ABSTRACT

Title of dissertation: PRIMAL-DUAL ALGORITHMS FOR COMBINATORIAL OPTIMIZATION PROBLEMS

Julián Mestre
Doctor of Philosophy, 2007

Dissertation directed by: Professor Samir Khuller
Department of Computer Science

Combinatorial optimization problems such as routing, scheduling, covering and packing problems abound in everyday life. At a very high level, a combinatorial optimization problem amounts to finding a solution with minimum or maximum cost among a large number of feasible solutions. An algorithm for a given optimization problem is said to be *exact* if it always returns an optimal solution and is said to be *efficient* if it runs in time polynomial on the size of its input. The theory of NP-completeness suggests that exact *and* efficient algorithms are unlikely to exist for the class of NP-hard problems. Unfortunately, a large number of natural and interesting combinatorial optimization problems are NP-hard.

One way to cope with NP-hardness is to relax the optimality requirement and instead look for solutions that are provably close to the optimum. This is the main idea behind approximation algorithms. An algorithm is said to be a ρ-approximation if it always returns a solution whose cost is at most a ρ factor away from the optimal cost.

Arguably, one of the most important techniques in the design of combinatorial algorithms is the primal-dual schema in which the cost of the primal solution is compared to the cost of a dual solution. In this dissertation we study the primal-dual schema in the design of approximation algorithms for a number of covering and scheduling problems.

PRIMAL-DUAL ALGORITHMS FOR COMBINATORIAL OPTIMIZATION PROBLEMS

by

Julián Mestre

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2007

Advisory Committee:
Professor Samir Khuller, Chair/Advisor
Professor David Mount
Professor William Gasarch
Professor Louiqa Raschid
Professor Subramanian Raghavan

DEDICATION

To my parents.

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my advisor, Professor Samir Khuller. His constant guidance and encouragement have made my graduate years at Maryland an enjoyable, enriching, and exciting experience. It was my great fortune to be able to work under his supervision. He is an excellent teacher, an insightful researcher, and a model of rectitude in academic matters and otherwise. These are just a few of the many characteristics that I greatly admire about him, and that I myself hope one day to be able to match.

I would like to thank Professors David Mount, William Gasarch, Louiqa Raschid and Subramanian Raghavan for serving on my thesis examination committee. Their thoughtful comments helped improve the presentation of this dissertation.

I was very lucky to be part of a friendly Department where I was able to find many good friends among the other graduate students. Special thanks go to Rajiv Gandhi for encouraging me to persevere at the beginning of my studies when graduation seemed like an unattainable goal, and to my office mates Maryam Farboodi, MohammadReza Ghodsi, Srinivas Kashyap, and Azarakhsh Malekian for endless discussions and in general for putting up with my constant chatter throughout these years.

The friendly atmosphere of the Computer Science Department goes well beyond the student body and can also be felt in the administrative staff. I owe my gratitude to Gwen Kaye, who was specially helpful and supportive during my first years in graduate school—her retirement certainly was a sad moment for everyone. Also thanks to Fatima Bangura, Sue Blanford and Kathy MacLeod who always handled my administrative matters with a smile.

The work in this dissertation was done under the support of the NSF grant CCF-0430650 and the Ann G. Wylie Dissertation Fellowship, which I am duly grateful for.

This thesis is dedicated to my parents, Susana and Carlos, who always encouraged me to be independent and supported my plans to study abroad from the very beginning. Also thanks to my mother-in-law, Ludmila, for her copious words of support.

Finally, I want thank my wife, Elena, who has made all the difference in my life.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Combinatorial Optimization

Combinatorial optimization problems such as routing, scheduling, covering and packing problems abound in everyday life. At a very abstract level, a combinatorial optimization problem can be defined as a pair $(\mathcal{F}, c)$ where $\mathcal{F}$ is a set of feasible solutions and $c : \mathcal{F} \to R^+$ is a cost function. For minimization problems, a solution $F \in \mathcal{F}$ is said to be *optimal* if $c(F) \leq c(F')$ for all $F' \in \mathcal{F}$. Likewise, for maximization problems, a solution $F \in \mathcal{F}$ is optimal if $c(F) \geq c(F')$ for all $F' \in \mathcal{F}$.

An algorithm $\mathcal{A}$ for an optimization problem $(\mathcal{F}, c)$ is said to be *exact* if $\mathcal{A}$ returns an optimal feasible solution. Of course, a trivial algorithm is to exhaustively consider every solution in $\mathcal{F}$ and keep the one with minimum or maximum cost. This is commonly not an option since $\mathcal{F}$ is usually given implicitly and $|\mathcal{F}|$ grows exponentially on the size of the string encoding the problem instance. To make this discussion concrete consider the following balancing problem: Given a set A of n integers, find a subset X such that X and $A \setminus X$ are as balanced as possible; in other words, $\mathcal{F}$ is the power set of A and $c(X) = |\sum_{a \in X} a - \sum_{a \in A \setminus X} a|$. Note that even though the input is a list of n numbers there are 2^n feasible solutions; as a result, exhaustive search becomes prohibitively slow even for small values of n. Therefore, from a practical point of view, it is crucial to have efficient algorithms that avoid exhaustive search.

An algorithm $\mathcal{A}$ is said to be *efficient* if its running time is polynomial on the size of its input. The theory of NP-completeness provides strong evidence that exact *and* efficient algorithms are unlikely to exist for the class of NP-hard problems—such algorithms would imply efficient algorithms for any problem whose solution can be *verified* efficiently. Moreover, the class of NP-

hard problems contains a huge number of practical, and sometimes deceivingly simple-looking, problems such as the above balancing problem.

Therefore, when dealing with NP-hard problems in order to obtain efficient algorithms we must give up optimality. An algorithm $\mathcal{A}$ for a given minimization problem $(\mathcal{F}, c)$ is said to be a *ρ-approximation* if it runs in polynomial time and always finds a solution $F \in \mathcal{F}$ such that $c(F) \leq \rho\, c(F')$ for all $F' \in \mathcal{F}$.

Arguably, one of the most important techniques in the design of combinatorial algorithms is the primal-dual schema [46, 91] in which the cost of the primal solution is compared to the cost of a dual solution. This dissertation is mainly concerned with the design of efficient approximation algorithms for combinatorial optimization problems. The unifying theme of our results is the sophisticated use of the primal-dual schema.

The rest of this chapter is organized as follows. Section 1.2 introduces the primal-dual schema. Section 1.3 presents our techniques. Finally, Sections 1.4 and 1.5 review the literature and outline our contributions in two application areas: partial cover and data migration.

1.2 Primal-dual schema

At the heart of the primal-dual schema is the theory of linear programming. A linear program is an optimization made up of a system of linear inequalities and a linear objective function:

$$\begin{aligned} \min\ & c \cdot x \\ Ax &\geq b \\ x &\geq 0 \end{aligned}$$

Here $A \in R^{n \times m}$ is a matrix and $c, x \in R^m$ are column vectors. The expression $c \cdot x$ denotes the dot product of c and x, that is, $c \cdot x = c^T x$.

A key realization is that optimization problems come in pairs—every minimization problem

having a maximization counterpart, and vice versa.

$$\max \ b \cdot y$$

$$A^T y \leq c$$

$$y \geq 0$$

The above linear program is the dual problem of the original, primal, problem. The relation between these two programs is captured by the following theorems, the proof of which can be found in any linear programming textbook [26, 30, 87, 97].

Theorem 1.1 (Weak duality)**.** *Let x be a feasible solution for the primal problem and y be a feasible solution for the dual problem, then $c \cdot x \geq b \cdot y$.*

Theorem 1.2 (Strong duality)**.** *Let x^* be an optimal solution for the primal problem and y^* be an optimal solution for the dual problem, then $c \cdot x^* = b \cdot y^*$.*

This is excellent news for algorithm design. Namely, the dual problem offers a tight lower bound for the primal problem. In addition to being each other's certificate of optimality, a pair of optimal primal and dual solutions are closely related via complementary slackness.

Theorem 1.3 (Complementary slackness)**.** *Let x^* and y^* be feasible primal and dual solutions respectively. Then x^* and y^* are optimal if and only if $x^* \cdot (A^T y - c) \geq 0$ and $y^* \cdot (Ax^* - b) \leq 0$.*

Complementary slackness is also the driving force of the primal-dual schema for the design of exact algorithms. Essentially, it constitutes a way of reducing a weighted instance to an unweighted instance, which in general is easier to deal with. The method was first used by Kuhn [78] to design an exact and efficient algorithm for the assignment problem. A thorough treatment of the primal-dual schema for exact algorithms can be found in the book of Papadimitriou and Steiglitz [91]. The main focus of this dissertation, however, is not exact but approximation algorithms.

At a very high level a primal-dual ρ-approximation for a given minimization problem works as follows. First, we formulate the optimization problem as an integer program, then we relax the integrality constrains to get a linear program and derive its dual. Note that, by Theorem 1.1, any solution to the dual program provides a lower bound for the original optimization problem. Our

algorithm constructs a dual solution, and then, guided by this process, builds an integral primal solution. In the analysis the cost of primal solution is shown to be at most ρ times the cost of the dual solution, where $\rho \geq 1$. The reason why in general we cannot have $\rho = 1$ is that by relaxing the integrality constraints of the primal program we may be enlarging the feasible region, thus an optimal fractional solution may be cheaper than an optimal integral solution, and by Theorems 1.1 and 1.2, the dual solution cannot offer a lower bound better than the optimal (primal) fractional solution. In other words ρ must be at least as large as the integrality gap of the integer program formulation, which is defined as the ratio of the cost of an optimal integral solution to the cost of an optimal fractional solution.

In the same way complementary slackness is used in the primal-dual schema for exact algorithms, a relaxed version thereof is used in the design of approximation algorithms.

Theorem 1.4. *Let x and y be feasible primal and dual solutions respectively satisfying the relaxed complementary slackness conditions $x \cdot (\alpha A^T y - c) \geq 0$ and $y \cdot (Ax - \beta\, b) \leq 0$ for some $\alpha, \beta \geq 1$. Then $c \cdot x \leq \alpha\, \beta\ b \cdot y$.*

Proof. It follows immediately from the relaxed complementary slackness conditions

$$c \cdot x \leq \alpha\ x \cdot (A^T y) = \alpha\ y \cdot Ax \leq \alpha\, \beta\ b \cdot y$$

□

Recall that a primal-dual algorithm constructs an integral primal solution x and a fractional dual solution y. These two solution are related via relaxed complementary slackness and Theorem 1.4 is invoked in the analysis to show that x is a ρ-approximation where $\rho = \alpha\, \beta$.

1.3 Our techniques

The role that relaxed complementary slackness plays in the primal-dual schema for approximation algorithms is not as central as in exact algorithms. Rather than being the driving force behind the dual update and the construction of the primal solution, these two procedures must be designed, some times in an ad-hoc fashion, to meet the relaxed complementary slackness

conditions. As such, the primal-dual schema for approximation algorithms leaves more room for new algorithmic ideas. Each chapter of this dissertation deals with a different aspect of the basic primal-dual schema and linear programming duality. What follows is a short description of the techniques used throughout the dissertation; a more detailed description can be found in the Overview section opening each respective chapter.

- Lagrangian relaxation. This technique is closely related to linear programming duality and has been successfully applied to the design of approximation algorithms. In Chapter 2 we study the strengths and limitations of Lagrangian relaxation applied to partial cover.

- Making educated guesses. Most algorithms for partial cover problems require that we guess some attribute of the optimal solution, modify the instance accordingly, run a basic algorithm on each guess and finally return the best solution found. In Chapter 3 we show how to speed up certain primal-dual algorithms for partial cover by reusing the work done in the dual update and making guesses along the way.

- Beyond primal complementary slackness. Usually, the integer linear formulation for the primal problem uses binary variables and after the dual update is carried out, a primal solution is built by setting to 1 a subset of those variables whose dual constraint is tight. In Chapter 4 we study a scheduling problem where the primal solution is constructed using a more sophisticated method.

- Adaptive local ratio. There is a close relation between the local ratio technique and the primal-dual schema. Local ratio algorithms decompose the input weight function into a positive linear combination of simpler weight functions called models. Based on this decomposition, a primal solution is constructed so that it is ρ-approximate with respect to every model. These models typically have a very simple structure that remains "unchanged" throughout the execution of the algorithm. In Chapter 5 we show that adaptively choosing a model from a richer spectrum of functions can lead to better approximations.

- Dual fitting. This is a technique used to bound the approximation ratio of a given heuristic.

The underlying idea is very simple: To show that the solution output by the algorithm is ρ-approximate in the analysis we construct a dual infeasible solution that violates the dual constraints by at most a ρ factor, but has enough cost to pay for the primal solution. In Chapter 6 we apply dual fitting to analyze an algorithm for a scheduling problem.

The problems to which we apply these techniques are drawn from two main application areas: partial cover and data migration. The next two sections introduce the main problems in these areas, review previous work and outline our contributions.

1.4 Partial cover

The input for a covering problem consists of a collection $\mathcal{S}$, which may be given implicitly or explicitly, of subsets of a universal set U and a cost function $c : \mathcal{S} \to R_+$. The set $\mathcal{C} \subseteq \mathcal{S}$ is said to *cover* an element j in U if j belongs to some subset in $\mathcal{C}$; we say $\mathcal{C}$ is a *cover* if every element j in U is covered by $\mathcal{C}$. The objective is to find a minimum cost cover. Alternatively, a covering problem can be specified by element-set incidence matrix $A = \{a_{ij}\}$ where $a_{ij} = 1$ if and only if the ith element of U belongs to the jth set in $\mathcal{S}$.

As numerous surveys [4, 21, 40, 61, 89] on the subject attest, much work has been done on covering problems because of both their simple and elegant formulation, and their pervasiveness in different application areas. In its most general form the problem, also known as the *set cover* problem, cannot be approximated within $(1-\epsilon)\ln|U|$ unless $NP \subseteq \text{DTIME}(|U|^{\log\log|U|})$ [32], and a simple greedy algorithm is a $\ln|U|$ approximation [25, 67, 80]. Due to this hardness, special, easier, cases have been studied.

The most general class of covering problems that can be solved efficiently are those whose element-set incidence matrix is balanced. A $0,1$ matrix is *balanced* if it does not contain a square submatrix of odd order with row and column sums equal to 2. These matrices were introduced by Berge [15] who showed that if A is balanced then the polyhedron $\{x \geq 0 : Ax \geq 1\}$ is integral. A $0,1$ matrix is *totally balanced* if it does not contain a square submatrix with row and column sums equal to 2 and no identical columns. Kolen [75] gave a simple primal-dual algorithm that solves

optimally the covering problem defined by a totally balanced matrix. A $0, \pm 1$ matrix is *totally unimodular* if every square submatrix has determinant 0 or ± 1. Totally balanced and totally unimodular matrices are subclasses of balanced matrices; the two classes are neither disjoint nor one is included in the other. For more results on balanced matrices, the reader is referred to the excellent survey of Conforti et al. [29].

Beyond this point, even minor generalizations can make our covering problem hard. For example, consider the *vertex cover* problem: Given a graph $G = (V, E)$ we are to choose a minimum size subset of vertices such that every edge is incident on at least one of the chosen vertices. This captures *precisely* the special case of set cover where each element belongs to two sets. If G is bipartite, the problem can be written as a totally unimodular matrix; however, if G is a general graph the problem becomes NP-hard. Indeed, it was one of the first problems shown to be NP-hard in Karp's seminal paper [69] on computational complexity. Numerous approximation algorithms have been developed for it [60], and the best known approximation factor for general graphs is $2 - o(1)$ [11, 53, 54, 68]. However, after 25 years of study, the best constant factor approximation for vertex cover remains 2 [10, 27, 58]. On the negative side, it is NP-hard to approximate vertex cover within any factor smaller than $10\sqrt{5} - 21 \approx 1.36$ [31], and conditional on the stronger assumption of the Unique Games Conjecture [70], vertex cover cannot be approximated within $2 - \epsilon$ [71].

The lack of progress in approximating vertex cover has led researchers to seek generalizations of this basic problem that can still be approximated within twice of optimum. Two such generalizations are multicut on trees and capacitated vertex cover. The input of the *multicut* problem on trees is a tree T and a collection of pairs of vertices; a cover is formed by a set of edges whose removal separates all pairs. The problem was first studied by Garg et al. [43] who gave an elegant primal-dual 2-approximation. Unlike most optimization problems that are easy to solve on trees, multicut is APX-hard even on unit cost stars, as it contains vertex cover as a special case. The input of the *capacitated vertex cover* problem is a graph $G = (V, E)$ and a capacity k_v for each vertex $v \in V$; unlike regular vertex cover a vertex v can cover only k_v edges incident on it, thus

enough copies of each vertex must be chosen to cover the edges assigned to them. Despite its seemingly abstract nature, capacitated vertex cover has applications in computational biology [49] and cellular network planning [14]. Guha et al. [49] developed a primal-dual 2-approximation for the problem. An LP rounding algorithm achieving the same approximation factor was later developed by Gandhi et al. [39]. The hard capacitated version, where there is a bound on how many copies of a vertex we can choose, was studied by Chuzhoy and Naor [24]. They showed that weighted vertex cover with hard capacities is as hard as set cover and gave a 3-approximation for the unweighted version. This was later improved by Gandhi et al. [38], who gave a 2-approximation.

A notable shortcoming of the standard set cover formulation is that certain hard-to-cover elements, also known as *outliers* [20], can render the optimal solution very expensive. For example, suppose a government planning office must decide where to open new facilities, e.g., fire stations. We would like every house to have a facility within a certain distance. Among a set of potential locations we must select a subset of locations to open facilities such that every house is covered. Because of sparsely populated areas, e.g., rural areas, it would be unreasonable to try to cover every house in the country. The number of facilities needed to cover *every* house will be much larger than the number needed to cover, say, 90% thereof. Motivated by the presence of outliers, the unit-profit partial version of a given covering problem calls for a collection of sets covering not all, but a specified number k of elements. Many 2-approximations are known for partial vertex cover [9, 17, 36, 59]. Partial multicut, also known as k-multicut, was recently studied independently of each other by Levin and Segev [79] and by Golovin et al. [47], who gave a $\frac{8}{3} + \epsilon$ approximation algorithm. In the general setting we are given a profit function $p : U \to R_+$ and a target coverage parameter P. The objective is to find a minimum cost collection of sets covering at least P profit. Könemann et al. [76] showed how to design a $\frac{4}{3}\alpha+\epsilon$ approximation for any partial covering problem using Lagrangian relaxation and an α-LMP[1] approximation as a black box. Their algorithm runs in time polynomial on $|U|, |\mathcal{S}|^{\frac{1}{\epsilon}}$ and the running time of the α-LMP approximation.

[1]The definition of an α-LMP approximation is somewhat technical, so it is deferred until Section 2.2.

