leq \frac{t(i)}{d} \\ f_d\left(\frac{t(i)}{d}\right) & \text{if } \frac{1}{B(i)} \sum_{j \mid i \in S(j)} b(j)y(i,j) \geq \frac{t(i)}{d} \end{cases}$$

The variables $x(i)$ are monotonically increasing and thus, once a primal constraint is satisfied, it remains satisfied throughout the run of the algorithm. Hence, in each iteration, it suffices to satisfy the newly added primal constraints.

Consider first the case in which product j was not fully sold by the algorithm. This means that at the end of the jth iteration all the buyers in $S(j)$ exhausted their budget. In this case the corresponding variables $x(i)$ at the end of the iteration are all 1, and thus all the new primal constraints are satisfied, and we can set $z(j) = 0$. We only need to show that the change in the primal profit in this iteration is not too large. When we increase a variable $y(i, j)$, the derivative of the dual profit of the algorithm is $b(j)$. The derivative of the primal cost is:

$$B(i) \cdot \frac{df_d}{d(y(i,j))} \leq B(i) \cdot \frac{b(j)}{B(i)} \cdot d \cdot a_d = \frac{1}{C(d)} \cdot b(j).$$

The inequality follows by taking the maximum derivative of the (convex) function f_d which is:

$$d \cdot a_d = da_1 \left(1 + \frac{1}{d-1}\right)^{d-1} = \frac{1}{C(d)}.$$

Thus, we get that in this iteration $\Delta X(j) \leq \frac{1}{C(d)} \cdot \Delta Y(j)$.

Assume now that product j was fully sold to the buyers. Let t, $0 \leq t \leq d-1$, be the highest level of buyers to which the product was sold. Since the algorithm always allocates the product to buyers in the lowest possible level it means that all buyers in $S(j)$ used at least t/d fraction of their money. Let $\Delta_0, \Delta_1, \ldots, \Delta_t$ be the fraction of the product that was allocated in each level $k \leq t$. By our assumption: $\sum_{k=1}^{t} \Delta_k = 1$. We consider two cases.

Case 1: All the buyers in $S(j)$ spend during the algorithm at least t'/d of their budget for $t' > t$. In this case, for each buyer i, the derivative of the primal cost due to the change in $x(i)$ is:

$$B(i) \cdot \frac{df_d}{d(y(i,j))} \leq B(i) \cdot \frac{b(j)}{B(i)} \cdot d \cdot a_{t+1} = b(j) \cdot d \cdot a_{t+1}.$$

The inequality follows by taking the derivative of f_d in the highest level in which the product was sold. We fully allocate the product and hence $\sum_{i \in S(j)} y(i,j) = 1$. Thus, the total change of the primal cost due to the change in the variables $x(i)$ is at most $b(j) \cdot d \cdot a_{t+1}$. Since all buyers in $S(j)$ eventually spend during the algorithm at least $(t+1)/d$ of their budget, variable $x(i)$ corresponding to buyer $i \in S(j)$ will be at the end of the allocation process at least $f(\frac{t+1}{d})$. Therefore, it is safe to set $z(j) = b(j) \cdot (1 - f(\frac{t+1}{d}))$ in order to satisfy all the new primal constraints. Thus, the total change in the primal cost in this iteration is:

$$\begin{aligned}
z(j) + \sum_{i \in S'(j)} B(i)\Delta(x(i)) &\leq b(j)\left(1 - f(\frac{t+1}{d})\right) + b(j) \cdot d \cdot a_{t+1} \\
&= b(j)\left(1 - a_1 \cdot \left[d\left(1 + \frac{1}{d-1}\right)^t - (d-1)\right]\right) + b(j) \cdot d \cdot a_1 \cdot \left(1 + \frac{1}{d-1}\right)^t \\
&= b(j)\left(1 + a_1 \cdot (d-1)\right) = b(j)\left(1 + \frac{d-1}{d\left(1 + \frac{1}{d-1}\right)^{d-1} - (d-1)}\right) = \frac{1}{C(d)} \cdot b(j).
\end{aligned}$$

Since the product was fully sold the dual profit in this case is $b(j)$ and hence we are done with this case.

Case 2: There exists at least one buyer in $S(j)$ who eventually spends (throughout the algorithm) less than a fraction of $(t+1)/d$ of his budget (but spend at least t/d). In this case, in order to satisfy the new primal constraint, it is only safe to set $z(j) = b(j) \cdot (1 - f(\frac{t}{d}))$. However, note that the buyer that spent less than $(t+1)/d$ fraction of its money was present throughout the whole process of dividing the product equally between all buyers in last level t. Thus, by our algorithm, this buyer receives at least a fraction $\frac{\Delta_t}{d}$ of the product. By the definition of the function associated with the variable $x(i)$, the growth function of $x(i)$ in this segment (which is larger than $t(i)$) is zero. Thus, the change in the primal cost due to the increase of the dual variables in the highest level is at most:

$$b(j)a_{t+1}d \cdot \frac{d-1}{d}\Delta_t = b(j) \cdot (d-1) \cdot a_{t+1} \cdot \Delta_t. \tag{13.1}$$