1.4.1 Our contributions

In Chapter 2 we show a lower bound of $\frac{4}{3}\alpha$ on the approximation ratio that can be achieved using Lagrangian relaxation and an α-LMP as a black box, thus matching the upper bound of Könemann et al. [76]. Then we study the integrality gap of the standard linear relaxation for partial totally balanced cover. We show that $\mathrm{IP} \leq \left(1 + \frac{1}{3^{k-1}}\right) \mathrm{LP} + k\, c_{\max}$ for any $k \geq 1$, and that there are instances where $\mathrm{IP} > \left(1 + \frac{1}{3^{k-1}}\right) \mathrm{LP} + \frac{k}{2} c_{\max}$, where IP and LP denote the cost of the optimal integral and fractional solutions respectively and $c_{\max}$ is the cost of the most expensive set in the instance. Our approach is based on Lagrangian relaxation and Kolen's primal-dual algorithm. Finally, we show that this implies improved approximations for the partial version of a number of problems such as multicut and path hitting on trees, rectangle stabbing, and set cover with ρ-blocks. These results appear in [85].

In Chapter 3 we present a technique to speed up algorithms for partial covering problems. We obtain 2-approximations for partial vertex cover and partial capacitated vertex cover with unit profits. For the latter problem, this is the first known 2-approximation. Both algorithms run in $O(m + n \log n)$ time. These results appear in [84]. Very recently, the same the technique has been successfully applied by Bar-Yehuda et al. [14] within the local ratio framework.

1.5 Data Migration

The *data migration* problem arises in large storage systems, such as *Storage Area Networks* [72], where a dedicated network of disks is used to store multimedia data. As the data access pattern changes over time, the load across the disks needs to be rebalanced so as to continue providing efficient service. This is done by computing a new data layout and then "migrating" data to convert the initial data layout to the target data layout. While migration is being performed, the storage system is running suboptimally, therefore it is important to compute a data migration schedule that converts the initial layout to the target layout quickly.

This problem can be modeled as a *transfer graph* [74], in which the vertices represent the storage disks and an edge between two vertices u and v corresponds to a data object that must be

transferred from u to v, or vice-versa. Each edge has a processing time that represents the transfer time of a data object between the disks corresponding to the end points of the edge. An important constraint is that any disk can be involved in at most one transfer at any time.

Several variations of the data migration problem have been studied. These variations arise either due to different objective functions or due to additional constraints. One common objective function is to minimize the *makespan* of the migration schedule, i.e., the time by which all migrations complete. Coffman et al. [28] show that when the edges have unit processing times, the problem reduces to edge coloring of the transfer (multi)graph of the system. The best approximation algorithm known for minimum edge coloring [88] then yields an algorithm for data migration with unit edge processing times, whose makespan is $1.1\chi' + 0.8$, where χ' is the chromatic index of the graph. An asymptotic scheme that uses $(1+\epsilon)\chi' + O(\frac{1}{\epsilon})$ colors is also known [96]. Approximation algorithms are also developed [1, 50, 72, 73] for generalizations of the makespan minimization problem in which there are storage constraints on disks and constraints on how the data can be transferred.

The data migration problem has also been studied with the objective of minimizing the sum of weighted completion time over all storage disks. Kim [74] proved that the problem is NP-hard when edges have unit processing times and showed that Graham's list scheduling algorithm [48], when guided by an optimal solution to a linear programming relaxation, gives an approximation ratio of 3. Gandhi et al. [37] show that Kim's analysis is tight. When edges have arbitrary processing times Kim [74] gave a 9-approximate solution. Gandhi et al. [37] improved the approximation factor to 5.03. They present two algorithms each achieving an approximation ratio of 5.83 and show that combining the two solutions yields an approximation ratio of 5.03.

A problem related to the data migration problem is *open shop scheduling*. In this problem, we have a set of jobs, $\mathcal{J}$, and a set of machines $M_1, \ldots, M_m$. Each job $J_j \in \mathcal{J}$ consists of a set of m_j operations. For $1 \leq i \leq m_j$, operation $o_{j,i}$ has processing time $p_{j,i}$ and must be processed on $M_{\phi(j,i)}$. Each machine can process a single operation at any time, and two operations that belong to the same job cannot be processed simultaneously. Each job J_j has a positive weight, w_j and

the objective is to minimize the sum of weighted completion times of all jobs. This problem is a special case of the data migration problem [37]. The open shop scheduling problem has been studied in [18, 63, 94, 95].

There has also been interest in the study of the data migration problem with the objective to minimize the average completion time over all data transfers. This corresponds to minimizing the average edge completion time in the transfer graph. For arbitrary edge processing times, several constant factor approximation algorithms [35, 52, 74] are known with the best approximation factor being 7.682 [35]. For the case of unit-length processing times, Bar-Noy et al. [6] showed that the problem is NP-hard and gave a simple 2-approximation algorithm. When restricted to bipartite graphs, the latter problem becomes a variant of open shop scheduling in which the operations have unit processing times and the objective is to minimize the sum of completion times of operations; for this problem Gandhi et al. [37] give a 1.796-approximate solution that uses a sum coloring algorithm due to Halldórsson et al. [52].

1.5.1 Our contributions

In Chapter 4 we study the data migration problem with the objective to minimize the average completion time of the disks. We give a 3-approximation for unit-length transfers and a 5.83-approximation for transfers with arbitrary lengths. Both algorithms are primal-dual and constitute the first purely combinatorial approximations for these problems—all previously known algorithms for these problems require solving a large linear program, and are therefore not very practical. We also show that two natural greedy heuristics for the problem are not constant factor approximations.

In Chapter 5 we improve the approximation ratio for unit-length transfers to $1 + \phi$, where $\phi = \frac{1+\sqrt{5}}{2}$ is the Golden ratio. Our approach is based on a novel use of the local ratio and the factor-revealing LP techniques. First we cast the primal-dual algorithm from Chapter 4 as a local ratio algorithm and provide a family of instances showing the analysis given there is tight. To overcome these difficult instances we propose to adaptively choose a model minimizing the local

ratio and formulate the problem of finding such a model as a linear program. In the analysis we show that the models found using this linear program exhibit a local ratio better than that of the usual 0-1 models found in the literature [13]. To derive the worst-case local ratio of our scheme we formulate as a mathematical program the problem of building a worst-case instance maximizing the local ratio. Since we already use a linear program to guide our local-ratio algorithm, the resulting factor-revealing program is non-linear. Somewhat surprisingly, even though we cannot solve numerically the factor-revealing program, we are still able to prove a bound of $1 + \phi$ on its cost. Finally, we show that our analysis is tight by giving a family of instances in which our algorithm attains this ratio.

In Chapter 6 we study the data migration problem with the objective to minimize average completion time of the transfers. We define the notion of strongly minimal schedule and prove that this schedule is $\sqrt{2}$-approximate. Our analysis is almost tight as these schedules can be, in the worst case, a 1.375-factor away from optimum. We show how to compute strongly minimal schedules for bipartite graphs in polynomial time.

The results in Chapters 4 and 6 appears in [33], while those in Chapter 5 appear in [86].

Chapter 2

Lagrangian Relaxation

This chapter studies the strengths and limitations of Lagrangian relaxation applied to partial cover. We show that for partial cover in general no algorithm that uses Lagrangian relaxation and an α-LMP approximation as a black box can yield an approximation factor better than $\frac{4}{3}\alpha$. This matches the upper bound given by Könemann et al. [76].

Faced with this limitation we study a specific, yet broad class of covering problems: partial totally balanced cover. By carefully analyzing the inner workings of a known primal-dual LMP algorithm we are able to give a tight characterization of the integrality gap of the standard linear relaxation of the problem. As a result, we obtain improved approximations for the partial version of multicut and path hitting on trees, rectangle stabbing, and set cover with ρ-blocks.

2.1 Overview

Lagrangian relaxation has been used extensively in the design of approximation algorithms for a variety of problems such as TSP [56, 57], k-MST [3, 23, 41, 42], partial vertex cover [59], k-median [2, 19, 65] and MST with degree constraints [77]. In this chapter we study the strengths and limitations of Lagrangian relaxation applied to the partial cover problem. Let $\mathcal{S}$ be collection of subsets of a universal set U with cost $c : \mathcal{S} \to R_+$ and profit $p : U \to R_+$, and let P be a target coverage parameter. A set $\mathcal{C} \subseteq \mathcal{S}$ is a *partial cover* if the overall profit of elements covered by $\mathcal{C}$ is at least P. The objective is to find a minimum cost partial cover.

The high level idea behind Lagrangian relaxation is as follows. In an IP formulation for partial cover, the constraint enforcing that at least P profit is covered is *relaxed*: The constraint is multiplied by a parameter λ and lifted to the objective function. This relaxed IP corresponds, up to a constant factor, to the prize-collecting version of the underlying covering problem in which

there is no requirement on how much profit to cover but a penalty of $\lambda\, p(i)$ must be paid if we leave element $i \in U$ uncovered. An approximation algorithm for the prize-collecting version having the Lagrangian multiplier preserving (LMP) property[1] is used to obtain values λ_1 and λ_2 that are close together for which the algorithm produces solutions $\mathcal{C}_1$ and $\mathcal{C}_2$ respectively. These solutions are such that $\mathcal{C}_1$ is inexpensive but infeasible (covering less than P profit), and $\mathcal{C}_2$ is feasible (covering at least P profit) but potentially very expensive. Finally, these two solutions are combined to obtain a cover that is both inexpensive and feasible.

Roughly speaking there are two ways one can combine $\mathcal{C}_1$ and $\mathcal{C}_2$. One option is to treat the approximation algorithm for the prize-collecting version as a black box, only making use of the LMP property in the analysis. Another option is to focus on a particular LMP algorithm, exploiting additional structure that this algorithm may offer. Perhaps not surprisingly, the latter approach has yielded better approximation guarantees. For example, for k-median compare the 6-approximation of Jain and Vazirani [65] to the 4-approximation of Charikar and Guha [19]; for k-MST compare the 5-factor to the 3-factor approximation, both due to Garg [42].

Our results support the common belief regarding the inherent weakness of the black-box approach. First, we show a lower bound on the approximation factor achievable for Partial Cover in general using Lagrangian relaxation and the black-box approach that matches the recent upper bound of Könemann et al. [76]. To overcome this obstacle, we concentrate on Kolen's algorithm for prize-collecting totally balanced cover [75]. By carefully analyzing the algorithm's inner workings we identify structural similarities between $\mathcal{C}_1$ and $\mathcal{C}_2$, which we later exploit when combining the two solutions. As a result we derive an almost tight characterization of the integrality gap of the standard linear relaxation for partial totally balanced cover. This in turn implies improved approximation algorithms for a number of related problems.

The rest of the chapter is organized as follows. Section 2.3 shows that for partial cover in general no algorithm that uses Lagrangian relaxation and an α-LMP approximation as a black box can yield an approximation factor better than $\frac{4}{3}\alpha$. Section 2.4 gives an almost tight char-

[1]The definition of the LMP property is outlined in Section 2.2.

acterization of the integrality gap of the standard LP for partial totally balanced cover, which settles a question posed by Golovin et al. [47]. Our approach is based on Lagrangian relaxation and Kolen's algorithm. We show that IP $\leq \left(1 + \frac{1}{3^{k-1}}\right)$ LP $+ k\, c_{\max}$ for any $k \geq 1$, where IP and LP are the costs of the optimal integral and fractional solutions respectively and $c_{\max}$ is the cost of the most expensive set in the instance. The trade-off between additive and multiplicative error is not an artifact of our analysis or a shortcoming of our approach. On the contrary, this is precisely how the integrality gap behaves. More specifically, we show a family of instances where IP $> \left(1 + \frac{1}{3^{k-1}}\right)$ LP $+ \frac{k}{2} c_{\max}$. In other words, there is an unbounded additive gap in terms of $c_{\max}$ but as it grows the multiplicative gap narrows exponentially fast.

Finally, in Section 2.5 we show how the above result can be applied, using ideas from [44, 47, 55], to get a $\rho + \epsilon$ approximation or a quasi-polynomial time ρ-approximation for covering problems that can be expressed with a suitable combination of ρ totally-balanced matrices. This translates into improved approximations for a number of problems: a $2 + \epsilon$ approximation for the partial multicut on trees [47, 79], a $4 + \epsilon$ approximation for partial path hitting on trees [92], a 2-approximation for partial rectangle stabbing [44], and a ρ approximation for partial set cover with ρ-blocks [55]. In addition, the ϵ can be removed from the first two approximation guarantees if we allow quasi-polynomial time. It is worth noting that prior to our work, the best approximation ratio for all these problems could be achieved with the framework of Könemann et al. [76]. In each case our results improve the approximation ratio by a $\frac{4}{3}$ multiplicative factor.

2.2 Partial cover and Lagrangian relaxation

Let $\mathcal{S} = \{1, \ldots, m\}$ be a collection of subsets of a universal set $U = \{1, \ldots, n\}$. Each set has a cost specified by $c \in R_+^m$, and each element has a profit specified by $p \in R_+^n$. Given a target coverage P, the objective of the partial cover problem is to find a minimum cost solution $\mathcal{C} \subseteq \mathcal{S}$ such that $p(\mathcal{C}) \geq P$, where the notation $p(\mathcal{C})$ denotes the overall profit of elements covered by $\mathcal{C}$. The problem is captured by the IP below. Matrix $A = \{a_{ij}\} \in \{0,1\}^{n \times m}$ is an element-set incidence matrix, i.e., $a_{ij} = 1$ if and only if element $i \in U$ belongs to set $j \in \mathcal{S}$; variable x_j indicates

whether set j is chosen in the solution $\mathcal{C}$; variable r_i indicates whether element i is left uncovered.

$$\begin{aligned} \min\ & c \cdot x \\ Ax + Ir &\geq 1 \\ p \cdot r &\leq p(U) - P \\ r_i, x_j &\in \{0,1\} \end{aligned}$$

Lagrangian relaxation is used to get rid of the constraint bounding the profit of uncovered elements to be at most $p(U) - P$. The constraint is multiplied by a parameter λ, called a Lagrange multiplier, and is lifted to the objective function. The resulting IP corresponds, up to the constant $\lambda\,(p(U) - P)$ factor in the objective function, to the prize-collecting version of the covering problem, where the penalty for leaving element i uncovered is λp_i.

$$\begin{aligned} \min\ & c \cdot x + \lambda p \cdot r - \lambda\,(p(U) - P) \\ Ax + Ir &\geq 1 \\ r_i, x_j &\in \{0,1\} \end{aligned}$$

Let OPT be the cost of an optimal partial cover and OPT-PC(λ) be the cost of an optimal prize-collecting cover for a given λ. Let $\mathcal{A}$ be an α-approximation for the prize-collecting variant of the problem. Algorithm $\mathcal{A}$ is said to have the Lagrangian multiplier preserving (LMP) property if it produces a solution $\mathcal{C}$ such that

$$c(\mathcal{C}) + \alpha\,\lambda\big(p(U) - p(\mathcal{C})\big) \leq \alpha\,\text{OPT-PC}(\lambda). \tag{2.1}$$

Note that OPT-PC(λ) $\leq$ OPT $+\ \lambda\,(p(U) - P)$. Thus,

$$c(\mathcal{C}) \leq \alpha\Big(\text{OPT} + \lambda\,\big(p(\mathcal{C}) - P\big)\Big). \tag{2.2}$$

Therefore, if we could find a value of λ such that $\mathcal{C}$ covers exactly P profit then $\mathcal{C}$ is α-approximate. However, if $p(\mathcal{C}) < P$, the solution is not feasible, and if $p(\mathcal{C}) > P$, equation (2.2) does not offer any guarantee on the cost of $\mathcal{C}$. Unfortunately, there are cases where no value of λ produces a solution covering exactly P profit. Thus, the idea is to use binary search to find two

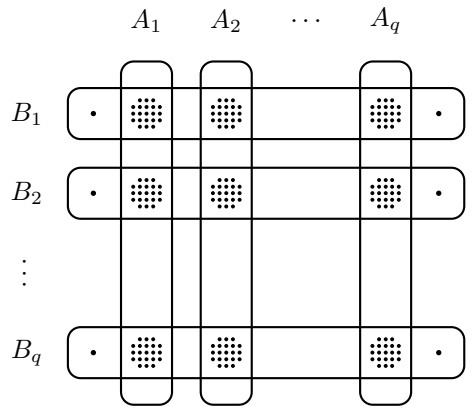


Figure 2.1: Lower bound instance for the black-box approach.

values λ_1 and λ_2 that are close together and are such that $\mathcal{A}(\lambda_1)$ covers less, and $\mathcal{A}(\lambda_2)$ covers more than P profit. The two solutions are then combined in some fashion to produce a feasible cover.

2.3 Limitations of the black-box approach

A common way to combine the two solutions returned by the α-LMP is to treat the algorithm as a black box, solely relaying on the LMP property (2.1) in the analysis. More formally, an algorithm for partial cover that uses Lagrangian relaxation and an α-LMP approximation $\mathcal{A}$ as a black box is as follows. First, we are allowed to run $\mathcal{A}$ with as many different values of λ as desired; then, the solutions thus found are combined to produce a feasible partial cover. No computational restriction is placed on the second step, except that only sets returned by $\mathcal{A}$ may be used.

Theorem 2.1. *In general, the partial cover problem cannot be approximated better than $\frac{4}{3}\alpha$ using Lagrangian relaxation and an α-LMP algorithm $\mathcal{A}$ as a black box.*

Let $A_1, \ldots A_q$ and $B_1, \ldots B_q$ be sets as depicted in Figure 2.1. For each i and j the intersection $A_i \cap B_j$ consists of a cluster of q elements. There are q^2 clusters. Set A_i is made up of q clusters; set B_i is made up of q clusters and two additional elements (the leftmost and rightmost elements in Figure 2.1.) Thus $|A_i| = q^2$ and $|B_i| = q^2 + 2$. In addition, there are sets $O_1, \ldots, O_q$, which are not shown in the picture. Set O_i contains one element from each cluster and the leftmost

element of B_i. Thus $|O_i| = q^2 + 1$. The cost of O_i is $\frac{1}{q}$, the cost of A_i is $\frac{2\,\alpha}{3\,q}$, and the cost of B_i is $\frac{4\,\alpha}{3\,q}$. Every element has unit profit and the target coverage is $P = q^3 + q$.

It is not hard to see that $O_1, \ldots, O_q$ is an optimal partial cover with a cost of 1. Furthermore, for any value of λ the optimal prize-collecting cover uses sets of one kind.

Lemma 2.1. *For all values of λ, the optimal prize-collecting cover is either the empty solution or $A_1, \ldots, A_q$ or $B_1, \ldots, B_q$ or $O_1, \ldots, O_q$.*

Proof. Suppose that some subset of the B-sets and the O-sets have already been chosen. Every A-set has the same marginal benefit independent of the other A-sets. Thus, in an optimal solution either all the A-sets are chosen or none is.