The change in the primal cost due to the increase of the dual variables in lower levels is at most $b(j) \cdot d \cdot a_t \cdot (1-\Delta_t)$. But, $a_{t+1} = a_t \cdot \left(1 + \frac{1}{d-1}\right)$, and so $a_t = \frac{d-1}{d} \cdot a_{t+1}$. Thus, the change in the primal cost due to change in the variables $x(i)$ is at most:

$$b(j) \cdot d \cdot a_t \cdot (1-\Delta_t) = b(j) \cdot d \cdot \frac{d-1}{d} \cdot a_{t+1} \cdot (1-\Delta_t) = b(j) \cdot (d-1) \cdot a_{t+1} \cdot (1-\Delta_t). \tag{13.2}$$

Adding up Equations (13.1) and (13.2), we get that the total change in the primal cost due to the increase in the primal variables $x(i)$ is $b(j)(d-1)a_{t+1}$. Since $f(\frac{t+1}{d}) = a_{t+1} + f(\frac{t}{d})$, the total change in the primal cost is at most:

$$\begin{aligned} b(j)\left(1 - f(\frac{t}{d})\right) + b(j) \cdot (d-1) \cdot a_{t+1} &= b(j)\left(1 - f(\frac{t+1}{d}) + a_{t+1})\right) + b(j) \cdot (d-1) \cdot a_{t+1} \\ &= b(j)\left(1 - f(\frac{t+1}{d})\right) + b(j)da_{t+1} = \frac{1}{C(d)} \cdot b(j). \end{aligned}$$

This change is exactly the same as in case (1). Similarly to case (1), the product was fully sold and so the dual profit is $b(j)$ and we are done with this case. ☐

Lower Bounds. For any value of d it is not hard to prove the following lower bound.

Lemma 13.2. *For any value d: $C(d) \leq 1 - \frac{k - kH(d) + \sum_{i=1}^{k} H(d-i)}{d}$, where $H(\cdot)$ is the harmonic number, and k is the largest value for which $H(d) - H(d-k) \leq 1$*

This bound is only tight for $d = 2$, but it is also possible to derive better tailor-made lower bounds for specific values of d. In particular it is not hard to show that the algorithm is optimal for $d = 3$.

Rounding the fractional solution It is possible to apply standard randomized rounding techniques in an online fashion. The main issue is that when applying randomized rounding the algorithm may allocate buyer i products with total value of more than $B(i)$. However, using standard techniques one can prove that when the budget of each buyer is much larger than the price of the individual products, then with high probability the budget excess is not going to be large, i.e., the additional loss in the competitive factor is $o(1)$. In this case it is also possible to apply de-randomization methods to the randomized rounding algorithm to obtain a deterministic algorithm for the problem.

13.2 Notes

The results in this chapter are based on the work of Buchbinder, Jain and Naor [30]. They also showed how to de-randomize the algorithm to obtain a deterministic algorithm for the problem. The same technique can be used to improve the competitive ratio for other problems. In the ski rental problem, for example, one can obtain using this method an algorithm with improved competitive factor of $C(B)$, where B is the cost of buying the skis. In the dynamic TCP acknowledgment problem studied in Chapter 12 it is also possible to improve the competitive ratio in certain scenarios. If it is possible to assume that packets only arrive in certain discrete times of $1/d$ (and not at any continuous time) then the competitive ratio can be improved to $C(d)$.

Chapter 14

Extension to General Packing/Covering Constraints

In this chapter we design primal-dual algorithms for more general settings of covering packing linear formulations. In the more general (fractional) covering problem the objective is still to minimize the total cost given by a linear cost function $\sum_{i=1}^{n} c(i)x(i)$. However, the feasible solution space is defined by a set of m linear constraints of the form $\sum_{i=1}^{n} a(i,j)x(i) \geq b(j)$, where the entries $a(i,j)$ and $b(j)$ are non-negative. This generalizes the setting of Chapter 4 in which $a(i,j) \in \{0,1\}$ and $b(j) = 1$. Given an instance of a covering problem we can always normalize each constraint to the form: $\sum_{i=1}^{n} a(i,j)x(i) \geq 1$. Any primal covering instance has a corresponding dual packing problem that provides a lower bound on any feasible solution to the instance. A general form of a (normalized) primal covering problem along with its (normalized) dual packing problem is given in Figure 14.1. Throughout this chapter we refer to the covering problem as the "primal problem" and the packing problem as the "dual problem".

The online setting we study here is the same as the online setting studied in Chapter 4. In the general *online fractional covering problem* the cost function is known in advance, but the linear constraints that define the feasible solution space are given to the algorithm one-by-one. Again we are only allowed to increase the variables $x(i)$, but not to decrease any previously increased variable. In the general *online fractional packing problem* the values $c(i)$ $(1 \leq i \leq n)$ are known in advance. However, the profit function and the exact packing constraints are not known in advance. In the jth round a new variable $y(j)$ is introduced to the algorithm, along with its set of coefficients $a(i,j)$ $(1 \leq i \leq n)$[1]. The algorithm may increase the value of a variable $y(j)$ only in the round where it is given, and may not decrease or increase the values of any previously given variables.

[1]We can always normalize the new variable such that its coefficient in the objective function is 1.

Primal (Covering)		Dual (Packing)	
Minimize:	$\sum_{i=1}^{n} c(i)x(i)$	Maximize:	$\sum_{j=1}^{m} y(j)$
Subject to:		Subject to:	
For each $1 \le j \le m$:	$\sum_{i=1}^{n} a(i,j)x(i) \ge 1$	For each $1 \le i \le n$:	$\sum_{j=1}^{m} a(i,j)y(j) \le c(i)$
For each $1 \le i \le n$:	$x(i) \ge 0$	For each $1 \le j \le m$:	$y(j) \ge 0$

Figure 14.1: Primal (covering) and dual (packing) problems

14.1 The General Online Fractional Packing Problem

In this section we describe an online scheme for computing a near-optimal fractional solution for the general online fractional packing problem. The scheme gets the desired competitive ratio $B > 0$ and returns a solution which is within a factor of B of the optimal, and which does not violate the packing constraints by too much (to be made more precise shortly). We prove that the scheme is optimal up to constant factors. Our scheme simultaneously maintains primal (covering) and dual (packing) solutions for the primal and dual instances.

Initially, each variable $x(i)$ is initialized to zero. In each round a new variable $y(j)$ is introduced along with its coefficients $a(i,j)$ $(1 \le i \le n)$. In the corresponding primal sub-instance a new constraint is introduced of the form $\sum_{i=1}^{n} a(i,j)x(i) \ge 1$. Without loss of generality, we can assume that this constraint has at least one non-zero coefficient, otherwise it means that there is no bound on the value of $y(j)$ and the profit function is unbounded. The algorithm increases the value of the new variable $y(j)$ and the values of the primal variables $x(i)$ until the new primal constraint is satisfied. The augmentation method is described here in a continuous fashion, but it is not hard to implement the augmentation in a discrete way in any desired accuracy. In our continuous description the variables $x(i)$ behave according to a monotonically increasing function of $y(j)$. To implement the scheme in a discrete fashion, one should find the minimal $y(j)$ such that the new primal constraint is satisfied. Note that variable $y(j)$ is being increased only in the jth round and the values of the primal variables never decrease. In the following we describe the jth round. The performance of the scheme is analyzed in Theorem 14.1.

1. $y(j) \leftarrow 0$; For each $x(i)$: $a_i(\max) \leftarrow \max_{k=1}^{j}\{a(i,k)\}$.
2. While $\sum_{i=1}^{n} a(i,j)x(i) < 1$:
   (a) Increase $y(j)$ continuously.
   (b) Increase each variable $x(i)$ by the following increment function:

$$x(i) \leftarrow \max\left\{x(i), \frac{1}{na_i(\max)}\left[\exp\left(\frac{B}{2c(i)}\sum_{k=1}^{j} a(i,k)y(k)\right) - 1\right]\right\}.$$

Theorem 14.1. *For any $B > 0$, the above scheme is a $B-$competitive algorithm for the* *general online fractional packing problem. Also, for any constraint it holds:*

$$\sum_{k=1}^{m} a(i,k)y(k) = c(i) \cdot O\left(\frac{\log n + \log \frac{a_i(\max)}{a_i(\min)}}{B}\right).$$

Where, $a_i(\max) = \max_{k=1}^{m}\{a(i,k)\}$ *and* $a_i(\min) = \min_{k=1}^{m}\{a(i,k) | a(i,k) \neq 0\}$.

Proof. Let $X(j)$ and $Y(j)$ be the values of the primal and dual solutions, respectively, obtained in round j. We prove the following claims on $X(j)$ and $Y(j)$:

1. In each round j: $Y(j) \geq X(j)/B$.
2. The primal solution produced by the scheme is feasible.
3. For any dual constraint:

$$\sum_{k=1}^{m} a(i,k)y(k) \leq c(i) \cdot \frac{2\log\left(1 + \frac{na_i(\max)}{a_i(\min)}\right)}{B} = c(i) \cdot O\left(\frac{\log n + \log \frac{a_i(\max)}{a_i(\min)}}{B}\right).$$

The proof of the theorem then follows directly from weak duality.

Proof of (1): Note first that when the value of $a_i(\max)$ increases, the value of the primal solution does not change. Thus, the value of the primal solution only increases when the dual solution increases. Initially, the values of the primal and dual solutions are zero. Consider the jth round in which $y(j)$ is being increased continuously. We prove that $\frac{\partial X(j)}{\partial y(j)} \leq B\frac{\partial Y(j)}{\partial y(j)}$, concluding that $X(j) \leq B \cdot Y(j)$.

$$\begin{aligned}
\frac{\partial X(j)}{\partial y(j)} &= \sum_{i=1}^{n} c(i)\frac{\partial x(i)}{\partial y(j)} \\
&\leq \sum_{i=1}^{n} \frac{c(i)Ba(i,j)}{2c(i)} \frac{1}{na_i(\max)} \exp\left(\frac{B}{2c(i)}\sum_{k=1}^{j} a(i,k)y(k)\right) \qquad (14.1) \\
&= \frac{B}{2}\sum_{i=1}^{n} a(i,j)\left[\frac{1}{na_i(\max)}\left(\exp\left(\frac{B}{2c(i)}\sum_{k=1}^{j} a(i,k)y(k)\right) - 1\right) + \frac{1}{na_i(\max)}\right] \\
&\leq \frac{B}{2}\sum_{i=1}^{n} a(i,j)\left[x(i) + \frac{1}{na_i(\max)}\right] \leq \frac{B}{2}(1+1) = B = B \cdot \frac{\partial Y(j)}{\partial y(j)}, \qquad (14.2)
\end{aligned}$$

where Inequality (14.1) follows from the derivative of $x(i)$, and (14.2) follows since: (i) $\sum_{i=1}^{n} a(i,j)x(i) < 1$, (ii) $x(i) \geq \frac{1}{na_i(\max)}\left[\exp\left(\frac{B}{2c(i)}\sum_{k=1}^{j} a(i,k)y(k)\right) - 1\right]$, and (iii) $\frac{1}{n}\sum_{i=1}^{n} \frac{a(i,j)}{a_i(\max)} \leq 1$. The final equality follows since the value of the dual is $\sum_{k=1}^{j} y(k)$, and so $\frac{\partial Y(j)}{\partial y(j)} = 1$.