Now suppose that the O-sets and the A-sets have already been chosen. In this case there are two marginal benefits for a B-set, depending on whether an O-set already covers the leftmost element of the set or not. Thus, an optimal strategy for the B-sets is either to choose none, all, or the complement of the O-sets, i.e., B_i is chosen if and only if O_i is not chosen. A solution where B-sets and O-sets complement each other is always worse than either choosing all the O-sets and no B-sets, or vice versa. Thus, in an optimal solution either all the B-sets are chosen or none is.

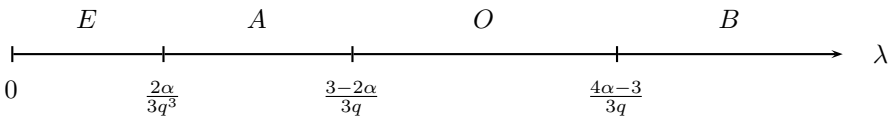
It follows that in an optimal solution O-sets are either all in or all out. Furthermore, if the B-sets are chosen then there is no reason to choose any of the remaining sets. If the B-sets are not chosen and the O-sets are chosen then there is no reason to choose any of the remaining sets. The only possibility left are to choose only the A-sets, or the empty cover. □

The α-LMP approximation algorithm we use has the unfortunate property that it never returns sets from the optimal partial cover.

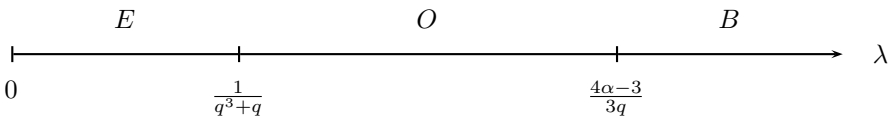
Lemma 2.2. *There exists an α-LMP approximation $\mathcal{A}$ that for the above instance and any value of λ outputs either $\emptyset$ or $A_1, \ldots, A_q$ or $B_1, \ldots, B_q$.*

Proof. Let us denote each of these four alternatives by E, A, B and O from Lemma 2.1. The prize-collecting cost of these covers is $(q^3 + 2q)\lambda$, $\frac{2}{3}\alpha + 2q\,\lambda$, $\frac{4}{3}\alpha$ and $1 + q\,\lambda$ respectively. For a fixed value of λ, the minimum of these four quantities corresponds to the cost of the optimal

prize-collecting cover. The line below shows which solution is optimal as a function of λ.



If α is big enough, the A-interval disappears. Namely, if $\frac{2\alpha}{3q^3} > \frac{3-2\alpha}{3q}$, the line looks as follows:



We are now ready to describe the α-LMP algorithm. Given a value of λ, we need to decide whether to output the empty cover, the A-sets or the B-sets. If λ falls in the interval corresponding to one of these three solutions then output that solution; since the cover output is optimal, the LMP property (2.1) follows trivially.

If λ falls in the O-interval and $\lambda \leq \frac{1}{3q}$ then output the A-sets; the LMP property holds since

$$c(A) + \alpha\,\lambda\,\overline{p(A)} = \frac{2}{3}\alpha + \alpha\,\lambda\,2q \leq \alpha(1+\lambda\,q) = \alpha\left(c(O) + \lambda\,\overline{p(O)}\right).$$

If λ falls in the O-interval and $\lambda > \frac{1}{3q}$ then output the B-sets; the LMP property holds since

$$c(B) + \alpha\,\lambda\,\overline{p(B)} = \frac{4}{3}\alpha < \alpha(1+\lambda\,q) = \alpha\left(c(O) + \lambda\,\overline{p(O)}\right).$$

□

Hence, if we use $\mathcal{A}$ from Lemma 2.2 as a black box we must build a partial cover with the sets $A_1, \ldots, A_q, B_1, \ldots, B_q$. Note that in order to cover $q^2 + q$ elements either all A-sets, or all B-sets must be used. In the first case $\frac{q}{2}$ additional B-sets are needed to attain feasibility, and the solution has cost $\frac{4}{3}\alpha$; in the second case the solution is feasible but again has cost $\frac{4}{3}\alpha$. Theorem 2.1 follows.

One assumption usually made in the literature [3, 34, 76] is that $c_{\max} = \max_j c_j \leq \epsilon\,\mathrm{OPT}$, for some constant $\epsilon > 0$, or more generally an additive error in terms of $c_{\max}$ is allowed. This does not help in our construction as $c_{\max}$ can be made arbitrarily small by increasing q.

Admittedly, our lower bound example belongs to a specific class of covering problem (every element belongs to at most three sets), and although the example can be embedded into a partial totally unimodular covering problem (see Theorem 2.2 below), it is not clear how to embed it into other classes of covering problems. Nevertheless, the $\frac{4}{3}\alpha$ upper bound of Könemann et al. [76] makes no assumption about the underlying covering problem, only using the LMP property in the analysis. It was entirely conceivable that the $\frac{4}{3}\alpha$ factor could be improved using a different merging strategy—Theorem 2.1 precludes this possibility.

Theorem 2.2. *Partial totally unimodular cover cannot be approximated better than $\frac{4}{3}$ using Lagrangian relaxation and a 1-LMP algorithm $\mathcal{A}$ as a black box.*

Proof. The instance is similar to that used in Theorem 2.1: The A-sets and the B-sets, given in Figure 2.1, are the same; for each i we define O_i as B_i minus the rightmost element. The cost of each A, B and O set is $\frac{2}{3}$, $\frac{4}{3}$ and 1 respectively. The target coverage parameter is again $P = q(q^2+1)$.

It is straightforward to check that Lemmas 2.1 and 2.2 still holds for our new instance and $\alpha = 1$. It only remains to show that the resulting element-set incidence matrix A is totally unimodular. A matrix A is totally unimodular if and only if every submatrix A' of A has an equitable coloring [45]. An equitable coloring of a 0,1 matrix A' is a partition of its columns into red and blue columns such that in every row of A' the number of blue 1's and red 1's differs by at most one. Let us construct an equitable coloring for A': all the A-sets are colored blue; for each i, if B_i and O_i are present in A' then color one red and the other blue, and if only one is present then color it red. Clearly the coloring is equitable; thus, A is totally unimodular. □

2.4 Partial totally balanced cover

In order to overcome the lower bound of Theorem 2.1, one must concentrate on a specific class of covering problems or make additional assumptions about the α-LMP algorithm. In this section we focus on covering problems whose IP matrix A is totally balanced. More specifically, we study the integrality gap of the standard linear relaxation for partial totally balanced cover

(P-TBC) shown below. Recall that these are problems defined by a 0-1 matrix that do not contain a square submatrix with row and column sums equal to 2 and no identical columns.

$$\begin{array}{lll} \min\ c \cdot x & & \max\ 1 \cdot y - (p(U) - P)\,\lambda \\ Ax + Ir \geq 1 & & A^T y \leq c \\ p \cdot r \leq p(U) - P & \xrightarrow{\text{LP Duality}} & y \leq \lambda p \\ r_i, x_e \geq 0 & & y_i, \lambda \geq 0 \end{array}$$

Theorem 2.3. *Let* IP *and* LP *be the cost of the optimal integral and fractional solutions of an instance of P-TBC. Then* $\text{IP} \leq \left(1 + \frac{1}{3^{k-1}}\right) \text{LP} + k\, c_{\max}$ *for any* $k \in Z_+$. *Furthermore, for any large enough* $k \in Z_+$ *the exists an instance where* $\text{IP} > \left(1 + \frac{1}{3^{k-1}}\right) \text{LP} + \frac{k}{2}\, c_{\max}$.

The rest of this section is devoted to proving Theorem 2.3. Our approach is based on Lagrangian relaxation and Kolen's algorithm for prize-collecting totally balanced cover (PC-TBC). The latter exploits the fact that a totally balanced matrix can be put into greedy standard form by permuting the order of its rows and columns; in fact, the converse is also true [62]. A matrix is said to be in standard greedy form if it does not contain as an induced submatrix

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \tag{2.3}$$

There are polynomial time algorithms that can transform a totally balanced matrix into greedy standard form [62, 81, 90, 99] by shuffling the rows and columns of A. Since this transformation does not affect the underlying covering problem, we assume that A is given in standard greedy form.

2.4.1 Kolen's algorithm for prize-collecting totally balanced cover

For the sake of completeness we describe Kolen's primal-dual algorithm for PC-TBC given in Figure 2.2. The algorithm finds a dual solution y and a primal solution C, which is then pruned in a reverse-delete step to obtain the final solution $\widehat{C}$. The linear and dual relaxations for PC-TBC appear below.

$$
\begin{array}{ll}
\min\ c \cdot x + \lambda p \cdot r & \\
Ax + Ir \geq 1 & \xrightarrow[\text{LP Duality}]{} \\
r_i, x_e \geq 0 &
\end{array}
\qquad
\begin{array}{l}
\max\ 1 \cdot y \\
A^T y \leq c \\
y \leq \lambda p \\
y_i \geq 0
\end{array}
$$

The residual cost of the set j w.r.t. y is defined as $c'_j = c_j - \sum_{i:a_{ij}=1} y_i$. The algorithm starts from the trivial dual solution $y = 0$, and processes the elements in increasing column order of A^T. Let i the index of the current element. Its corresponding dual variable, y_i, is increased until either the residual cost of some set j containing i equals 0 (we say set j becomes tight), or y_i equals λp_i (Lines 3-5).

```
KOLEN(A, c, p, λ)
 1  // Dual update
 2  y ← 0, C ← ∅, Ĉ ← ∅
 3  for i ← 1 to n do
 4      δ ← min{c'_j | a_ij = 1}
 5      y_i ← min{λp_i, δ}
 6  C ← {j | c'_j = 0}
 7  // Reverse delete
 8  while C ≠ ∅ do
 9      j ← largest set index in C
10      Ĉ ← Ĉ + j
11      C ← C \ { j' | j dominates j' or j = j' }
12  return (Ĉ, y)
```

Figure 2.2: Kolen's algorithm for partial totally-balanced cover.

Let $C = \{j \mid c'_j = 0\}$ be the set of tight sets after the dual update is completed. As it stands the cover C may be too expensive to be accounted for using the lower bound provided by $1 \cdot y$ because a single element may belong to multiple sets in C. The key insight is that some of the sets in C are redundant and can be pruned.

Definition 2.1. *Given sets j_1, j_2 we say that j_1* dominates *j_2 in y if $j_1 > j_2$ and there exists an item i such that $y_i > 0$ and i belongs to j_1 and j_2, that is, $a_{ij_1} = a_{ij_2} = 1$.*

The reverse-delete step iteratively identifies the largest index $j \in C$, adds it to $\widehat{C}$, and removes j and all the sets dominated by j. This is repeated until no set is left in C (Lines 8–11).

Notice that all sets $j \in C$ are tight, thus we can pay for set j by charging the dual variables of items that belong to j. Because of the reverse-delete step if $y_i > 0$ then i belongs to at most one set in $\widehat{C}$; thus in paying for $\widehat{C}$ we charge covered items at most once. Using the fact A is in standard greedy form, it can be shown [75] that if i was left uncovered then we can afford its penalty, i.e., $y_i = \lambda p_i$. The solution $\widehat{C}$ is optimal for PC-TBC since

$$\sum_{j \in \widehat{C}} c_j \;+ \sum_{\substack{i \in U \text{ s.t.} \\ \nexists\, j \in \widehat{C} \,:\, a_{ij}=1}} \lambda p_i \;= \sum_{\substack{i \in U \text{ s.t.} \\ \exists\, j \in \widehat{C} \,:\, a_{ij}=1}} y_i + \sum_{\substack{i \in U \text{ s.t.} \\ \nexists\, j \in \widehat{C} \,:\, a_{ij}=1}} y_i \;= \sum_{i \in U} y_i. \tag{2.4}$$

If we could find a value of λ such that $\textsc{Kolen}(A, c, p, \lambda)$ returns a solution $(\widehat{C}, y)$ covering *exactly* P profit, we are done since from (2.4) it follows that

$$\sum_{j \in \widehat{C}} c_j = \sum_{i \in U} y_i - \lambda\,(p(U) - P). \tag{2.5}$$

Notice that (y, λ) is a feasible for the dual relaxation of P-TBC and its cost is precisely the right hand side of (2.5). Therefore for this instance IP=DL=LP and Theorem 2.3 follows.

Unfortunately, as Section 2.4.4 shows, there are cases where no such value of λ exists. Nonetheless, we can always find a *threshold value* λ such that for any infinitesimally small $\delta > 0$, $\lambda^- = \lambda - \delta$ and $\lambda^+ = \lambda + \delta$ produce solutions covering less and more than P profit respectively.

2.4.2 Finding a threshold value

One way to find a threshold value is to do a binary search on λ in the interval $[0, c_{\max}/p_{\min}]$. This leads to pseudo-polynomial time algorithm. Below we show an algorithm that runs in strongly

polynomial time using Megiddo's parametric search [83] that makes $O(n \log m)$ calls to the procedure KOLEN.

The idea is to treat λ as an unknown that lies in a certain range (λ_l, λ_r). Initially $\lambda_l = 0$ and $\lambda_r = \max_{i,j} \frac{c_j}{p_i}$. Residual capacities and dual variables are kept as a linear function of λ. We maintain the invariant that λ_l^+ covers less than P profit and λ_r^- covers more than P profit. Suppose that in the interval (λ_l, λ_r) the algorithm *agrees* on the first q elements. By this we mean that if we run the algorithm with any value $\lambda \in (\lambda_l, \lambda_r)$ the value of the dual variables of these elements (as a function of λ) is the always the same. In each iteration we either find a threshold value or we narrow the interval such that the algorithm agrees on one more element. This cannot go on forever because the algorithm will eventually behave the same way throughout the interval and the invariant would be violated. If at any point along the way we find a value of λ covering exactly P profit we stop as the solution is optimal. For simplicity, from now on we assume that this never happens.

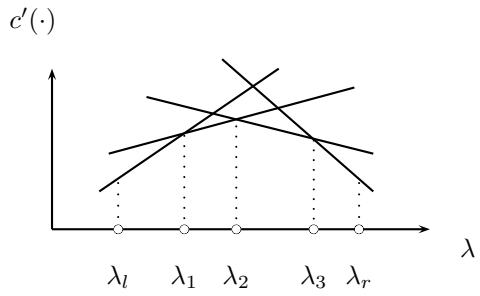


Figure 2.3: Narrowing the interval for λ.

Suppose that KOLEN agrees on the first $i-1$ elements in the interval (λ_l, λ_r). Note that the residual costs and i's penalty are linear functions of λ. As a result, which set has the minimum residual cost, and thus which one becomes tight, if any, varies with λ. Our goal is to narrow the interval such that the set that becomes tight is always the same, or $y_i = p_i\lambda$ within the new interval. If we draw the lines corresponding to the residual costs of set that i belongs to and $p_i\lambda$, the lower envelope corresponds to the next tight event, either a set or element i, see Figure 2.3. Let $\lambda_1, \ldots, \lambda_s$ correspond to the intersection points of the lower envelope, and $\lambda_0 = \lambda_l$ and $\lambda_{s+1} = \lambda_r$.

For every $0 \leq a \leq s$, within the interval $(\lambda_a, \lambda_{a+1})$ the algorithm agree on i. Note that either one of the λ_a is a threshold value, or there exists an a such that λ_a^+ covers less than P profit and λ_{a+1}^- covers more than P profit. Given the latter we update $\lambda_l = \lambda_a$ and $\lambda_r = \lambda_{a+1}$ and repeat.

Theorem 2.4. *A threshold value can be found by making $O\big(|U| \log |\mathcal{S}|\big)$ calls to* KOLEN.

Proof. When searching for the next set to become tight binary search can be used to find the right a to narrow the interval using $\log |\mathcal{S}| + 1$ calls to KOLEN. □

2.4.3 Merging two solutions

Let y (y^-) be dual solution and C (C^-) the set of tight sets when KOLEN is run on λ (λ^-). Without loss of generality assume $\widehat{C}$ covers more than P profit. (The case where $\widehat{C}$ covers less than P profit is symmetrical: we work with y^+ and C^+ instead of y^- and C^-.)

Our plan to prove Theorem 2.3 is to devise an algorithm to merge $\widehat{C}$ and $\widehat{C}^-$ in order to obtain a cheap solution covering at least P profit. Before presenting the algorithm we need to establish an important property regarding these two solutions.

Lemma 2.3. *For each $i \in U$ there exists $a \in Z$, independent of δ, such that $y_i = y_i^- + a\delta$.*

Proof. By induction, using the fact that the same property holds for the residual cost of the sets. □

A useful corollary of Lemma 2.3 is that $C^- \subseteq C$, since if the residual cost of a set is non-zero in y it must necessarily be non-zero in y^-. Notice that the converse of Lemma 2.3 does not hold in general.

At the heart of our approach is the notion of a merger graph $G = (V, E)$. The vertex set of G is made up of sets from the two solutions, i.e., $V = \widehat{C} \oplus \widehat{C}^-$. The edges of G are directed and given by

$$E = \left\{ (j_1, j_2) \;\middle|\; \begin{array}{l} j_1 \in \widehat{C}^- \setminus \widehat{C},\, j_2 \in \widehat{C} \setminus \widehat{C}^- \text{ s.t. } j_1 \text{ dominates } j_2 \text{ in } y^-, \text{ or} \\ j_1 \in \widehat{C} \setminus \widehat{C}^-,\, j_2 \in \widehat{C}^- \setminus \widehat{C} \text{ s.t. } j_1 \text{ dominates } j_2 \text{ in } y \end{array} \right\} \tag{2.6}$$

This graph has a lot of structure that can be exploited when merging the solutions.

Lemma 2.4. *The merger graph $G = (V, E)$ of $\widehat{C}^-$ and $\widehat{C}$ is a forest of out-branchings.*

Proof. First note that G is acyclic, since if $(j_1, j_2) \in E$ then necessarily $j_1 > j_2$. Thus, it is enough to show that the in-degree of every $j \in V$ is at most one. Suppose otherwise, that is, there exist $j_1, j_2 \in V$ such that $(j_1, j), (j_2, j) \in E$. Assume that $j_1 < j_2$ and $j \in \widehat{C}$ (other cases are symmetrical).

By definition (2.6), we know that $j_1\,(j_2) \in \widehat{C}^-$ and that there exists i_1 (i_2) that belongs to j and j_1 (j_2) such that $y^-_{i_1} > 0$ ($y^-_{i_2} > 0$). Since A^T is in standard greedy form we can infer that i_2 belongs to j_1 if $i_1 < i_2$, or i_1 belongs to j_2 if $i_1 > i_2$: The diagram on the right shows how, using the fact that A^T does not contain (2.3) as an induced submatrix, we can infer that the boxed entries must be 1. In either case we get that j_2 dominates j_1 in y^-, which contradicts the fact that both belong to $\widehat{C}^-$. □

	i_1	i_2	i_2	i_1
j	1	1	1	1
j_1	1	$\boxed{1}$		1
j_2		1	1	$\boxed{1}$

The procedure MERGE, given in Figure 2.4, starts from the infeasible solution $D = \widehat{C}^-$ and guided by the merger graph G, it modifies D step by step until feasibility is attained. The operation used to update D is to take the symmetric difference of D and a subtree of G rooted at a vertex $r \in V$, which we denote by T_r. For each root r of an out-branchings of G we set $D \leftarrow D \oplus T_r$, until $p(D \oplus T_r) > P$. At this point we return the solution produced by INCREASE(r, D).