Proof of (2): This claim is trivial since we increase the primal variables until the current primal constraint becomes feasible. We never decrease any $x(i)$, so (feasible) constraints remain feasible.

Proof of (3): Consider the ith dual constraint of the form $\sum_{k=1}^{j} a(i,k)y(k) \leq c(i)$. Each time a variable $y(k)$ with coefficient $a(i,k) > 0$ is increased, the primal variable $x(i)$ is increased too. Let $a_i(\min) = \min_{k=1}^{m}\{a(i,k)|a(i,k) \neq 0\}$ and $a_i(\max) = \max_{k=1}^{m}\{a(i,k)\}$ be as defined previously. During the run of the algorithm, $x(i) \leq 1/a_i(\min)$, since if equality holds, then each primal constraint ($1 \leq j \leq m$) with $a(i,j) > 0$ is already feasible. Thus, we get the following:

$$\frac{1}{na_i(\max)}\left[\exp\left(\frac{B}{2c(i)}\sum_{k=1}^{j} a(i,k)y(k)\right) - 1\right] \leq x(i) \leq 1/a_i(\min).$$

Simplifying, we obtain:

$$\sum_{k=1}^{j} a(i,k)y(k) \leq \frac{2\log\left(1 + \frac{na_i(\max)}{a_i(\min)}\right)}{B} \cdot c(i).$$

□

14.1.1 Lower Bounds

In this section we prove a simple lower bound showing that the additional additive factor of $\log \frac{a_i(\max)}{a_i(\min)}$ is indeed necessary.

Lemma 14.2. *There is an instance of the general fractional packing problem with a single constraint such that $\sum_{j=1}^{m} a(i,j)y(j) \geq \frac{H(a(\max)/a(\min))}{B}$ for any online $B-$competitive algorithm, where $H(m)$ denotes the mth harmonic number, and $a(\max)/a(\min)$ is the ratio between the maximum and minimum entries in the (single) constraint.*

Proof. Consider the following instance, for any m:

$$\max \quad \sum_{j=1}^{m} y(j)$$

$$\text{subject to} \qquad \sum_{j=1}^{m} (m-j+1)y(j) \leq 1.$$

Note that for this instance $a(\max)/a(\min) = m$. The variables $y(j)$ arrive one by one. After the jth round (for each j), the optimal offline value is $1/(m-j+1)$. Thus, the value of the objective function given by a B-competitive algorithm must be at least

$1/B(m-j+1)$. This yields the following sequence of inequalities:

$$\begin{aligned} y(1) &\geq 1/Bm \\ y(1)+y(2) &\geq 1/B(m-1) \\ y(1)+y(2)+y(3) &\geq 1/B(m-2) \\ &\cdots \end{aligned}$$

Summing up over all m inequalities we get the desired bound:

$$\sum_{j=1}^{m}(m-j+1)y(j) \geq \frac{1}{B}\sum_{j=1}^{m}\frac{1}{m-j+1} = \frac{H(m)}{B}.$$

□

14.2 The General Online Fractional Covering Problem

In this section we describe our online scheme for computing a near-optimal fractional solution for the online fractional covering problem. Our scheme for the general online fractional covering problem gets a parameter $B > 0$. With $B > 0$ the competitive ratio of the scheme is $O(\frac{\log n}{B})$ and the following holds for each constraint: $\sum_{i=1}^{n} a(i,j)x(i) \geq \frac{1}{B}$. The scheme works in phases: When the first constraint is introduced, the scheme generates the first lower bound:

$$\alpha(1) \leftarrow \frac{1}{B}\cdot\min_{i=1}^{n}\left\{\frac{c(i)}{a(i,1)}\right\} \leq \frac{OPT}{B}.$$

During the rth phase, it is assumed that the lower bound on the optimum is $\alpha(r)$, as long as the total primal cost does not exceed $\alpha(r)$. When the primal cost exceeds this bound, the scheme "forgets" about all the values given to the primal and dual variables so far, and starts a new phase in which the lower bound is doubled, i.e., $\alpha(r+1) \leftarrow 2\alpha(r)$. Nevertheless, the values of the "forgotten" variables are accounted for in the total cost of the solution. That is, the algorithm maintains in each phase r a new set of variables $x(i,r)$. However, since the variables of the linear program are required to be monotonically non-decreasing, the value of each variable $x(i)$ is actually set to $\max_r\{x(i,r)\}$ (or alternatively $\sum_r x(i,r)$). The cost of maintaining the variables of the linear program is, thus, at most the cost of maintaining the new variables in each phase. When we start processing a new phase we also set to zero all dual variables, and start processing again all primal constraints from the first one. Thus, in each such phase, our algorithm produces "fresh" primal and dual solutions.