Notice that after setting $D \leftarrow D \oplus T_r$ in Line 5, the solution D "looks like" $\widehat{C}$ within T_r. Indeed, if all roots are processed then $D = \widehat{C}$. Therefore, at some point we are bound to have $p(D \oplus T_r) > P$ and to make the call INCREASE(r, D) in Line 6. Before describing INCREASE we need to define a few terms. Let the *absolute benefit* of set j, which we denote by b_j, be the profit of elements uniquely covered by set j, that is,

$$b_j = p\left(\left\{\ i \in U\ \mid\ \forall\, j' \in \widehat{C} \cup \widehat{C}^- : a_{ij'} = 1 \text{ iff } j' = j\ \right\}\right). \tag{2.7}$$

Let $D \subseteq \widehat{C} \cup \widehat{C}^-$. Note that if $j \in D$, the removal of j decreases the profit covered by D by at least b_j; on the other hand, if $j \notin D$, its addition increases the profit covered by at least b_j. This

```
MERGE(Ĉ⁻, Ĉ)
1  let G be the merger graph for Ĉ⁻ and Ĉ
2  D ← Ĉ⁻
3  for each root r in G  do
4      if p(D ⊕ T_r) ≤ P
5          then D ← D ⊕ T_r
6          else return INCREASE(r, D)
```

Figure 2.4: Merging two solutions.

notion of benefit can be extended to subtrees,

$$\Delta(T_j, D) = \sum_{j' \in T_j \setminus D} b_{j'} - \sum_{j' \in T_j \cap D} b_{j'}. \tag{2.8}$$

We call this quantity the *relative benefit* of S with respect to D. It shows how the profit of uniquely covered elements changes when we take $D \oplus T_j$. Note that $\Delta(T_j, D)$ can positive or negative.

Everything is in place to explain INCREASE(j, D), whose pseudo-code is given in Figure 2.5. The algorithm assumes the input solution is infeasible (more precisely, $p(D) \leq P$) but can be made feasible by adding some sets in T_j, that is, $p(P) + \Delta(T_j, D) > P$. If adding j to D makes the solution feasible then return $D + j$ (Lines 2-3). If there exists a child c of j that can be used to propagate the call down the tree then do that (Lines 4-5). Otherwise, *split* the subtree T_j: Add j to D and process the children of c, setting $D \leftarrow D \oplus T_c$ until D becomes feasible (Lines 6-9). At this point $p(D) > P$ and $p(D \oplus T_c) \leq P$. If $P - p(D \oplus T_c) < p(D) - P$ then call INCREASE(c, D) else call DECREASE(c, D) and let D' be the cover returned by the recursive call (Line 10-12). Finally, return the cover with minimum cost between D and D'.

The twin procedure DECREASE(j, D) is essentially symmetrical: The input is feasible ($p(D) \geq P$) but can be made infeasible by removing some sets in T_j, that is, $p(D) + \Delta(T_j, D) < P$.

At a very high level, the intuition behind the INCREASE/DECREASE scheme is as follows. In

```
INCREASE(j, D)
1  // assume p(D) ≤ P < p(D) + Δ(T_j, D)
2  if p(D + j) ≥ P
3      then return D + j
4  if ∃ child c of j : p(D) + Δ(T_c, D) > P
5      then return INCREASE(c, D)
6  D ← D + j
7  while p(D) ≤ P do
8      c ← child of j maximizing Δ(T_c, D)
9      D ← D ⊕ T_c
10 if P − p(D ⊕ T_c) < p(D) − P
11     then D' ← INCREASE(c, D)
12     else D' ← DECREASE(c, D ⊕ T_c)
13 return min cost {D, D'}
```

```
DECREASE(j, D)
1  // assume p(D) ≥ P > p(D) + Δ(T_j, D)
2  if p((D ⊕ T_j) + j) ≥ P
3      then return (D ⊕ T_j) + j
4  if ∃ child c of j : p(D) + Δ(T_c, D) < P
5      then return DECREASE(c, D)
6  D ← D + j
7  while p(D) ≥ P do
8      c ← child of j minimizing Δ(T_c, D)
9      D ← D ⊕ T_c
10 if p(D ⊕ T_c) − P < P − p(D)
11     then D' ← INCREASE(c, D ⊕ T_c)
12     else D' ← DECREASE(c, D)
13 return min cost {D ⊕ T_c, D'}
```

Figure 2.5: Increase/Decrease procedures for reducing the deficiency/excess.

each call one of three things must occur:

(i) A feasible cover with a small coverage excess is found (Lines 2-3), or

(ii) The call is propagated down the tree at no cost (Lines 4-5), or

(iii) A subtree T_j is split (Lines 6-9). In this case, the cost c_j cannot be accounted for, but the offset in coverage $|P - p(D)|$ is reduced at least by a factor of 3.

If the INCREASE/DECREASE algorithms split many subtrees (incurring a high extra cost) then the offset in coverage must have been very high at the beginning, which means the cost of the dual solution is high and so the splitting cost can be charged to it. In order to flesh out these ideas into a formal proof we need to establish some crucial properties of the merger graph and the algorithms.

Lemma 2.5. *If $y_i < \lambda p_i$, there exist $j' \in \widehat{C}$ and $j'' \in \widehat{C}^-$ such that $j' = j''$ or $(j', j'') \in E$ or $(j'', j') \in E$.*

Proof. Since $y_i < \lambda p_i$, by Lemma 2.3 we get that $y_i^- < \lambda p_i$ as well. The way KOLEN does the dual update ensure the existence of a set $j \in C^-$ such that $a_{ij} = 1$. Because of the reverse-delete step either $j \in \widehat{C}^-$ or there exists $j_1 \in \widehat{C}^-$ that dominates j in y^- trough $i_1 \in U$, i.e., $y_{i_1} > 0$ and $a_{i_1 j} = a_{i_1 j_1} = 1$. Notice that, set j is tight after we process i; since $y_{i_1} > 0$ and $a_{i_1 j} = 1$ it follows that $i_1 \leq i$. Since A is in standard greedy form we infer that $a_{i j_1} = 1$. By Lemma 2.3, we have $C^- \subseteq C$, thus $j \in C$. A similar reasoning as above shows that either $j \in \widehat{C}$ or there exists $j_2 \in \widehat{C}$ that dominates j in y trough $i_2 \in U$ and $a_{i j_2} = 1$.

If there exists a set in $\widehat{C}^- \cap \widehat{C}$ covering i the lemma follows, so suppose otherwise. If $j \in \widehat{C}^-$ ($j \in \widehat{C}$) then there exists $j_2 \in \widehat{C}$ ($j_1 \in \widehat{C}^-$) that dominates j in y^- (y), and again the lemma holds. Finally, consider the case $j \notin \widehat{C}^-$ and $j \notin \widehat{C}$. Assume $j_1 < j_2$, the other case is symmetrical. Because A is in standard greedy form we get that $(j_2, j_1) \in E$ if $i_1 < i_2$ or $(j_1, j_2) \in E$ if $i_2 < i_1$. In either case the lemma follows. The diagram on above shows a summary of the entries that were inferred using the fact that A does not contain (2.3) as an induced submatrix. □

	i_1	i_2	i	i_2	i_1	i
j	1	1	1	1	1	1
j_1	1	[1]	[1]		1	[1]
j_2		1	[1]	1	[1]	[1]

Lemma 2.6. *At all times $\widehat{C}^- \cap \widehat{C} \subseteq D \subseteq \widehat{C}^- \cup \widehat{C}$, and for every arc in E at least one endpoint is in D.*

Proof. Before $D \leftarrow D \oplus T_c$ (Line 9), for every (j', j'') in T_c we have $|\{j', j''\} \cap D| = 1$ and $j \in D$. □

Lemma 2.7. *Let (j, D) be the input of* INCREASE/DECREASE. *For* INCREASE *we always have $p(D) \leq P < p(D) + \Delta(T_j, D)$, while for* DECREASE *we always have $p(D) \geq P > p(D) + \Delta(T_j, D)$.*

Proof. We show by induction on the number of calls that the property holds. At the base case the call is made by MERGE. Here we have $p(D) \leq P < p(D \oplus T_j)$, we claim that

$$p(D \oplus T_j) - p(D) = \Delta(T_j, D), \tag{2.9}$$

from which the property follows. We argue that for each $i \in U$ its contribution to both sides of (2.9) is the same. If $y_i < \lambda p_i$ then by Lemmas 2.5 and 2.6 element i is covered by both D and $D \oplus T_j$, thus its contribution to (2.9) is nil. The same is true if $y_i = \lambda p_i$, and i is covered by $\widehat{C}^-$ and $\widehat{C}$. In the remaining cases i is uniquely covered, thus its contribution is also the same on both sides.

For the inductive step suppose the lemma holds in the this call. Clearly, if the next call is made in Line 3 the lemma holds. Suppose that the call is made in Line 11-12. Note that after adding j to D (Line 7) property (2.9) holds for every child c of j, that is, $p(D \oplus T_c) - p(D) = \Delta(T_c, D)$. By inductive hypothesis we are bound to exit the while loop, thus the lemma holds in the next call. □

Recall that y is also a feasible solution for the dual relaxation of P-TBC and its cost is given by $\mathrm{DL} = \sum_{i=1}^{n} y_i - (p(U) - P)\lambda$. The following lemma proves the upper bound of Theorem 2.3.

Lemma 2.8. *Suppose* MERGE *outputs D. Then $c(D) \leq \left(1 + \frac{1}{3^{k-1}}\right)\mathrm{DL} + k\, c_{\max}$ for all $k \in Z_+$.*

Proof. Let us digress for a moment for the sake of exposition. Suppose that in Line 6 of MERGE, instead of calling INCREASE, we return $D' = D \oplus T_r$. Notice every arc in the merger graph has exactly one endpoint in D'. By Lemma 2.5, any element i not covered by D' must have $y_i = \lambda\, p_i$.

Furthermore, if $y_i > 0$ then there exists at most one set in D' that covers i; if two such sets exist, one must dominate the other in y and y^-, which is not possible. Hence,

$$c(D) = \sum_{j \in D'} \sum_{i : a_{ij}=1} y_i = \sum_{\substack{i \text{ s.t.} \\ \exists\, j \in D' : a_{ij}=1}} y_i = \sum_{i \in U} y_i - (p(U) - p(D'))\lambda \leq \mathrm{DL} + (p(D') - P)\lambda \tag{2.10}$$

In the fortunate case that $(p(D') - P)\lambda \leq k c_{\max}$, the lemma would follow. Of course, this need not happen and this is why we make the call to INCREASE instead of returning D'.

Let j_q be the root of the q^{th} subtree split by INCREASE/DECREASE. Also let D_q the solution right before splitting T_{j_q}, and D'_q and D''_q be the feasible/infeasible pair of solutions after the splitting, which are used as parameters in the recursive calls (Lines 11-12). Suppose Lines 7-9 processed only one child of j_q, this can only happen in INCREASE, in which case $p(D''_q) > P$ but $p(D''_q) - b_{j_q} < P$. The same argument used to derive (2.10) gives us

$$c\left(D''_q \setminus \{j_{\leq q}\}\right) \leq \sum_{i \in U} y_i - \left(p(U) - p(D''_q) + b_{j_q}\right)\lambda \leq \mathrm{DL} \tag{2.11}$$

The cost of the missing sets is $c(\{j_{\leq q}\}) \leq q\, c_{\max}$, thus if $q \leq k$ the lemma follows. A similar bound can be derived if the recursive call ends in Line 3 before splitting the k^{k} subtree. Finally, the last case to consider is when Lines 7-9 process two or more children j_q for all $q \leq k$. In this case

$$|p(D_q) - P| \geq 3 \min\left\{|p(D'_q) - P|, |p(D''_q) - P|\right\} = 3\,|p(D_{q+1}) - P|, \tag{2.12}$$

which implies $|p(D_1) - P| \geq 3^{k-1}|p(D_k) - P| \geq 3^{k-1}|p(D''_k) - P|$. Also, $P - p(D_1) \leq \mathrm{DL}$ since all elements i not covered by D_1 must be such that $y_i = \lambda p_i$. Hence, as before

$$c\left(D''_k \setminus \{j_{\leq k}\}\right) \leq \mathrm{DL} + (p(D''_k) - P)\,\lambda \leq \mathrm{DL} + \frac{P - p(D_1)}{3^{k-1}} \leq \left(1 + \frac{1}{3^{k-1}}\right)\mathrm{DL} \tag{2.13}$$

Adding the cost of $\{j_{\leq k}\}$ we get the lemma. □

2.4.4 Integrality gap example

Let T be a rooted tree and $\{(s_i, t_i)\}_{i=1}^n$ be a collection of pairs of nodes of T, each defining a unique path in T. Let A be the incidence matrix of paths to edges of T. The covering problem defined by A is the well-know Multicut problem where the objective is to find a minimum cost set

of edges whose removal separates all pairs. If for every pair, s_i is the ancestor of t_i or vice-versa, then A is totally balanced. We now show a family of instances of P-TBC with an integrality gap of IP $> \left(1 + \frac{1}{3^{k-1}}\right)$ LP $+ \frac{k}{2} c_{\max}$. This finishes the proof of Theorem 2.3.

Our tree T is made up of a complete binary tree with height $2q$ plus a 2-path coming out of the apex of the binary tree going up into the real root of T; thus the tree has $2^{2q+1} + 1$ nodes. See Figure 6.3 for a picture of the instance. The cost of every edge is 3. There are two kinds of paths: *internal* and *fringe* paths. For every node in the binary tree there is an internal path of length two coming out of the node going up; there are $2^{2q+1} - 1$ such paths each having a profit of 4^q. For each leaf and the root there is a fringe path of length 1 incident on it; there are $2^{2q} + 1$ such paths each having a profit of 2. The target coverage is given by $\overline{P} = 2\left(4^{q-1} + \ldots + 4^0 + 1\right) = \frac{2^{2q+1}+4}{3}$, where the shorthand notation $\overline{X}$ means $p(U) - X$.

Consider the dual solution y where every internal path gets a dual value of 1 and every fringe path gets a dual value of 2. The solution is feasible for $\lambda = 1$ and has cost

$$\text{DL} = \sum_{i \in U} y_i - \overline{P}\lambda = \frac{10\,4^q - 1}{3} \tag{2.14}$$

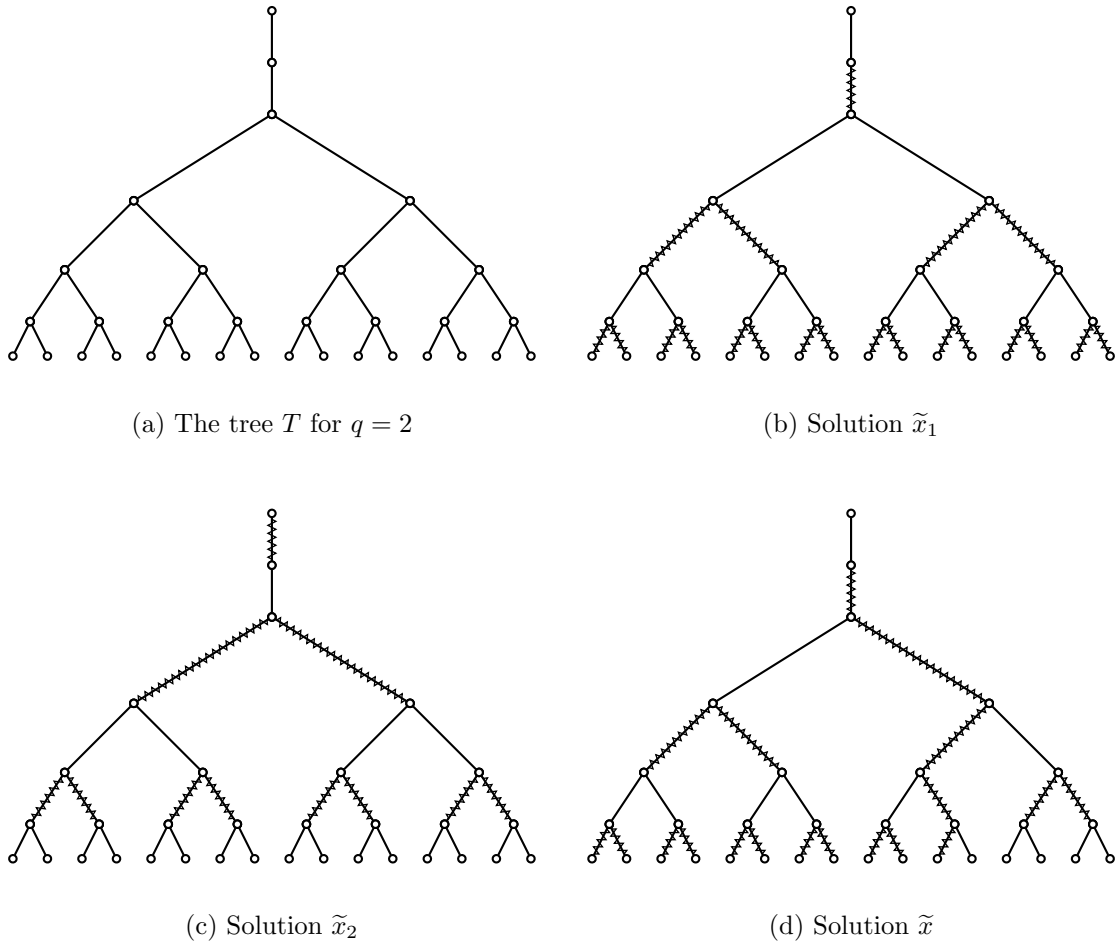
To show that y is optimal, we construct a primal fractional solution x with the same cost, which is a convex combination of two integral solutions $\widetilde{x}_1$ and $\widetilde{x}_2$. Let $\widetilde{x}_1$ consist of edges in every other level of T starting at the leaf level. Let $\widetilde{x}_2$ be the complement of $\widetilde{x}_1$. Note that $\overline{p(\widetilde{x}_1)} = 2$ and $\overline{p(\widetilde{x}_2)} = 2^{2q+1}$; consider the convex combination $\alpha\overline{p(\widetilde{x}_1)} + \beta\overline{p(\widetilde{x}_2)} = \overline{P}$ and let $x = \alpha\widetilde{x}_1 + \beta\widetilde{x}_2$.

$$c(x) = \alpha c(\widetilde{x}_1) + \beta c(\widetilde{x}_2) = \alpha\left(c(\widetilde{x}_1) + \overline{p(\widetilde{x}_1)}\right) + \beta\left(c(\widetilde{x}_2) + \overline{p(\widetilde{x}_2)}\right) - \overline{P} = \text{DL} \tag{2.15}$$

Let $\widetilde{x}$ be the solution defined as follows. For edges incident on a leaf, the $\frac{\overline{P}}{2} - 1$ rightmost ones are not chosen in $\widetilde{x}$, while the remaining are. For other edges, choose the edge only if one of the edges immediately below is not chosen (see Figure 2.6). If we try to pay for $\widetilde{x}$ using the dual cost we will charge twice $2q - 1$ internal paths whose both edges are chosen in $\widetilde{x}$. In other words,

$$c(\widetilde{x}) = \text{DL} + 2q - 1 \tag{2.16}$$

Due to their high profit, internal paths cannot be left uncovered. Using this fact we can infer that $\widetilde{x}$ is indeed an optimal integral solution covering P profit. Choosing $k = q \log_3 4$ we get



(a) The tree T for $q = 2$

(b) Solution $\widetilde{x}_1$

(c) Solution $\widetilde{x}_2$

(d) Solution $\widetilde{x}$

Figure 2.6: Integrality gap example for partial totally balanced cover. a) The instance for $q = 2$. b-d) The wiggly edges belong to the corresponding solution.

the lower bound of Theorem 2.3. That is, for large enough q,

$$c(\widetilde{x}) > \left(1 + \frac{1}{3^{k-1}}\right) c(x) + c_{\max} \frac{k}{2}. \tag{2.17}$$

For smaller values of k the slightly weaker bound with $c_{\max} \frac{k-4}{2}$ additive error holds. It is worth noting that the example can be adapted to yield the same bound for instances with unit profits.

2.5 Applications

In this section we show how Theorem 2.3 implies better approximation algorithms for a number of covering problems that can be expressed with a suitable combination of ρ totally-balanced matrices.

Definition 2.2. *Matrix B is said to be row-induced by a collection of matrices $A_1, \ldots A_k \in R^{n \times m}$ if for all i, the ith row of B equals the ith row of A_j for some $1 \leq j \leq k$.*

Definition 2.3. *Matrix $A \in \{0,1\}^{n \times m}$ is said to be ρ-separable if there exist matrices $A_1, \ldots A_\rho \in \{0,1\}^{x \times m}$ such that $A = \sum_q A_q$ and every matrix row-induced by $A_1, \ldots, A_\rho$ is totally balanced.*

Our algorithms make us of following lemma to absorb the additive error in our bounds.

Lemma 2.9. *Let $\mathcal{A}$ be an algorithm for a given partial covering problem $(U, \mathcal{S}, P)$ that produces a solution with cost at most $\alpha \,\mathrm{OPT} + k\, c_{\max}$, where OPT is the cost of the optimal solution. Then there exists an α-approximation that makes $|U|^{\frac{k}{\alpha-1}}$ calls to $\mathcal{A}$.*

Proof. The idea is to run $\mathcal{A}$ on a modified instance $(U', \mathcal{S}', P')$. Let X be the $\frac{k}{\alpha-1}$ most expensive sets in the optimal cover for $(U, \mathcal{S}, P)$. Let $\mathcal{S}' = \mathcal{S} \setminus \{\, j \,|\, c_j > \min_{j' \in X} c_{j'} \}$, $U' = U \setminus \{\, i \,|\, \text{covered by } X\}$, and $P' = P - p(X)$. The optimal solution in the new instance has cost $\mathrm{OPT}' = \mathrm{OPT} - c(X)$.

Adding X to the solution returned by $\mathcal{A}(U', S', P')$ gives us a feasible solution, for the original instance, with cost at most

$$\alpha \,\mathrm{OPT}' + k\, c'_{\max} + c(X) \leq \alpha \,\mathrm{OPT}' + k\, \frac{\alpha - 1}{k} c(X) + c(X) = \alpha \,\mathrm{OPT}.$$

Unfortunately we do not know which sets comprise X. Therefore $\mathcal{A}$ is run on every choice of X and the best cover found is returned. The number of calls to $\mathcal{A}$ needed is $|U|$ choose $\frac{k}{\alpha-1}$. □

We are ready to describe our approximation algorithms for covering problems that can be described with a ρ-separable matrix. We assume the decomposition is given to us. For an arbitrary matrix finding such a decomposition, or even testing for its existence, may be hard. However, for our application problems it is easy to find the ρ matrices using the problem definition.

Theorem 2.5. *Let A be ρ-separable into matrices $A_1, \ldots, A_\rho$ where $\rho > 1$. For any constant $\epsilon > 0$ there is a $(\rho+\epsilon)$-approximation and a quasi-polynomial time ρ-approximation for the partial covering problem defined by A.*

Proof. Our algorithm is based on the approach of [47, 55]. First, we find an optimal fraction solution (x, r) for the partial covering problem defined by A. Let a_i^q be the ith row of A_q. Notice that for each i there must exist a q_i such that $a_i^{q_i} \cdot x \geq \frac{1-r_i}{\rho}$. Second, we construct a matrix B by choosing $a_i^{q_i}$ as the ith row of B. Note that B is totally balanced. Finally, we find a threshold value λ^* for B and invoke MERGE to find a cover D.

Any feasible solution for B is also feasible for A, thus D is a feasible cover for A. Note that $(\rho x, r)$ is a feasible fractional solution for B. Let OPT be the cost optimal solution for A. By Lemma 2.8 and letting $k \geq \log_3 \frac{\rho}{\epsilon} + 1$ we get,

$$c(D) \leq \left(1 + \frac{1}{3^{k-1}}\right) c(x') + k\, c_{\max} = \left(1 + \frac{1}{3^{k-1}}\right) \rho\, c(x) + k\, c_{\max} \leq (\rho+\epsilon)\mathrm{OPT} + k\, c_{\max} \quad (2.18)$$

This solution is $\rho + \epsilon$ approximate with an additive error of $k\, c_{\max}$ that can be absorbed using Lemma 2.9

For the quasi-polynomial time ρ-approximation, setting $k \geq \log \rho\, |U| + 1$ we get

$$c(D) \leq \left(1 + \frac{1}{\rho\, |U|}\right) c(x') + k\, c_{\max} \leq \rho\, c(x) + (k+1)\, c_{\max} \leq \rho\, \mathrm{OPT} + (k+1)\ c_{\max}. \quad (2.19)$$

And the theorem follows. □

This implies improved approximation algorithm for the partial version of Multicut and Path Hitting on Trees, and Rectangle Stabbing. To show this, we use the following fact about totally

balanced matrices. Let T be a rooted tree. An s-t path in T is said to be *descending* if s is an ancestor of t. Let $\mathcal{P}$ and $\mathcal{Q}$ be collections of descending paths in T, and let A be the $\mathcal{P}$-$\mathcal{Q}$ incidence matrix $A = \{a_{i,j}\}$, where $a_{i,j} = 1$ if and only if the ith path in $\mathcal{P}$ intersects the jth path in $\mathcal{Q}$. It is known that A is totally balanced: To put the matrix into Greedy Standard form arrange the columns and rows of A so that the paths in $\mathcal{P}$ and $\mathcal{Q}$ appear in non-increasing distance from the root.

Corollary 2.1. *There is a $2+\epsilon$ approximation and a quasi-polynomial time 2-approximation for Partial Multicut on Trees.*

Proof. The input of Partial Multicut is a tree T and a collection of paths $\mathcal{P}$ in T, the problem is defined by the $\mathcal{P}$-$E[T]$ incidence matrix A. Even though a path in $\mathcal{P}$ may not be descending, we can always split such a path into two separate descending paths. Therefore, A is 2-separable. □

Corollary 2.2. *There is a $4+\epsilon$ approximation and a quasi-polynomial time 4-approximation for Path Hitting on Trees.*

Proof. The input of Partial Path Hitting is a tree T and two collections of paths $\mathcal{P}$ and $\mathcal{Q}$ in T, the covering problem is defined by the $\mathcal{P}$-$\mathcal{Q}$ incidence matrix A. Parekh and Segev [92] noted that if we split each path in $\mathcal{Q}$ into two descending paths the cost of the optimal solution increases by at most a factor of 2. The matrix of this modified problem is 2-separable. □

Our last application problem is Rectangle Stabbing. This is a special case of Set Cover with ρ-Blocks, a broad class of covering problems introduced by Hassin and Segev [55], where the incidence matrix defining the problem is such that every row has ρ blocks of contiguous 1's.

Theorem 2.6. *Let A be a matrix defining an instance of Set Cover with ρ-Blocks. Then there exists a polynomial time ρ-approximation algorithm for the partial covering problem defined by A.*

Proof. We proceed as in Theorem 2.5 to reduce A to a matrix B. This new matrix is not only totally balanced, but each row consists of a single block of consecutive 1's. For such matrices the merger graph used in Section 2.4.3 is in fact a path. In this case, at most one subtree is split in the

execution of INCREASE. Therefore we get the stronger guarantee that $\mathrm{IP} \leq \mathrm{LP} + c_{\max}$. Plugging in this into the proof of Theorem 2.5 gives the desired result. □

Corollary 2.3. *There is a 2-approximation for Partial Rectangle Stabbing and a d-approximation for Partial d-dimensional Rectangle Stabbing.*

Chapter 3

Making Educated Guesses

In order to narrow the integrality gap of a natural LP formulation, approximation algorithms for partial cover usually need to modify the instance by pruning certain expensive sets. Indeed, many algorithms work as follows: Guess the most expensive set in an optimal solution, prune sets more expensive than the guessed set, run a basic algorithm on each modified instance, and return the best solution found. In this chapter we present a technique for speeding up primal-dual algorithms for partial covering. The high level idea is to run the basic algorithm *once* and make several guesses *along the way*. This allows us to design faster algorithms for two variants of vertex cover.

3.1 Overview

Let $G = (V, E)$ be a graph and let c_v be the cost of vertex $v \in V$. A set of vertices $C \subseteq V$ is said to *cover* an edge $e \in E$ if at least one of the endpoints of e is present in C. Given a target coverage parameter P, the partial vertex cover problem asks for a set of vertices with minimum weight covering at least P edges.

The problem is clearly NP-hard as it generalizes vertex cover. The paper by Bshouty and Burroughs [17] was the first to give a factor 2 approximation for partial vertex cover using LP-rounding. Subsequently, combinatorial algorithms with the same approximation guarantee were proposed. Let the input be a graph on n vertices and m edges. Hochbaum [59] presented an $O(nm \log \frac{n^2}{m} \log n)$ time algorithm based on Lagrangian Relaxation; Bar-Yehuda [9] developed an $O(n^2)$ time algorithm using the local-ratio technique; finally, a primal-dual algorithm which runs in $O(n(n \log n + m))$ time was given by Gandhi et al. [36]. Our approach builds on the work of Gandhi et al. [36].

Motivated by a problem in computational biology, the capacitated vertex cover problem was proposed by Guha et al. [49]. For each vertex $v \in V$ we are given a capacity k_v and a weight c_v. A *capacitated vertex cover* is a function $x : V \to \mathbb{N}_0$ such that there exists an orientation of all edges in which the number of edges directed into $v \in V$ is at most $k_v x_v$. When e is oriented toward v we say e is *covered by* or *assigned to* v. The weight of the cover is $\sum_{v \in V} c_v x_v$. We want to find a cover with minimum weight. In their paper, Guha et al. [49] give a primal-dual 2-approximation for the problem which runs in $O(m + n \log n)$ time. An LP rounding algorithm achieving the same approximation factor was later developed by Gandhi et al. [39]. Given a coverage parameter P, the partial capacitated vertex cover problem ask for a minimum cost cover covering at least P edges.

Our results are fast algorithms for partial vertex cover and partial capacitated vertex cover. Section 3.2 explains some of the challenges in designing approximation algorithms for these problem and outlines the high level idea of our approach. In Section 3.3 we give a primal-dual 2-approximation algorithm for partial vertex cover that runs in $O(n \log n + m)$ time. This represents an improvement in running time over previously known algorithms for the problem. Section 3.4 shows how to adapt the primal-dual algorithm of [49] for capacitated vertex cover to get a 2-approximation for the partial version, which constitutes the first approximation algorithm for partial capacitated vertex cover.

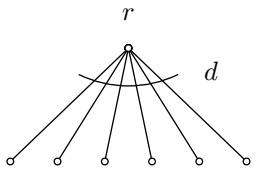
3.2 LP Formulation

Let us describe an integer linear program formulation for the partial vertex cover problem. Each vertex $v \in V$ has associated a variable $x_v \in \{0, 1\}$ which indicates if v is picked for the cover. For every edge $e \in E$ one of its endpoints is picked or the edge is left uncovered. A variable $p_e \in \{0, 1\}$ indicates the latter, i.e., $p_e = 1$ means e is left uncovered. The number of uncovered edges cannot exceed $s = m - P$. The LP relaxation of the IP just described is given below.

$$\min \sum_{v \in V} c_v x_v$$

$$\begin{aligned} p_e + x_u + x_v &\geq 1 && \forall\, e = \{u, v\} \in E \\ \sum_{e \in E} p_e &\leq s \\ x_v, p_e &\geq 0 && \forall\, e \in E, v \in V \end{aligned} \tag{$\mathrm{LP}_{\mathrm{PVC}}$}$$

As it stands, $\mathrm{LP}_{\mathrm{PVC}}$ exhibits an unbounded integrality gap. Consider the graph on the right: A star where the center node r has degree d, all leaves have unit weight while the center node has weight 10. Suppose $P = 2$, that is, we must cover two edges. The optimal integral solution picks two leaves, thus having a total cost of 2. On the other hand, a fractional solution can set $x_r = \frac{2}{d}$ of the center node r and $x_u = 0$ for $u \neq r$, and $p_e = 1 - \frac{2}{d}$ for every edge e. The cost of the fractional solution is $20/d$. Choose d big enough to make the cost as small as desired.



Fortunately, the integrality gap can be narrowed provided we know what is the most expensive vertex in the optimal solution. Let OPT be the optimal solution; we will also use OPT to denote the cost of the optimal solution. Suppose we knew that $h \in V$ is the most expensive vertex in OPT. We modify the LP formulation as follows: our solution must pick h and is not allowed to use vertices more expensive than h, the latter can be achieved by setting their weight to ∞. While this change does not increase the cost of the integral optimal solution, it does narrow the integrality gap. For example, in the above star example setting the weight of the center to ∞ effectively closes the integrality gap. Gandhi et al. [36] showed how to use the above modification to develop a primal-dual 2-approximation algorithm. Since we do not know the most expensive vertex in OPT, we must guess. Their procedure, which takes $O(n \log n + m)$ time, is run on every choice of $h \in V$, and the cheapest cover produced is returned. Therefore, their algorithm runs in $O(n(n \log n + m))$ time.

Our approach is along these lines, but instead of doing exhaustive guessing we run our algorithm just once and make different guesses along the way. One may say that the algorithm

makes educated guesses. This allows us to bring down the running time.

Theorem 3.1. *There is a 2-approximation algorithm for the partial vertex cover problem which runs in $O(n \log n + m)$ time.*

In the next section we describe our algorithm and prove the above theorem.

3.3 Partial vertex cover

The dual program for $\mathrm{LP}_{\mathrm{PVC}}$ is given below. By $E(v)$ we refer to the edges incident to vertex v.

$$\begin{aligned} \max \ & \textstyle\sum_{e \in E} y_e - sz \\ \textstyle\sum_{e \in E(v)} y_e &\leq c_v && \forall\, v \in V \\ y_e &\leq z && \forall\, e \in E \\ y_e, z &\geq 0 && \forall\, e \in E \end{aligned} \tag{$\mathrm{DL}_{\mathrm{PVC}}$}$$

Initially all dual variables are set to 0, and all edges are unassigned. The algorithm works in iterations, each consisting of a *pruning step* followed by a *dual update step.* As the algorithm progresses we build a set $C \subseteq V$, which is initially empty. Along the way we may disallow some vertices; we keep track of these with the set R, which also starts off empty.

In the pruning step we check, for every vertex $v \in V \setminus \{C \cup R\}$, if $C + v$ is a feasible cover. If that is the case, we guess $C + v$ as a candidate cover, and *disallow* vertex v: set $c_v \leftarrow \infty$ and add v to R. Notice that multiple vertices may be disallowed in a single pruning step.

Let $E(R)$ be the edges with both endpoints in R. If $|E(R)| > s$ we stop as there is no hope of finding a cover anymore since $V \setminus R$ is not feasible. At this point we return the candidate cover with minimum weight.

In the dual update step we uniformly raise z and the y_e variables of unassigned edges until some vertex u becomes tight, i.e., its dual constraint is met with equality. Notice that once disallowed, a vertex can never become tight since $c_v = \infty$ for all $v \in R$. Hence, $u \in V \setminus R$. If multiple vertices become tight at the same time then arbitrarily pick one to process. Vertex u

is added to C, unassigned edges in $E(u)$ are assigned to u and their dual variables are frozen. After processing u we proceed to the next iteration. In each dual update step only one vertex is processed, even if multiple tight vertices are available. This ensures that adding u to C does not make C feasible, otherwise u would have been disallowed in the previous pruning step.

The algorithm terminates when $|E(R)| > s$, at which point we know OPT must use at least one vertex in R. Let h be the first vertex in OPT to be disallowed, and C the set at the moment h was pruned. By definition $C + h$ is a feasible solution; suppose its cost, $c(C + h)$, is at most twice OPT. Since the algorithm returns a candidate cover with minimum weight, we achieve an approximation ratio of 2. It all boils down to proving the following lemma.

Lemma 3.1. *Let h be the first vertex in* OPT *to be disallowed. Also, let R and C be the sets constructed by the algorithm right before h was disallowed. Then $c(C + h) \leq 2\,\mathrm{OPT}$.*

Proof. We start off by strengthening the LP formulation. Let $c'_u = c_u$ for $u \notin R$ and $c'_u = +\infty$ for $u \in R$. Since $h \in \mathrm{OPT}$ and $R \cap \mathrm{OPT} = \emptyset$, the following is a valid LP relaxation for our problem.

$$\begin{aligned} \min \ & \textstyle\sum_{v \in V} c'_v x_v \\ p_e + x_u + x_v &\geq 1 && \forall\, e = \{u, v\} \in E \\ \textstyle\sum_{e \in E} p_e &\leq s \\ x_h &\geq 1 \\ x_u, p_e &\geq 0 && \forall\, e \in E, v \in V \end{aligned} \tag{$\mathrm{LP}'_{\mathrm{PVC}}$}$$

The dual of $\mathrm{LP}'_{\mathrm{PVC}}$ is given below.

$$\begin{aligned} \max \ & \textstyle\sum_{e \in E} y_e - sz + \gamma \\ \textstyle\sum_{e \in E(v)} y_e &\leq c'_v && \forall\, v \in V - h \\ \textstyle\sum_{e \in E(h)} y_e + \gamma &\leq c'_h \\ y_e &\leq z && \forall\, e \in E \\ y_e, z, \gamma &\geq 0 && \forall\, e \in E \end{aligned} \tag{$\mathrm{DL}'_{\mathrm{PVC}}$}$$

Let (y, z) be the value of the dual variables just before h was disallowed. Let $\gamma = c'_h - \sum_{e \in E(h)} y_e$. The solution (y, z, γ) is feasible for $\mathrm{DL}'_{\mathrm{PVC}}$. Thus, by weak duality, $\mathrm{OPT} \leq \sum_{e \in E} y_e - sz + \gamma$.