In the following we describe the behavior of our scheme in one round of the rth phase. Let $\sum_{i=1}^{n} a(i,j)x(i) \geq 1$ be the new primal constraint that is introduced and let $y(j)$ be the corresponding dual variable. The values of the primal and dual variables are increased as follows. Note that during each phase $x(i)$ only increases. The performance of the scheme is analyzed in Theorem 14.3.

1. $y(j) \leftarrow 0$
2. While $\sum_{i=1}^{n} a(i,j)x(i) < \frac{1}{B}$:
   (a) increase $y(j)$ continuously.
   (b) Increase each variable $x(i)$ by the following increment function:
$$x(i) \leftarrow \frac{\alpha(r)}{2nc(i)} \exp\left(\frac{\log 2n}{c(i)} \sum_{k=1}^{j} a(i,k)y(k)\right).$$

Theorem 14.3. *For any $B > 0$, the scheme for the general online fractional covering problem achieves a competitive ratio of $O(\frac{\log n}{B})$, such that for each constraint $\sum_{i=1}^{n} a(i,j)x(i) \geq \frac{1}{B}$.*

Proof. Let $X(r)$ and $Y(r)$ be the values of the primal and dual solutions, respectively, generated during the rth phase. We prove the following claims on $X(r)$ and $Y(r)$:

1. For each finished phase r: $Y(r) \geq \frac{B\alpha(r)}{2\log 2n}$.
2. The dual solution generated during the rth phase is feasible.
3. The total cost of the primal solutions generated from the first phase until the rth phase is less than $2\alpha(r)$.
4. For any primal constraint given to the algorithm, $\sum_{i=1}^{n} a(i,j)x(i) \geq \frac{1}{B}$.

From the first three claims together with weak duality we conclude that the total cost of the primal solutions in all the phases up to phase r is at most:

$$2\alpha(r) \leq 4\alpha(r-1) \leq 8 \cdot \frac{\log 2n}{B} \cdot Y(r-1) \leq 8 \cdot \frac{\log 2n}{B} \cdot OPT.$$

Notice that if the scheme finishes in the first phase, then the total cost is at most $\alpha(1) \leq \frac{OPT}{B}$.

Proof of (1): Initially, $x(i) = \frac{\alpha(r)}{2nc(i)}$, and so $X(r)$ is initially at most $\alpha(r)/2$. The total profit of the dual solution is initially zero. From then on, the primal cost increases only when some dual variable $y(j)$ is increased. When the phase ends, $X(r) \geq \alpha(r)$. Thus, it suffices to prove that during the phase

$$\frac{\partial X(r)}{\partial y(j)} \leq \frac{\log 2n}{B} \frac{\partial Y(r)}{\partial y(j)}.$$

This follows since,

$$\begin{aligned}
\frac{\partial X(r)}{\partial y(j)} &= \sum_{i=1}^{n} c(i)\frac{\partial x(i)}{\partial y(j)} \\
&= \sum_{i=1}^{n} \frac{c(i)\log(2n)a(i,j)}{c(i)} \frac{\alpha(r)}{2nc(i)} \exp\left(\frac{\log 2n}{c(i)} \sum_{k=1}^{j} a(i,k)y(k)\right) \\
&= \log 2n \sum_{i=1}^{n} a(i,j)x(i) \leq \frac{\log 2n}{B} = \frac{\log 2n}{B} \cdot \frac{\partial Y(j)}{\partial y(j)}, \qquad (14.3)
\end{aligned}$$

where (14.3) follows since $\sum_{i=1}^{n} a(i,j)x(i) \leq \frac{1}{B}$. The final equality follows since the value of the dual solution is $\sum_{k=1}^{j} y(k)$ and thus $\frac{\partial Y(j)}{\partial y(j)} = 1$.

Proof of (2): Consider the ith dual constraint of the form $\sum_{k=1}^{m} a(i,k)y(k) \leq c(i)$. Each time variable $y(k)$ with coefficient $a(i,k) > 0$ is increased, the corresponding primal variable $x(i)$ is increased too. During the rth phase of the algorithm, $x(i) \leq \frac{\alpha(r)}{c(i)}$, since otherwise it would have contributed to the cost of the primal solution more than $\alpha(r)$, and the current phase would have ended. Thus, we get the following equation:

$$x(i) = \frac{\alpha(r)}{2nc(i)} \exp\left(\frac{\log 2n}{c(i)} \sum_{k=1}^{j} a(i,k)y(k)\right) \leq \frac{\alpha(r)}{c(i)}.$$

Simplifying we get the desired result:

$$\sum_{k=1}^{m} a(i,k)y(k) \leq c(i).$$

Proof of (3): We bound the total cost paid by the online algorithm. The total primal cost in the rth phase is at most $\alpha(r)$. Since the ratio between $\alpha(k)$ and $\alpha(k-1)$ is 2, we get that the total cost until the rth phase is at most $\sum_{k=1}^{r} \alpha(k) \leq 2\alpha(r)$.

Proof of (4): The claim is trivial, since each round terminates only when the value of the left hand side of the new primal constraint is at least $1/B$. The value of each variable $x(i)$ never decreases, thus all previous primal constraints remain feasible. $\square$

14.3 Notes

The results in this chapter are based on the work of Buchbinder and Naor [32]. In this work a more general setting is considered in which each variable in the covering problem has an upper bound (box constraint).