Let r be the vertex added to C just before h was disallowed. (If there is no such vertex, i.e., $C = \emptyset$, then the cover $\{h\}$ is optimal.) Let f be the number of unassigned edges just before r became tight. Among these f edges, suppose t_r were incident on r and t_h were incident on h.

Every vertex $v \in C$ is tight ($c_v = \sum_{e \in Ev} y_e$), thus

$$\begin{aligned} c(C + h) &= \left(\sum_{v \in C} \sum_{e \in E(v)} y_e \right) + \sum_{e \in E(h)} y_e + \gamma, \\ &\leq 2 \left(\sum_{e \in E} y_e - fz \right) + (t_r + t_h)z + \gamma. \end{aligned}$$

The second line follows because edges assigned before r became tight are considered at most twice in the first line. The remaining edges have $y_e = z$ and there are $t_r + t_h$ of them.

Now consider the pruning step right before r became tight. Vertex r was not disallowed then because $f - t_r > s$. Similarly $f - t_h > s$. Thus $t_r + t_h \leq 2\,(f - s)$. It follows that

$$c(C + h) \leq 2 \left(\sum_{e \in E} y_e - sz \right) + \gamma \leq 2\,\mathrm{OPT}.$$

□

For the implementation we keep two priority queues for the vertices. We refer to the value of z as the current time in the execution of the algorithm. The first queue uses the key $k_1(u)$, which denotes the time at which vertex u will become tight; initially $k_1(u) = \frac{c_u}{|E(u)|}$. The second queue uses the key $k_2(u)$, which denotes the number of unassigned edges incident to u; initially $k_2(u) = |E(u)|$.

Let g be the number of unassigned edges; initially $g = m$. In the pruning step we fetch a vertex v with maximum k_2 value. If $g - k_2(v) > s$ proceed to the dual-update step. Otherwise, disallow v by removing v from both queues and adding v to R. Repeat until $|E(R)| > s$, or $g - k_2(v) > s$.

In the dual update step we fetch a vertex v with minimum k_1 value. Set $g \leftarrow g - k_2(v)$, add

v to C and remove v from both queues. For every neighbor $u \notin C \cup R$, if $k_1(u) = 1$ remove u from both queues. Otherwise increase $k_1(u)$ to $z + (k_1(u) - z)\frac{k_2(u)}{k_2(u)-1}$ and decrease $k_2(u)$ to $k_2(u) - 1$.

In order to deal with the guessing we store C as a sorted list, adding vertices at the end. Also we keep two variables h and b. When a vertex v is disallowed we check if the candidate cover $C + v$ is cheaper than the current best candidate cover. If that is the case, set $h \leftarrow v$ and $b \leftarrow |C|$. At the end of the algorithm the cover returned is made up of h and the first b elements of C.

The running time is dominated by the priority queue operations. At most $2n$ remove and $2m$ modify key operations are performed, which using Fibonacci heaps takes $O(n \log n + m)$ time. This finishes the proof of Theorem 3.1.

3.4 Partial capacitated vertex cover

Our technique can be used to solve a more general version of the vertex cover problem. In the partial capacitated vertex cover problem every vertex $u \in V$ has a capacity k_u and a weight c_u. A single copy of u can cover at most k_u edges incident to it, the number of copies picked is given by $x_u \in \mathbb{N}_0$. Every edge $e = \{u, v\} \in E$ is either left uncovered ($p_e = 1$) or is assigned to one of its endpoints, variables y_{eu} and y_{ev} indicate the latter. The number of edges assigned to a vertex u cannot exceed $k_u x_u$, while the ones left unassigned cannot be more than $s = m - P$. The LP relaxation of the program just described is given below. The additional constraint $x_v \geq y_{ev}$ is needed to narrow the integrality gap (c.f. [49]).

$$\min \textstyle\sum_{v \in V} c_v x_v$$

$$\begin{aligned}
p_e + y_{eu} + y_{ev} &\geq 1 && \forall\, e = \{u, v\} \in E \\
\textstyle\sum_{e \in E(v)} y_{ev} &\leq k_v x_v && \forall\, v \in V \\
x_v &\geq y_{ev} && \forall\, v \in e \in E \\
\textstyle\sum_{e \in E} p_e &\leq s \\
p_e, y_{ev}, x_v &\geq 0 && \forall\, v \in e \in E
\end{aligned} \tag{$\mathrm{LP}_{\mathrm{PCVC}}$}$$

For the sake of completeness we describe how to, given an integral feasible cover x, set the y and p variables. This an assignment problem that can be solved using max flow. Consider a network with two layers, a source, and a sink. In the first layer there is a node for every $e \in E$; in the second layer there is a node for every $v \in V$ plus a dummy node that represents the option of leaving an edge uncovered. Every e is connected with unit-capacity edges to the source, its two endpoints and the dummy node. Every node v is connected with an edge with capacity $k_v x_v$ to the sink. Finally, the dummy node is connected to the sink using an edge with capacity s. An integral flow shipping m units of flow from the source to the sink corresponds to a valid edge assignment for x and vice versa.

Our algorithm, however, does not use this approach, rather, $m - s$ edges are assigned to one of their endpoints; then as many copies of each vertex as necessary are chosen to cover the edges assigned to it. The algorithm works in iterations, each having a pruning step followed by a dual update step. For the latter we follow the algorithm in [49] which is described here for completeness. The dual program of $\mathrm{LP}_{\mathrm{PCVC}}$ is given below.

$$\max \textstyle\sum_{e \in E} \alpha_e - sz$$

$$\begin{aligned} k_v q_v + \textstyle\sum_{e \in E(v)} l_{ev} &\leq c_v && \forall\, v \in V \\ \alpha_e &\leq q_v + l_{ev} && \forall\, v \in e \in E \\ \alpha_e &\leq z && \forall\, e \in E \\ q_v, l_{ev}, \alpha_e, z &\geq 0 && \forall\, v \in e \in E \end{aligned} \tag{$\mathrm{DL}_{\mathrm{PCVC}}$}$$

Initially all variables are set to 0 and all edges are unassigned. At the beginning, vertex u is said to be high degree if more than k_u unassigned edges are incident to u, otherwise u is low degree. As we will see a vertex u may start off as high degree and later become low degree as adjacent edges get assigned to its neighbors. For such u we define L_u to be the set of unassigned edges incident to u just after u becomes low degree. If u is low degree from the beginning, i.e., $\deg(u) \leq k_u$, we define $L_u = E(u)$.

In the pruning step when processing a vertex $v \in V \setminus \{C \cup R\}$, although in principle we are

allowed to pick multiple copies of it, we check if adding just one copy of v to the current solution makes it feasible. If that is the case, guess the current solution plus v and then disallow v: add it to R and set $c_v \leftarrow \infty$.

In the dual update step we uniformly raise z and α_e for all unassigned edges. Because of the constraint $\alpha_e \leq q_v + l_{ev}$ one of the terms on the right hand side must be raised by the same amount. Which variable is increased depends on the nature of v: If v is high degree we raise q_v, otherwise we raise l_{ev}. When some vertex u becomes tight we add it to C and *open* it: If u is high degree we assign to it all the unassigned edges in $E(u)$, otherwise u is low degree and all edges in L_u are assigned to it. Notice that in the later case some of the edges in L_u might have already been assigned. This means that some of the edges originally assigned to a high degree vertex when it was opened may later be taken away by the opening of low degree neighbors. Therefore the number of copies of a vertex that is needed can decrease over time after it is opened. In what follows, when we talk about the current solution we mean the current assignment of edges.

Suppose that in the dual update step the algorithm opens vertex u. If u is low degree, its opening cannot make the solution feasible, otherwise it would have been disallowed in the previous pruning step. On the other hand, if u is high degree, opening u may make the solution feasible, since we can use multiple copies of u. In the latter case we say the algorithm *ends prematurely*: we assign to u just enough edges to make the solution feasible (i.e., exactly s edges are left unassigned) and we return the best solution among the candidate covers and the current solution. If on the other hand this does not happen and R grows to the point where $|E(R)| > s$ we stop and return the best candidate cover.

Let us first consider the case where the algorithm ends prematurely, we will show that we can construct a solution with cost at most $2\left(\sum_{e\in E}\alpha_e - sz\right)$. Let φ be the edge assignment when the algorithm ended prematurely, where $\varphi(v)$ denotes the set of edges assigned to vertex v. Constructing the solution is simple: we pick enough copies of every vertex to cover the edges assigned to it, that is, for every $u \in V$ we chose $\left\lceil \frac{|\varphi(u)|}{k_u} \right\rceil$ copies of u.

Let u be one of the vertices opened. Furthermore suppose u was low degree when it was

opened, this means only one copy is needed. If u started off as low degree then $c_u = \sum_{e \in L_u} l_{eu} = \sum_{e \in L_u} \alpha_e$ as $q_u = 0$. On the other hand if u became low degree afterward then $|L_u| = k_u$ and only edges in L_u have nonzero l_{eu}. Thus,

$$c_u = q_u k_u + \sum_{e \in L_u} l_{eu} = \sum_{e \in L_u} (q_u + l_{eu}) = \sum_{e \in L_u} \alpha_e.$$

In both cases we can pay for u by charging once the edges in L_u.

Now let us consider the case when u is high degree. By definition more than k_u edges were assigned to u when it became tight. Note that some of these edges may later be taken away by low degree neighbors. There are two cases to consider. If $|\varphi(u)| < k_u$ we only need one copy of u, charging once the edges originally assigned to u (which are more than k_u and have $\alpha_e = q_u$) is enough to pay for $c_u = k_u q_u$. Otherwise $\left\lceil \frac{|\varphi(u)|}{k_u} \right\rceil$ copies of u are needed. Unfortunately our budget of $\sum_{e \in \varphi(u)} \alpha_e = |\varphi(u)| q_u = \frac{|\varphi(u)|}{k_u} c_u$ is only enough to pay for the first $\left\lfloor \frac{|\varphi(u)|}{k_u} \right\rfloor$ copies. The key observation is that edges in $\varphi(u)$ will not be charged from the other side, therefore charging any k_u edges in $\varphi(u)$ one more time is enough to pay for the extra copy needed.

How many times can a single edge be charged? At most twice, either from a single (high degree) endpoint or once from each endpoint. We are leaving s edges uncharged with $\alpha_e = z$, therefore the solution can be paid with $2(\sum_{e \in E} \alpha_e - sz)$.

Before ending prematurely the algorithm may have disallowed some vertices. If none of them is used by OPT, strengthening the LP by setting their weight to ∞ does not increase the cost of the integral optimal solution. Therefore the cost of the dual solution is a lower bound on OPT. It follows from the above analysis that the cover found has cost at most $2\,\mathrm{OPT}$. Otherwise OPT uses at least one vertex in R. Notice that if the algorithm terminates because $|E(R)| > s$ we also get that $R \cap \mathrm{OPT} \neq \emptyset$. In either case, let $h \in R \cap \mathrm{OPT}$ be such that vertices disallowed by the algorithm before h do not belong to OPT.

Let R and C be the sets constructed by the algorithm just before h was disallowed. We proceed as before by strengthening the LP relaxation. Let $c'_u = c_u$ for $u \notin R$ and $c'_u = +\infty$ for

$u \in R$. Since $h \in \text{OPT}$ and $R \cap \text{OPT} = \emptyset$, the following is a valid LP relaxation for our problem.

$$\min \textstyle\sum_{v \in V} c'_v x_v$$

$$\begin{aligned}
p_e + y_{eu} + y_{ev} &\geq 1 && \forall\, e = \{u, v\} \in E \\
\textstyle\sum_{e \in E(v)} y_{ev} &\leq k_v x_v && \forall\, v \in V \\
x_v &\geq y_{ev} && \forall\, v \in e \in E \\
\textstyle\sum_{e \in E} p_e &\leq s \\
x_h &\geq 1 \\
p_e, y_{ev}, x_v &\geq 0 && \forall\, v \in e \in E
\end{aligned} \tag{LP'_{PCVC}}$$

The dual of LP'_{PCVC} is given below.

$$\max \textstyle\sum_{e \in E} \alpha_e - sz + \gamma$$

$$\begin{aligned}
k_v q_v + \textstyle\sum_{e \in E(v)} l_{ev} &\leq c'_v && \forall\, v \in V - h \\
k_h q_h + \textstyle\sum_{e \in E(h)} l_{eh} + \gamma &\leq c'_h \\
\alpha_e &\leq q_v + l_{ev} && \forall\, v \in e \in E \\
\alpha_e &\leq z && \forall\, e \in E \\
q_v, l_{ev}, \alpha_e, z &\geq 0 && \forall\, v \in e \in E
\end{aligned} \tag{DL'_{PCVC}}$$

Let (α, q, l, z) be the dual solution constructed by the algorithm just before h was disallowed. Let $\gamma = c'_h - \left(k_h q_h + \sum_{e \in E(h)} l_{eh}\right)$. The solution $(\alpha, q, l, z, \gamma)$ is a feasible solution for DL'_{PCVC}. Hence, its cost offers a lower bound on OPT.

Now consider the edge assignment when h was pruned. Before constructing the cover we need to modify the assignment. If h is low degree, we assign to h the edges in L_h, otherwise we assign to it any k_h unassigned edges in $E(h)$. This may cause strictly less than s edges to be left unassigned. To fix this we take away from r, the last vertex to become tight, enough edges as to leave exactly s edges unassigned. (If there is no such vertex, i.e., $C = \emptyset$, then the cover $\{h\}$ is optimal.) This can always be done because h was not disallowed before r was opened. Let φ be the edge assignment after this modification. Now we build the cover by picking as many copies of

each vertex as necessary to cover the edges assigned to it.

Suppose that just before r became tight there were f unassigned edges. Among these f edges, suppose t_r were incident on r and t_h were incident on h. There are two cases to consider depending on the number of copies of r needed.

Suppose a single copy of r is needed. This can happen because r was low degree, or because r was high degree but later some edges were taken away from r in order to leave exactly s edges uncovered. In either case, using the charging scheme described above, we get:

$$\begin{aligned}\sum_{u \in V} \left\lceil \frac{|\varphi(u)|}{k_u} \right\rceil c_u &\leq 2 \left(\sum_{e \in E} y_e - fz \right) + \Big(t_r + \min\{t_h, k_h\} \Big) z + \gamma, \\ &\leq 2 \left(\sum_{e \in E} y_e - sz \right) + \gamma.\end{aligned}$$

The second inequality follows because $f - t_r > s$ and $f - \min\{t_h, k_h\} > s$, which in turn hold because the algorithm did not end prematurely when r was opened and h was not disallowed in the pruning step before r became tight.

Now suppose r was high degree and multiple copies of r were chosen. Then

$$\begin{aligned}\sum_{u \in V} \left\lceil \frac{|\varphi(u)|}{k_u} \right\rceil c_u &\leq 2 \left(\sum_{e \in E} y_e - fz \right) + \Big(2|\varphi(r)| + \min\{t_h, k_h\} \Big) z + \gamma, \\ &\leq 2 \left(\sum_{e \in E} y_e - sz \right) + \gamma.\end{aligned}$$

The second inequality follows because after the reassignment $f - s = |\varphi(r)| + \min\{t_h, k_h\}$.

In both cases we can pay for the cover with twice the cost of the dual solution, thus, there is a candidate cover that is 2 approximate. Since we chose a candidate cover with minimum weight the algorithm is a 2-approximation.

The implementation is similar to that of the algorithm for partial vertex cover described in Section 3.3. We keep two priority queues for the vertices. The first queue is used during the dual update to fetch the next vertex to become tight. The second priority queue is used during the pruning step to fetch a vertex with most unassigned edges incident on.

When a certain vertex v is disallowed we check if the cost of the candidate cover induced by v and the current edge assignment is cheaper than the current best candidate cover, if that is the

case we remember v. In order to do this efficiently we need to keep track of the cost of the current edge assignment. Note that when a vertex u is opened the cost increases because of the additional copies of u that are needed. However, if u is low degree it is possible that the number of copies of a high degree neighbor in L_u may decrease. This is easy to check by scanning the edges in L_u.

Suppose h was the vertex that induced the best candidate cover when it was disallowed. To recover the actual solution the algorithm is run a second time. When the algorithm tries to disallow h we stop and return the cover built at this point.

Theorem 3.2. *There is a 2-approximation algorithm for the partial capacitated vertex cover problem which runs in $O(n \log n + m)$ time.*

3.5 Generalizations

The pruning/dual update technique is quite general and may be applied to other covering problems where a primal-dual already exist for the full version of the problem. For example, the primal-dual algorithm of Jain and Vazirani [65] for the facility location problem can be used in conjunction with our method to design a faster approximation algorithm for the partial version of the problem, which was studied by Charikar et al. [20]. Very recently, the technique has been used by Bar-Yehuda et al. [14] to develop approximation algorithms for a more general version of partial capacitated vertex cover where the edges have arbitrary profits.

Chapter 4

Beyond Primal Complementary Slackness

Primal-dual algorithms are typically based on linear programming formulations in which the primal program uses 0-1 variables. After the dual update is carried out, a primal solution is built by setting to 1 a subset of those variables whose dual constraint is tight. In this chapter we explore the use of more sophisticated procedures for building a primal solution. As a result, we obtain the first purely combinatorial algorithm for the data migration problem where the objective is to minimize the average completion time over all storage disks.

4.1 Overview

Let $G = (V, E)$ be a graph. (All our graphs are multigraphs, for simplicity, however, we drop the multi prefix when talking about graphs and sets of edges.) Let $E(u)$ denote the set of edges incident on a vertex u. The vertices and edges in G are jobs to be completed. Each vertex v has weight w_v and each edge has processing time p_e. The completion time, C_e, of edge e is simply the time at which its processing is completed. The completion time, C_v, of vertex v is the latest completion time of any edge in $E(v)$. The crucial constraint is that two edges incident on the same vertex cannot be processed at the same time. The objective is to minimize $\sum_{v \in V} w_v C_v$.