Chapter 15

Conclusions and Further Research

We showed in this thesis how to extend the primal-dual method to the setting of online algorithms, and also showed that it is applicable to a wide variety of problems. There are other online problems not discussed here that can be stated and analyzed via the primal-dual approach. Such problems are, for example, the admission control problem in [5], the parking permit problem in [81] and the inventory problem in [31]. We are certain that many other online problems and algorithms also fit our online framework.

There are many questions and directions for further research. First, it will be nice to show other online scenarios that can benefit from the primal-dual framework. One such interesting problem is the k-server problem. For this problem the randomized competitiveness is a major open question. Another interesting problem is the metrical task problem in general metrics. It will be nice to use the primal-dual framework to close the gap between the upper and lower bounds. Another research direction is to extend the primal-dual framework itself beyond covering/packing formulations. The framework cannot be extended to any general LP formulation. However, there can be other special linear program formulations that can be solved online.

Bibliography

[1] D. Achlioptas, M. Chrobak, and J. Noga. Competitive analysis of randomized paging algorithms. *Theor. Comput. Sci.*, 234(1-2):203–218, 2000.

[2] S. Albers, S. Arora, and S. Khanna. Page replacement for general caching problems. In *SODA '99: Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms*, pages 31–40, 1999.

[3] N. Alon, B. Awerbuch, Y. Azar, N. Buchbinder, and J. Naor. The online set cover problem. In *Proceedings of the 35th annual ACM Symposium on the Theory of Computation.*, pages 100–105, 2003. To appear, Siam Journal on Computing.

[4] N. Alon, B. Awerbuch, Y. Azar, N. Buchbinder, and J. Naor. A general approach to online network optimization problems. *ACM Transactions on Algorithms*, 2(4):640–660, 2006.

[5] N. Alon, Y. Azar, and S. Gutner. Admission control to minimize rejections and online set cover with repetitions. In *SPAA '05: Proceedings of the seventeenth annual ACM symposium on Parallelism in algorithms and architectures*, pages 238–244, 2005.

[6] N. Alon and J. H. Spencer. *The probabilistic method*. Wiley, New York, 2 edition, 2000.

[7] N. Andelman and Y. Mansour. Auctions with budget constraints. In *SWAT 2004: 9th Scandinavian Workshop on Algorithm Theory*, pages 26–38, 2004.

[8] J. Aspnes, Y. Azar, A. Fiat, S. A. Plotkin, and O. Waarts. On-line routing of virtual circuits with applications to load balancing and machine scheduling. *J. ACM*, 44(3):486–504, 1997.

[9] B. Awerbuch, Y. Azar, and Y. Bartal. On-line generalized Steiner problem. In *Proc. of the 7th Ann. ACM-SIAM Symp. on Discrete Algorithms*, pages 68–74, 1996.

[10] B. Awerbuch, Y. Azar, A. Fiat, and T. Leighton. Making commitments in the face of uncertainty: how to pick a winner almost every time. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing*, pages 519–530, 1996.

[11] B. Awerbuch, Y. Azar, and S. Plotkin. Throughput-competitive online routing. In *Proc. of 34th FOCS*, pages 32–40, 1993.

[12] Y. Azar. On-line load balancing. *Online Algorithms - The State of the Art, Springer*, pages 178–195, 1998.

[13] Y. Azar, J. Naor, and R. Rom. The competitiveness of on-line assignments. *Journal of Algorithms*, 18:221–237, 1995.

[14] N. Bansal, N. Buchbinder, and J. S. Naor. A primal-dual randomized algorithm for weighted paging. In *FOCS '07: Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science*, pages 507–517, 2007.

[15] N. Bansal, N. Buchbinder, and J. S. Naor. Randomized competitive algorithms for generalized caching. In *to appear in STOC '08*, 2008.

[16] A. Bar-Noy, R. Bar-Yehuda, A. Freund, J. S. Naor, and B. Schieber. A unified approach to approximating resource allocation and scheduling. *J. ACM*, 48(5):1069–1090, 2001.

[17] R. Bar-Yehuda and S. Even. A linear-time approximation algorithm for the weighted vertex cover problem. *Journal of Algorithms*, 2:198–203, 1981.

[18] Y. Barta, B. Bollobás, and M. Mendel. A ramsey-type theorem for metric spaces and its applications for metrical task systems and related problems. In *FOCS '01: Proceedings of the 42nd IEEE symposium on Foundations of Computer Science*, page 396, 2001.

[19] Y. Bartal, A. Blum, C. Burch, and A. Tomkins. A polylog(n)-competitive algorithm for metrical task systems. In *Proceedings of the 29th Annual ACM Symposium on Theory of computing*, pages 711–719, 1997.

[20] Y. Bartal, N. Linial, M. Mendel, and A. Naor. On metric ramsey-type phenomena. In *STOC '03: Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 463–472, 2003.

[21] Y. Bartal and M. Mendel. Randomized k-server algorithms for growth-rate bounded graphs. *J. Algorithms*, 55(2):192–202, 2005.

[22] P. Berman and C. Coulston. On-line algorithms for Steiner tree problems. In *Proc. of the 29th annual ACM Symp. on the Theory of Computation*, pages 344–353, 1997.

[23] D. Bertsekas and R. Gallager. *Data networks*. Prentice-Hall, Inc., 1987.