Kim [74] gave approximation algorithms for this problem: A 3-approximation when edges have unit processing times and a 9-approximation for the general case. Kim's algorithms round the solution produced by a linear programming relaxation for the problem. This algorithm involves solving a linear program with an exponential number of constraints, though there are equivalent linear programs with a polynomial number of constraints (cf. [35]). Gandhi et al. [37] show that Kim's algorithm can not give an approximation guarantee better than 3. The best approximation guarantee for the general case is 5.03 [37] which is obtained by combining solutions to two algo-

rithms each of which yields an approximation guarantee of 5.83. At a very high level all known algorithms for this problem comprise of the following two steps: (i) label the edges, (ii) consider the edges for scheduling in sorted order of their labels. The algorithms in [37, 74] label the edges based on the fractional completion times in the linear programming solution.

In this chapter we present simple and efficient primal-dual algorithms, which constitute the first purely combinatorial algorithms for the problem. A novel aspect of our approach is that, unlike typical primal-dual algorithms, the primal solution is not constructed using relaxed complementary slackness. Rather, in step (i), the dual update assigns labels to the edges. These labels are used, in step (ii), to guide a scheduling procedure, which is almost identical to that in [74] for unit edge processing times and that in [37] for arbitrary edge processing times. In the analysis, the cost of the schedule is related to the cost of the dual solution via the labels, yielding a 3-approximation for unit-length processing times, and a 5.83-approximation for arbitrary processing times.

4.2 A Linear Programming Relaxation

The following linear programming relaxation for the data migration problem was given by Kim [74]. Such relaxations have been proposed earlier by Wolsey [101] and Queyranne [93] for single machine scheduling problems and by Schulz [98] and Hall et al. [51] for parallel machines and flow shop problems. For a vertex v, let C_v represent the completion time of v. Let $N(u)$ represent the set of neighbors of vertex u and $E(u)$ the set of edges incident on u. For any edge $e = (u, v)$, we use p_e to denote the processing time of e. For any set $F \subseteq E$, let $p(F) = \sum_{e \in F} p_e$ and $p(F)^2 = \sum_{e \in F} p_e^2$.

$$\min \sum_{v \in V} w_v \, C_v$$

subject to

$$\sum_{e=(u,v) \in S(u)} p_e \, C_v \geq \frac{p(S(u))^2 + p(S(u)^2)}{2} \qquad \forall u \in V, S(u) \subseteq E(u) \qquad (4.1)$$

$$C_v \geq p(E(v)) \qquad \forall v \in V \qquad (4.2)$$

The dual LP contains a variable $y_{S(u)}$ for each set $S(u)$, corresponding to constraint (4.1), and a variable z_v for each $v \in V$, corresponding to constraint (2) . The dual LP is given below.

$$\max \sum_{\substack{u \in V \\ S(u) \subseteq E(u)}} \frac{p(S(u))^2 + p(S(u)^2)}{2} y_{S(u)} + \sum_{v \in V} p(E(v)) z_v$$

subject to

$$z_v + \sum_{\substack{e=(u,v) \\ S(u): e \in S(u)}} y_{S(u)}\, p_e \leq w_v \qquad \forall v \in V \qquad (4.3)$$

$$y_{S(u)} \geq 0 \qquad \forall u \in V, S(u) \subseteq E(u) \qquad (4.4)$$

4.3 Algorithm

The algorithm consists of two phases—*labeling* and *scheduling*. In the labeling phase, the vertices receive labels which are then used to label the edges. The scheduling phase uses the edge labels to decide the order in which the edges must be considered for processing.

4.3.1 Labeling Phase

The high level idea of the labeling algorithm is as follows. Each vertex is initially unlabeled. The algorithm proceeds in iterations. In each iteration we find a vertex x which maximizes the total length of edges going from x to unlabeled neighbors of x, call this set of edges $S(x)$. If there exists a high degree node h such that $p(E(h)) > p(S(x))$, then assign h a label of $p(S(x))$ and raise z_h until constraint (4.3) is tight, namely,

$$z_h \;=\; w_h \;-\; \sum_{\substack{e=(u,h) \\ S(u): e \in S(u)}} y_{S(u)}\, p_e.$$

Otherwise, the value of the dual variable $y_{S(x)}$ is increased until the dual constraint (4.3) is met with equality for some unlabeled neighbor of v. In other words, $y_{S(x)}$ assumes the smallest value such that for some vertex $v \in N(x)$ such that $(x, v) \in S(x)$ we have

$$\sum_{\substack{e=(u,v) \\ S(u): e \in S(u)}} y_{S(u)}\, p_e \;=\; w_v.$$

LABEL(V, E)

1 **for** $v \in V$ **do**

2 $\quad \ell_v \leftarrow$ **nil** // v is unlabeled

3 **while** there exists an unlabeled vertex **do**

4 $\quad x \leftarrow$ vertex maximizing $p(S(x))$, where $S(x) = \{(x, v) \in E(x) \mid \ell_v = \textbf{nil}\}$

5 $\quad h \leftarrow$ unlabeled vertex maximizing $p(E(h))$

6 $\quad$ **if** $p(E(h)) > p(S(x))$

7 $\quad\quad z_h \leftarrow w_h$

8 $\quad\quad \ell_h \leftarrow p(S(x))$ // h is now labeled

9 $\quad$ **else**

10 $\quad\quad y_{S(x)} \leftarrow \min\left\{ \frac{w_v}{\sum_{e=(x,v)} p_e} \mid v \in N(u) \text{ s.t. } \ell_v = \textbf{nil} \right\}$

11 $\quad\quad$ **for** $v \in N(x)$ s.t. $\ell_v =$ **nil do**

12 $\quad\quad\quad w_v \leftarrow w_v - y_{S(x)} \sum_{e=(x,v)} p_e$

13 $\quad\quad\quad$ **if** $w_v = 0$

14 $\quad\quad\quad\quad \ell_v \leftarrow p(S(x))$ // v is now labeled

15 **return** ℓ

Figure 4.1: Labeling procedure.

All unlabeled neighbors of x for which the above equality holds are labeled $p(S(x))$.

Let us denote the label of vertex u by ℓ_u. These labels induce an ordering on the vertices. This ordering can be naturally extended to the edges: For any two edges $e = (u, v)$ and $f = (u', v')$, we say $\ell_e \leq \ell_f$ if

(i) $\min\{\ell_u, \ell_v\} < \min\{\ell_{u'}, \ell_{v'}\}$, or

(ii) $\min\{\ell_u, \ell_v\} = \min\{\ell_{u'}, \ell_{v'}\}$ and $\max\{\ell_u, \ell_v\} \leq \max\{\ell_{u'}, \ell_{v'}\}$.

The pseudo-code for the labeling phase is given in Figure 4.1. We now state and prove an important properties of the labels and the dual solution generated by our algorithm.

Lemma 4.1. *Let $e = (v, w) \in E$ and $F_e(w) = \{f \in E(w) \,|\, \ell_f \leq \ell_e\}$. Then, $p(F_e(w)) \leq \ell_v$.*

Proof. Let $N_e(w) = \{y \in N(w) \,|\, (w, y) \in F_e(w)\}$. Note that for each vertex $y \in N_e(w)$ we have $\ell_y \leq \ell_v$. Consider the iteration of the algorithm in which the first vertex in $N_e(w)$ is labeled, and let y be that vertex. At the beginning of this iteration the sum of the processing times of edges in $E(w)$ whose other endpoints are unlabeled is at least $p(F_e(w))$. Notice that x is chosen (Line 4 in the pseudo-code LABEL) to be the vertex with the maximum value of $p(S(x))$, where $S(x)$ is set of edges incident on x whose other endpoints are unlabeled. Therefore, it must be the case that $p(F_e(w)) \leq p(S(x)) = \ell_y$, which by definition is at most ℓ_v. □

Lemma 4.2. *For any $h \in V$. If $z_h > 0$ then $p(E(h)) > \ell_h$.*

Proof. In the pseudo-code, variable z_h is set in Line 7. In order to reach Line 7, the condition of if statement in Line 6 must be satisfied. Thus, $p(E(h)) > \ell_h$. □

Lemma 4.3. *Let $x, v \in V$ and $S(x) \subseteq E(x)$ such that $(x, v) \in S(x)$. If $y_{S(x)} > 0$ then $\max\{\ell_v, p(E(v))\} \leq p(S(x))$.*

Proof. Consider the iteration in which $y_{S(x)}$ was set. In the pseudo-code, variable $y_{S(x)}$ is set in Line 10. In order to reach Line 10, it must be that in this iteration all unlabeled vertices have degree at most $p(S(x))$. In particular, since v was unlabeled at the beginning of the iteration, $p(E(v)) \leq p(S(x))$. If vertex v is labeled in this iteration then $\ell_v = p(S(x))$. Otherwise v is

```
SCHEDULE(V, E, ℓ)
1  sort edges in non-decreasing ℓ order
2  for e = (u, v) ∈ E (in sorted order) do
3      schedule e as early as possible without creating
       conflicts with edges scheduled so far.
```

Figure 4.2: Scheduling with unit-length processing times

labeled in a later iteration. Since the value of the labels assigned by the algorithm only decreases with time, we have $\ell_v \leq p(S(x))$. □

4.3.2 Scheduling edges with unit processing times

The scheduling phase is almost the same as the list scheduling algorithm [48] used by Kim [74]. The only difference is in the entity that is used to decide the order in which the edges are processed. We use the edge labels generated by algorithm LABEL to decide the ordering whereas Kim [74] uses the completion times of edges as returned by the linear programming solution.

The edges are sorted in increasing order of their labels. The edges are then processed in this order. When processing $(u, v) \in E$, the edge is scheduled at the earliest time such that no edge incident upon u or v is already scheduled at that time. The pseudo-code is given in Figure 4.2.

Let us analyze the algorithm. Let $\widetilde{C}_v$ be the completion time of vertex v in our algorithm. Recall that $E(v)$ denotes the set of edges incident on v, and $N(v)$ the set of neighbors of v.

Lemma 4.4. *For each $v \in V$, $\widetilde{C}_v \leq \ell_v + |E(v)| - 1$.*

Proof. Let $e_v = (w, v)$ be the last edge to finish among the edges in $E(v)$. Recall from Lemma 4.1 that $F_{e_v}(w) = \{f \in E(w) \,|\, \ell_f \leq \ell_{e_v}\}$. Observe that because of the order in which the edges are scheduled, we have

$$\widetilde{C}_v \leq |F_{e_v}(w)| + |E(v)| - 1.$$

From Lemma 4.1 we know that $p(F_{e_v}(w)) = |F_{e_v}(w)| \leq \ell_v$. Substituting this in the above expression completes the proof. □

Theorem 4.1. *The data migration problem with edges having unit processing times has a 3-approximate primal-dual algorithm.*

Proof. Let $G = (V, E)$ be an instance of the data migration problem. The cost of the schedule found by our algorithm is given by

$$\begin{aligned}
\sum_{v \in V} w_v \widetilde{C}_v &\leq \sum_{v \in V} w_v \left(\ell_v + |E(v)|\right) && \text{[using Lemma 4.4]} \\
&= \sum_{v \in V} w_v \, \ell_v + \sum_{v \in V} w_v \, |E(v)|
\end{aligned}$$

Clearly $\sum_{v \in V} w_v |E(v)| \leq OPT(G)$. Thus, if we can show that $\sum_{v \in V} w_v \ell_v \leq 2\, OPT(G)$ we are done. To that end, we relate $\sum_{v \in V} w_v \ell_v$ to the cost of dual feasible solution obtained by the algorithm, which we denote by $DFS(G)$.

$$\begin{aligned}
\sum_{v \in V} w_v \, \ell_v &= \sum_{v \in V} \left(z_v + \sum_{\substack{e=(u,v) \\ S(u): e \in S(u)}} y_{S(u)} \right) \ell_v && \text{[every vertex is tight]} \\
&= \sum_{v \in V} z_v \, \ell_v \; + \sum_{v \in V} \sum_{\substack{e=(u,v) \\ S(u): e \in S(u)}} y_{S(u)} \, \ell_v \\
&\leq \sum_{v \in V} z_v \, |E(v)| \; + \sum_{v \in V} \sum_{\substack{e=(u,v) \\ S(u): e \in S(u)}} y_{S(u)} \, \ell_v && \text{[using Lemma 4.2]} \\
&\leq \sum_{v \in V} z_v \, |E(v)| \; + \sum_{v \in V} \sum_{\substack{e=(u,v) \\ S(u): e \in S(u)}} y_{S(u)} \, |S(u)| && \text{[using Lemma 4.3]} \\
&= \sum_{v \in V} z_v \, |E(v)| \; + \sum_{\substack{u \in V \\ S(u) \subseteq E(u)}} \sum_{e \in S(u)} y_{S(u)} \, |S(u)| \\
&= \sum_{v \in V} z_v \, |E(v)| \; + \sum_{\substack{u \in V \\ S(u) \subseteq E(u)}} y_{S(u)} \, |S(u)|^2 \\
&\leq 2\, DFS(G)
\end{aligned}$$

Since $DFS(G)$ is a lower bound on $OPT(G)$, it follows that $\sum_{v \in V} w_v \ell_v \leq 2\, OPT(G)$.

□

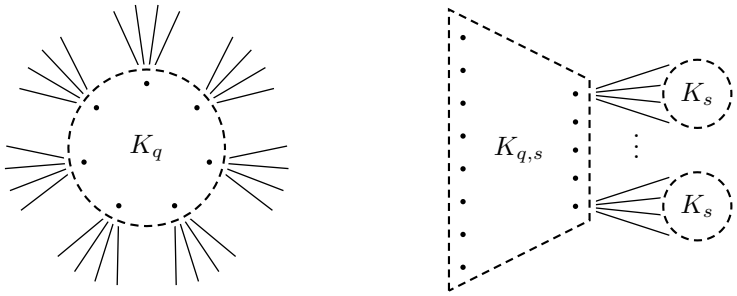


Figure 4.3: Instances showing the relevance of the labeling step.

4.3.2.1 Order matters

In this section we explore the importance of the labeling step. Consider the following natural and intuitive variant of SCHEDULE: Instead of sorting the edges based on the label we process the edges in any arbitrary order, scheduling them as early as possible without creating conflicts with the edges scheduled so far. While this algorithm gives a solution that is at most twice the cost of optimal for $\min \sum_e C_e$ [6] and $\min \max_e C_e$, the following example shows that for the objective of $\min \sum_v C_v$ it may produce a solution with cost $\Omega(\sqrt[3]{n})$ times the optimum.

Consider a complete graphs on q vertices. To this, attach q copies of $K_{1,\sqrt{q}}$, so that each node in K_q is the center of one of the stars, as shown in Figure 4.3 on the left. The optimal solution first schedules the stars in parallel and then the edges in K_q, with a total cost of $\Theta(q^2)$. On the other hand, a minimal solution may schedule the edges in K_q before the stars, incurring an overall cost of $\Theta(q^{2.5})$. Since the graph has $\Theta(q^{1.5})$ vertices, this shows that a minimal schedule can be a factor $\Omega(\sqrt[3]{n})$ away from the optimum.

Looking at the previous example we see that we should have scheduled the edges incident to the nodes with degree 1 *before* working on edges whose endpoints have higher degree. This suggests a heuristic based on degrees instead of labels: First sort the edges $(u, v) \in E$ in non-decreasing value of $\min(d_u, d_v)$, breaking ties with $\max(d_u, d_v)$; then schedule the edges (in sorted order) as early as possible without creating a conflict with the partial schedule built so far.

This new heuristic clearly solves the previous example optimally. Unfortunately, it is not a constant approximation either, as it may produce a solution with cost $\Omega(\sqrt[4]{n})$ times optimum.

Our bad instance, shown in Figure 4.3 on the right, has three layers. The first, second and third layers contain q, s and s^2 nodes respectively. The first and second layer are connected with a complete bipartite graph $K_{q,s}$. The nodes in the third layer are divided into s groups, each forming a K_s that is connected to a single node in the second layer. The parameters q and s will be chosen to get the desired gap.

Notice that nodes in the first and third layers have degree s, thus, the heuristic first schedules the edges in the K_s's, and then we are free to schedule the remaining edges in *any* order as their endpoints have degree s and $q+s$. Suppose a solution S_1 first schedules the edges from the first to the second layer, while S_2 first schedules the edges from the second to the third layer. In S_1 the second-layer nodes are busy for the first q time steps working on the $K_{q,s}$ edges, as a result, every node in the third layer finishes by $\Omega(q)$, and the overall cost is $\Omega(s^2q)$. On the other hand, in S_2 the third-layer nodes finish by $O(s)$ and the first and second-layer nodes finish by $O(q+s)$; thus, the overall cost is $O((q+s)^2+s^3) = O(q^2+s^3)$. Choosing $q = s^{\frac{3}{2}}$, the ratio of the cost of S_1 and S_2 is $\Omega(\sqrt{s})$. The graph has $O(s^2)$ nodes, therefore the greedy schedule can be an $\Omega(\sqrt[4]{n}\,)$ factor away from optimum.

4.3.3 Scheduling edges with arbitrary processing times

The scheduling phase is essentially the same as in [37]. The only difference is in the order in which the edges are scheduled – we decide the order using the labels on the edges, whereas in [37], the ordering is based on the completion times of the edges in the optimal linear programming solution. For completeness, we restate the scheduling algorithm here as described in [37]. In the scheduling phase, each edge $e = (u, v)$ waits for W_e time units before it can be scheduled, where

$$W_e = \beta \max\{p(F_e(u)), p(F_e(v))\}.$$

In the above expression, $F_e(u) = \{f \in E(u) \,|\, \ell_f \leq \ell_e\}$ and $F_e(v) = \{f \in E(v) \,|\, \ell_f \leq \ell_e\}$. When processing $(u, v) \in E$, the edge is scheduled at the earliest time such that no edge incident upon u or v is already scheduled at that time. When e is being processed, we say that e is *active*. Once it becomes active, it remains active for p_e time steps, after which it is *finished*. A not-yet-active edge

can be *waiting* only if none of its neighboring edges are active; otherwise, it is said to be *delayed*. Thus, at any time, an edge is in one of four modes: *delayed*, *waiting*, *active*, or *finished*. When adding new active edges, among those that have done their waiting duty, the algorithm uses the labels on edges as priority. The precise rules are given in the pseudo-code below. Let $wait(e,t)$ denote the number of time steps that e has waited until the end of time step t. Let $Active(t)$ be the set of active edges during time step t. Let $\widetilde{C}_e$ ($\widetilde{C}_u$) be the completion time of edge e (vertex u) in our algorithm.

The pseudo code for the algorithm, as it appears in Figure 4.4, would run in pseudo-polynomial time; however, it is easy to implement the algorithm in strongly polynomial time, by increasing t in each iteration by the smallest remaining processing time of an active edge.

Let us analyze the algorithm. Consider a vertex x and an edge $e = (x, y)$. Let $B_e(x) = \{f \in E(x) \,|\, \ell_f > \ell_e, \widetilde{C}_f < \widetilde{C}_e\}$, i.e., edges in $E(x)$ that have a greater label than that of e, but finish before e in our algorithm. Recall that $F_e(x) = \{f \in E(x) \,|\, \ell_f \leq \ell_e\}$. Note that $e \in F_e(x)$. Let $\overline{F_e(x)} = F_e(x) \setminus \{e\}$.