[24] A. Blum, C. Burch, and A. Kalai. Finely-competitive paging. In *IEEE Symposium on Foundations of Computer Science*, pages 450–458, 1999.

[25] A. Blum, M. Furst, and A. Tomkins. What to do with your free time: algorithms for infrequent requests and randomized weighted caching, 1996.

[26] A. Blum and J. Hartline. Near-optimal online auctions. In *Proceedings of the 16th Annual ACMSIAM Symposium on Discrete Algorithms*, 2005.

[27] C. Borgs, J. Chayes, N. Immorlica, M. Mahdian, and A. Saberi. Multi-unit auctions with budget-constrained bidders. In *Proc. of the 6th ACM conference on Electronic commerce*, pages 44–51, 2005.

[28] A. Borodin and R. El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, 1998.

[29] A. Borodin, N. Linial, and M. E. Saks. An optimal on-line algorithm for metrical task system. *J. ACM*, 39(4):745–763, 1992.

[30] N. Buchbinder, K. Jain, and J. S. Naor. Online primal-dual algorithms for maximizing ad-auctions revenue. In *ESA '07: Proceedings of the 15th Annual European Symposium*, pages 253–264, 2007.

[31] N. Buchbinder, T. Kimbrel, R. Levi, K. Makarychev, and M. Sviridenko. Online make-to-order joint replenishment model: primal dual competitive algorithms. In *SODA '08: Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 952–961, 2008.

[32] N. Buchbinder and J. Naor. Online primal-dual algorithms for covering and packing problems. In *13th Annual European Symposium on Algorithms*, 2005.

[33] N. Buchbinder and J. Naor. Fair online load balancing. In *18th ACM Symp. on Parallelism in Algorithms and Architectures*, 2006.

[34] P. Cao and S. Irani. Cost-aware WWW proxy caching algorithms. In *Proceedings of the 1997 Usenix Symposium on Internet Technologies and Systems (USITS-97)*, 1997.

[35] R. D. Carr, L. K. Fleischer, V. J. Leung, and C. A. Phillips. Strengthening integrality gaps for capacitated network design and covering problems. In *SODA '00: Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, pages 106–115, 2000.

[36] C. Chekuri, G. Even, A. Gupta, and D. Segev. Set connectivity problems in undirected graphs and the directed steiner network problem. In *SODA '08: Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 532–541, 2008.

[37] C. Chekuri, S. Khanna, J. Naor, and L. Zosin. A linear programming formulation and approximation algorithms for the metric labeling problem. *SIAM J. Discret. Math.*, 18(3):608–625, 2005.

[38] C. Chekuri, S. Khanna, and F. B. Shepherd. The all-or-nothing multicommodity flow problem. In *STOC '04: Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 156–165, 2004.

[39] M. Chrobak, H. Karloff, T. Payne, and S. Vishwanathan. New results on server problems. *SIAM J. Discret. Math.*, 4(2):172–181, 1991.

[40] V. Chvátal. A greedy heuristic for the set-covering problem. In *Mathematics of Operations Research*, pages 233–235, 1979.

[41] V. Chvátal. *Linear Programming*. W.H. Freeman, 1983.

[42] E. Cohen and H. Kaplan. Lp-based analysis of greedy-dual-size. In *SODA '99: Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms*, pages 879–880, 1999.

[43] B. Csaba and S. Lodha. A randomized on-line algorithm for the k-server problem on a line. *Random Structures and Algorithms*, 29(1):82–104, 2006.

[44] D. R. Dooly, S. Goldman, and S. D. Scott. On-line analysis of the TCP acknowledgement delay problem. *Journal of the ACM*, 48:243–273, 2001.

[45] J. Fakcharoenphol, S. Rao, and K. Talwar. A tight bound on approximating arbitrary metrics by tree metrics. In *STOC '03: Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 448–455, 2003.

[46] U. Feige. A threshold of $\ln n$ for approximating set cover. *Journal of the ACM*, 45(4):634–652, 1998.

[47] A. Fiat, R. M. Karp, M. Luby, L. A. McGeoch, D. D. Sleator, and N. E. Young. Competitive paging algorithms. *J. Algorithms*, 12(4):685–699, 1991.

[48] A. Fiat and M. Mendel. Better algorithms for unfair metrical task systems and applications. *SIAM J. Comput.*, 32(6):1403–1422, 2003.

[49] A. Fiat and M. Mendel. Better algorithms for unfair metrical task systems and applications. *SIAM J. Comput.*, 32(6):1403–1422, 2003.

[50] A. Fiat, Y. Rabani, and Y. Ravid. Competitive k-server algorithms. *Journal of Computer and System Sciences*, 48(3):410–428, 1994.

[51] L. Fleischer. A fast approximation scheme for fractional covering problems with variable upper bounds. In *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1001–1010, 2004.

[52] L. K. Fleischer. Approximating fractional multicommodity flow independent of the number of commodities. *SIAM Journal on Discrete Mathematics*, 13(4):505–520, 2000.

[53] N. Garg and R. Khandekar. Fractional covering with upper bounds on the variables: solving LPs with negative entries. In *Proceedings, 12th European Symposium on Algorithms (ESA)*, volume 3221 of *LNCS*, pages 371–382, 2004.

[54] N. Garg and J. Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In *IEEE Symposium on Foundations of Computer Science*, pages 300–309, 1998.

[55] N. Garg, G. Konjevod, and R. Ravi. A polylogarithmic approximation algorithm for the group Steiner tree problem. In *SODA: ACM-SIAM Symposium on Discrete Algorithms (A Conference on Theoretical and Experimental Analysis of Discrete Algorithms)*, 1998.

[56] A. Goel and A. Meyerson. Simultaneous optimization via approximate majorization for concave profits or convex costs. In *To appear in Algorithmica*, 2005.

[57] A. Goel, A. Meyerson, and S. A. Plotkin. Approximate majorization and fair online load balancing. In *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 384–390, 2001.

[58] A. Goel, A. Meyerson, and S. A. Plotkin. Combining fairness with throughput: online routing with multiple objectives. *J. Comput. Syst. Sci.*, 63(1):62–79, 2001.

[59] M. Goemans and D. Williamson. A general approximation technique for constrained forest problems. *SIAM Journal on Computing*, 24:296–317, 1995.

[60] R. L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45(1):1563–1581, 1966.

[61] D. Hochbaum. *Approximation algorithms for NP-Hard problems.* PWS Publishing Company, Boston, MA, 1997.

[62] M. Imase and B. Waxman. Dynamic Steiner tree problem. *SIAM Journal Discrete Math.*, 4:369–384, 1991.

[63] S. Irani. Competitive analysis of paging: a survey. In *Proceedings of the Workshop on Online Algorithms, Dagstuhl, Germany.*

[64] S. Irani. Page replacement with multi-size pages and applications to web caching. In *STOC '97: Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 701–710, 1997.

[65] S. Irani. Randomized weighted caching with two page weights. *Algorithmica*, 32(4):624–640, 2002.

[66] J. Jaffe. Bottleneck flow control. *IEEE Transactions on Communications*, 29(7):954–962, 1981.

[67] D. S. Johnson. Approximation algorithms for combinatorial problems. *J. Comput. System Sci.*, 9:256–278, 1974.

[68] B. Kalyanasundaram and K. R. Pruhs. An optimal deterministic algorithm for online b -matching. *Theoretical Computer Science*, 233(1–2):319–325, 2000.

[69] A. R. Karlin, C. Kenyon, and D. Randall. Dynamic TCP acknowledgement and other stories about e/(e-1). In *ACM Symposium on Theory of Computing*, pages 502–509, 2001.

[70] A. R. Karlin, M. S. Manasse, L. A. McGeoch, and S. S. Owicki. Competitive randomized algorithms for nonuniform problems. *Algorithmica*, 11(6):542–571, 1994.

[71] R. Karp, U. Vazirani, and V. Vazirani. An optimal algorithm for online bipartite matching. In *In Proceedings of the 22nd Annual ACM Symposium on Theory of Computing*, pages 352–358, 1990.

[72] J. Kleinberg, E. Tardos, and Y. Rabani. Fairness in routing and load balancing. In *FOCS '99: Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, page 568, 1999.

[73] S. G. Kolliopoulos and N. E. Young. Tight approximation results for general covering integer programs. In *IEEE Symposium on Foundations of Computer Science*, pages 522–528, 2001.

[74] E. Koutsoupias and C. H. Papadimitriou. On the k-server conjecture. *J. ACM*, 42(5):971–983, 1995.

[75] A. Kumar and J. M. Kleinberg. Fairness measures for resource allocation. In *IEEE Symposium on Foundations of Computer Science*, pages 75–85, 2000.

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

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

[78] M. Mahdian and A. Saberi. Multi-unit auctions with unknown supply. In *EC '06: Proceedings of the 7th ACM conference on Electronic commerce*, pages 243–249, 2006.

[79] L. A. McGeoch and D. D. Sleator. A strongly competitive randomized paging algorithm. *Algorithmica*, 6(6):816–825, 1991.

[80] A. Mehta, A. Saberi, U. Vazirani, and V. Vazirani. Adwords and generalized online matching. *J. ACM*, 54(5):22, 2007.

[81] A. Meyerson. The parking permit problem. In *FOCS '05: Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science*, pages 274–284, 2005.

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

[83] P. Raghavan. Probabilistic construction of deterministic algorithms: approximating packing integer programs. *J. Comput. Syst. Sci.*, 37(2):130–143, 1988.

[84] P. Raghavan and C. Thompson. Randomized rounding: A technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7:365–374, 1987.

[85] S. S. Seiden. A general decomposition theorem for the k-server problem. In *ESA '01: Proceedings of the 9th Annual European Symposium on Algorithms*, pages 86–97, 2001.

[86] D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Commun. ACM*, 28(2):202–208, 1985.

[87] A. Srinivasan. Improved approximation guarantees for packing and covering integer programs. *SIAM Journal on Computing*, 29(2):648–670, 1999.

[88] V. V. Vazirani. *Approximation Algorithms*. Springer, 2001.

[89] N. Young. On-line caching as cache size varies. In *SODA '91: Proceedings of the second annual ACM-SIAM symposium on Discrete algorithms*, pages 241–250, 1991.

[90] N. E. Young. The k-server dual and loose competitiveness for paging. *Algorithmica*, 11(6):525–541, 1994.

[91] N. E. Young. Randomized rounding without solving the linear program. In *SODA '95: Proceedings of the sixth annual ACM-SIAM symposium on Discrete algorithms*, pages 170–178, 1995.

[92] N. E. Young. On-line file caching. In *SODA '98: Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms*, pages 82–86, 1998.