We analyze the completion time of an arbitrary but fixed vertex $v \in V$. Without loss of generality, let $e_v = (v, w)$ be the edge that finishes last among the edges in $E(v)$. We analyze our algorithm for the case where all edges in $\overline{F_{e_v}(v)} \cup \overline{F_{e_v}(w)}$ finish before e_u in our algorithm. If this is not true then our results can only improve. Let $\widetilde{C}_v$ be the completion time of vertex v in our algorithm.

Lemma 4.5. *For each $v \in V$, $\widetilde{C}_v \leq \beta \max\{p(F_{e_v}(v)), \ell_v\} + p(E(v)) + p(F_{e_v}(w)) + p(B_{e_v}(w))$.*

Proof. Observe that when e_v is in delayed mode it must be that some edge in $\overline{F_{e_v}(v)} \cup B_{e_v}(v) \cup \overline{F_{e_v}(w)} \cup B_{e_v}(w)$ must be active. Hence, we have

$$\begin{aligned}
\widetilde{C}_v &= \widetilde{C}_{e_v} \\
&\leq W_{e_v} + p(F_{e_v}(v)) + p(B_{e_v}(v)) + p(F_{e_v}(w)) + p(B_{e_v}(w)) \\
&= \beta \max\{p(F_{e_v}(v)), p(F_{e_v}(w))\} + p(E(v)) + p(F_{e_v}(w)) + p(B_{e_v}(w)) \\
&\leq \beta \max\{p(F_{e_v}(v)), \ell_v\} + p(E(v)) + p(F_{e_v}(w)) + p(B_{e_v}(w))
\end{aligned}$$

```
SCHEDULE(V, E, ℓ, W)
 1  sort edges in non-decreasing ℓ order
 2  t ← 0
 3  Finished ← Active(t) ← ∅
 4  for e ∈ E do
 5      wait(e, t) ← 0
 6  while Finished ≠ E do
 7      t ← t + 1
 8      Active(t) ← {e | e ∈ Active(t − 1) and e ∉ Active(t − p_e)}
 9      for e ∈ Active(t − 1) \ Active(t) do
10          C̃_e ← t − 1
11          Finished ← Finished ∪ {e}                    // e is finished
12      for e = (u, v) ∈ E \ (Active(t) ∪ Finished) in sorted order do
13          if Active(t) ∩ (E(u) ∪ E(v)) = ∅ and wait(e, t − 1) = W_e
14              then Active(t) ← Active(t) ∪ {e}         // e is active
15      for e = (u, v) ∈ E \ (Active(t) ∪ Finished) do
16          if (Active(t) ∩ (E(u) ∪ E(v)) = ∅)
17              then wait(e, t) ← wait(e, t − 1) + 1     // e is waiting
18              else wait(e, t) ← wait(e, t − 1)         // e is delayed
19  return C̃
```

Figure 4.4: Scheduling with arbitrary processing times.

The last expression follows using Lemma 4.1. □

Lemma 4.6. *For any vertex $v \in V$, $p(F_{e_v}(w)) + p(B_{e_v}(w)) \leq \max\{p(F_{e_v}(v)), \ell_v\} + \frac{1}{\beta}\, p(E(v))$.*

Proof. Let $f = (w, z) \in B_{e_v}(w)$ be an edge with the largest label. When edge f is waiting, it must be that e_v is waiting or some edge in $E(v)$ is being processed. Thus we have

$$\begin{aligned}
\beta \left(p(F_{e_v}(w)) + p(B_{e_v}(w)) \right) &\leq W_f \\
&\leq W_{e_v} + p(E(v)) \\
&= \beta \max\{p(F_{e_v}(v)), p(F_{e_v}(w))\} + p(E(v)) \\
&\leq \beta \max\{p(F_{e_v}(v)), \ell_v)\} + p(E(v))
\end{aligned}$$

The last inequality follows from Lemma 4.1. □

Lemma 4.7. *For any vertex $v \in V$, $\widetilde{C_v} \leq (1+\beta) \max\{p(E(v)), \ell_v\} + (1 + \frac{1}{\beta})\, p(E(v))$*

Proof. The claim follows from Lemmas 4.5 and 4.6, and the fact that $p(F_{e_v}(v)) \leq p(E(v))$. □

Theorem 4.2. *The data migration problem with edges having arbitrary processing times has a 5.83-approximate primal-dual algorithm.*

Proof. Let $G = (V, E)$ be an instance of the data migration problem. The cost the schedule found by our algorithm is given by

$$\begin{aligned}
\sum_{v \in V} w_v \widetilde{C}_v &\leq \sum_{v \in V} w_v \left((1+\beta) \max\{p(E(v)), \ell_v\} + \left(1 + \frac{1}{\beta}\right) p(E(v)) \right) \qquad [\text{ using Lemma 4.7 }] \\
&= (1+\beta) \sum_{v \in V} w_v \max\{p(E(v)), \ell_v\} + \left(1 + \frac{1}{\beta}\right) \sum_{v \in V} w_v\, p(E(v))
\end{aligned}$$

Clearly, $\sum_{v \in V} w_v\, p(E(v)) \leq OPT(G)$. Now, suppose

$$\sum_{v \in V} w_v \max\{p(E(v)), \ell_v\} \leq 2\, OPT(G). \tag{4.5}$$

It follows that

$$\sum_{v \in V} w_v \widetilde{C}_v \leq \left(3 + 2\beta + \frac{1}{\beta}\right) OPT(G).$$

The right hand side in the above expression is minimized for $\beta = \frac{1}{\sqrt{2}}$, which gives a ratio of $3 + 2\sqrt{2} \approx 5.83$. It all boils down to showing (4.5). This is done by relating the left hand side

of (4.5) to the cost of the dual feasible solution obtained by the algorithm, which we denote by $DFS(G)$.

$$
\begin{aligned}
&\sum_{v \in V} w_v \, \max\{p(E(v)), \ell_v\} \\
&\qquad = \sum_{v \in V} \left(z_v + \sum_{\substack{e=(u,v) \\ S(u): e \in S(u)}} y_{S(u)} p_e \right) \max\{p(E(v)), \ell_v\} && [\text{ all } v \in V \text{ are tight }] \\
&\qquad \leq \sum_{v \in V} z_v \, p(E(v)) \; + \sum_{v \in V} \sum_{\substack{e=(u,v) \\ S(u): e \in S(u)}} y_{S(u)} \, p_e \max\{p(E(v)), \ell_v\} && [\text{ using Lemma 4.2 }] \\
&\qquad \leq \sum_{v \in V} z_v \, p(E(v)) \; + \sum_{v \in V} \sum_{\substack{e=(u,v) \\ S(u): e \in S(u)}} y_{S(u)} \, p_e \, p(S(u)) && [\text{ using Lemma 4.3 }] \\
&\qquad = \sum_{v \in V} z_v \, p(E(v)) \; + \sum_{\substack{u \in V \\ S(u) \subseteq E(u)}} \sum_{e \in S(u)} y_{S(u)} \, p_e \, p(S(u)) \\
&\qquad = \sum_{v \in V} z_v \, p(E(v)) \; + \sum_{\substack{u \in V \\ S(u) \subseteq E(u)}} y_{S(u)} \, p(S(u))^2 \\
&\qquad \leq 2 \, DFS(G)
\end{aligned}
$$

Since $DFS(G)$ is a lower bound on $OPT(G)$, it follows that $\sum_{v \in V} w_v \max\{p(E(v)), \ell_v\} \leq 2 \, OPT(G)$.

□

Chapter 5

Adaptive Local Ratio

This chapter deals with the local ratio technique, a well-known paradigm for designing approximation algorithms for combinatorial optimization problems which is closely related to the primal-dual schema. At the heart of every local ratio algorithm is the update step in which a *model* weight function is subtracted from the input weight function. These models usually have a very simple structure that remains "unchanged" throughout the execution of the algorithm. We show that adaptively choosing a model from a richer spectrum of functions can lead to a better local ratio, which translates into better approximations. Indeed, by turning the search for a good model into an optimization problem of its own, we get improved approximations for a data migration problem.

5.1 Overview

The local ratio technique and primal dual schema are two well-known paradigms for designing approximation algorithms for combinatorial optimization problems. Over the years a clear connection between the two paradigms was observed as researchers found primal-dual interpretations for local-ratio algorithms [7, 22] and vice-versa [8, 11]. This culminated with the work of Bar-Yehuda and Rawitz [12] showing their equivalence under a fairly general and encompassing definition of primal-dual and local-ratio algorithms. For a survey of results and a historical account on the local ratio technique see [13].

At a very high level a, local-ratio algorithm starts by decomposing the input weight function w into a positive linear combination of *models* $\hat{w}_i$, that is, $w = \epsilon_1\hat{w}_1 + \ldots + \epsilon_n\hat{w}_k$ and $\epsilon_i \geq 0$. Guided by this process, a solution S is constructed such that $\hat{w}_i(S) \leq \alpha\,\hat{w}_i(A)$ for any feasible solution A, we refer to α as the *local ratio* of $\hat{w}_i$. As a result, S is α-approximate with respect to w [11].

Typically the models used in local-ratio approximation algorithms are 0-1 functions or simple aggregates of structural features of the problem at hand. (In the primal-dual parlance this corresponds to increasing some dual variables uniformly when constructing the dual solution.) Furthermore, the structure of the models remains "unchanged" throughout the execution of the algorithm. For example, for the vertex cover problem, Bar-Yehuda and Even [11] set the weight of the endpoints of a yet-uncovered edge to 1 and the remaining vertices to 0, while Clarkson [27] chooses a set of edges forming a star and sets the weight of each vertex to the number of yet-uncovered star edges incident on it.

In this chapter we demonstrate that adaptively choosing a model from a richer spectrum of weight functions can lead to a better local ratio, and thus to better approximations. Indeed, by turning the search for a good model into an optimization problem of its own, we get improved approximations for the data migration problem with the objective minimizing sum of vertex completion times. We hope our findings encourage the study of non-uniform updates for local-ratio or primal-dual algorithms; perhaps in some cases, as in our problem, this may help realize the full potential of these techniques.

In Section 5.2, we cast the primal-dual algorithm from the previous chapter as a local-ratio algorithm and provide a family of instances showing the analysis is tight. To overcome these difficult instances we propose to adaptively choose a model minimizing the local ratio and formulate the problem of finding such a model as a linear program. Interestingly, our algorithm is neither purely combinatorial nor LP rounding, but lies somewhere in between. Every time the weight function needs to be updated, an LP is solved to find the best model. These LPs are much smaller that the usual LP formulations, so the our scheme should be faster than an LP rounding algorithm.

In the analysis we show that the models found using the LP exhibit a better local ratio than the usual 0-1 models. Somewhat surprisingly a precise characterization of the local ratio can be derived analytically. We prove that the overall scheme is a $1 + \phi$ approximation, where $\phi = \frac{1+\sqrt{5}}{2}$ is the Golden ratio, and give a family of instances achieving this ratio.

To derive the worst-case local ratio of our scheme we use a method similar to the factor-revealing LP approach of Jain et al. [66], which has been successfully applied in the analysis of many greedy heuristics [5, 16, 64, 66]. The idea there is to formulate as an LP the problem of building a worst-case instance maximizing the approximation ratio achieved by the heuristic at hand, and then upperbound the cost of this LP by producing a dual solution. We too formulate as a mathematical program the problem of building a worst-case instance maximizing the local ratio. The main difference here is that, since we already use an LP to guide our local-ratio algorithm, the resulting factor-revealing program is non-linear. Even though we cannot solve this program numerically, we are still able to prove a tight bound of $1 + \phi$ on its cost.

5.2 Algorithmic Framework

The input to our problem consists of a transfer graph $G = (V, E)$ and a weight function $w : V \to R^+$. For ease of exposition, we assume G is simple[1]. A feasible schedule $S : E \to Z^+$ is a proper edge-coloring of G, that is, if two edges $e_1 \neq e_2$ are incident on the same vertex then $S(e_1) \neq S(e_2)$. We are to find a scheduling minimizing $w(S) = \sum_{u \in V} w(u) \max_{v \in N(u)}\{S(u, v)\}$.

Let us cast the primal-dual algorithm of Section 4.3 as a local-ratio algorithm; in the process, we generalize it slightly. Algorithm ALR has two stages: labeling and scheduling. The labeling stage assigns a label ℓ_u to every $u \in V$; these labels are then used to guide the scheduling stage. The pseudo-code of ALR is given in Figure 5.1.

Initially every node is unlabeled, i.e., $\ell_v = \textbf{nil}$ for all $v \in V$. Denote the set of unlabeled neighbors of u with $\mathrm{UN}(u) = \{v \in \mathrm{N}(u) \,|\, \ell_v = \textbf{nil}\}$. In each iteration, choose a node u with the maximum number of unlabeled neighbors $\Delta = |\mathrm{UN}(u)|$. Then choose a *model* $\hat{w} : V \to R^+$ with support in $\mathrm{UN}(u)$, find the largest $\epsilon \geq 0$ such that $\epsilon\hat{w} \leq w$, and set $w \leftarrow w - \epsilon\hat{w}$. As a result, at least one vertex in $\mathrm{UN}(u)$ has zero weight in the updated w; set the label of these vertices to Δ. This continues until all vertices are labeled and $w = 0$.

Once the labels are computed, the edges $(u, v) \in E$ are sorted in increasing value of

[1] In Section 5.5 we show how to remove this assumption

```
ALR(V, E, w)
 1  // LABELING STAGE
 2  for u ∈ V do
 3      ℓ_u ← nil
 4  while there exists an unlabeled vertex do
 5      choose u ∈ V maximizing Δ = |UN(u)|
 6      choose ŵ with support in UN(u)
 7      w ← w − min { w(u)/ŵ(u) | ŵ(u) > 0 } ŵ
 8      for v ∈ UN(u) | w(v)=0 do
 9          ℓ_v ← Δ
10  // SCHEDULING STAGE
11  sort the edges in non-decreasing ℓ order
12  S ← empty schedule
13  for e ∈ E in sorted order do
14          add e to S as early as possible
15  return S
```

Figure 5.1: Pseudo-code for ALR.

$\min\{\ell_u, \ell_v\}$, breaking ties with $\max\{\ell_u, \ell_v\}$. The edges are scheduled greedily in sorted order: Start with the empty schedule, and add the edges, one by one in sorted order, to the current schedule as early as possible without creating a conflict.

The labels guide the scheduling phase and let us bound the finishing time of the vertices.

Lemma 5.1. *In S, every vertex $v \in V$ finishes no later than $\ell_v + d_v - 1$.*

Proof. Let (v, y) be the edge incident on v that is scheduled the latest in S. Note that the algorithm can schedule at most $d_v - 1$ edges incident on v before scheduling (v, y). How many edges incident

on y were scheduled before (v, y)? We claim that at most $\ell_v - 1$. Thus, (v, y) must be scheduled not later than $\ell_v + d_v - 1$. To prove our claim, let $X = \{x \in \mathrm{N}(y) \,|\, \ell_x \leq \ell_v\}$. The quantity we want to bound is clearly at most $|X| - 1$. Notice that the value of the labels assigned in Line 9 of ALR can only decrease with time. Consider the first iteration of the algorithm in which the node u chosen in Line 4 of ALR was such that $|\mathrm{UN}(u)| = \ell_v$; at this point in time $\mathrm{UN}(y) = X$. Since u is chosen to maximize $|\mathrm{UN}(u)|$ it follows that $|X| = |\mathrm{UN}(y)| \leq |\mathrm{UN}(u)| = \ell_v$. □

As it stands, the algorithm is underspecified: We have not described how the model $\hat{w}$ chosen in Line 5. It is important to realize though, that Lemma 5.1 holds regardless of our choice of $\hat{w}$. In Section 4.3 we used $\hat{w}(v) = \mathbb{I}\,[v \in \mathrm{UN}(u)]$ as a model, where $\mathbb{I}\,[\cdot]$ is a 0-1 indicator function. Recall that this yields a 3-approximation. In order to gain some intuition on the local ratio technique let us show that the local ratio of $\hat{w}(v) = \mathbb{I}\,[v \in \mathrm{UN}(u)]$ is at most 3.

Lemma 5.2. *If Line 5 always uses $\hat{w}(v) = \mathbb{I}\,[v \in \mathrm{UN}(u)]$ then $\hat{w}(S) \leq 3\,\hat{w}(A)$ for any schedule A.*

Proof. An obvious lowerbound on $\hat{w}(A)$ is $\sum_{v \in V} \hat{w}(v) d_v = \sum_{v \in \mathrm{UN}(u)} d_v$. Furthermore, since nodes in $\mathrm{UN}(u)$ share u as a common neighbor, it follows that $\hat{w}(A) \geq \sum_{i \in [\Delta]} i > \Delta^2/2$, where $\Delta = |\mathrm{UN}(u)|$. On the other hand, by Lemma 5.1

$$\hat{w}(S) \leq \sum_{v \in \mathrm{UN}(u)} (\ell_v + d_v - 1) \leq \Delta^2 + \sum_{v \in \mathrm{UN}(u)} d_v < 3\,\hat{w}(A).$$

The second inequality follows from the fact that the value of labels only decreases with time. □

Since S is a 3-approximation with respect to every model and the input weight function w is a positive linear combination of these models, it follows that S is 3-approximate with respect to w as well. It is worth pointing out that the bound on the local ratio of Lemma 5.2 is essentially tight with respect to Lemma 5.1. To see this, consider what happens when the ith vertex in $\mathrm{UN}(u)$ has degree $d_i = i$. Of course, this alone does not imply a tight bound on the overall approximation guarantee, but as we will see in Lemma 5.4, there is a family of instances where the algorithm in Section 4.3 produces a schedule whose cost is $3 - o(1)$ times optimum.

Notice that the degree sequence $d_i = i$ can be easily circumvented if we use a different model: Instead of $\hat{w}(v) = \mathbb{I}\,[v \in \mathrm{UN}(u)]$, which has a local ratio ≈ 3, use the model that just sets

to 1 the node in $\mathrm{UN}(u)$ with degree Δ, which in this case has a local ratio of ≈ 2. Indeed, in general choosing the best model between $\hat{w}(v) = \mathbb{I}[v \in \mathrm{UN}(u)]$ and $\hat{w}(v) = \mathbb{I}[v \in \mathrm{argmax}_{x \in \mathrm{UN}(u)} d_x]$ leads to a local ratio of $3-\beta^2 \approx 2.802$, where $\beta \approx 0.445$ is a root of the polynomial $p(x) = x^3 - x^2 - 2x + 1$. The ratio can be achieved with with the degree sequence $d_i = \min\{i, \lceil \beta\Delta \rceil\}$, which again, can be shown to be tight using Lemma 5.4.

Besides a modest improvement in the approximation guarantee, this suggests a general line of attack: In each iteration find a model that minimizes the local ratio.

5.3 Minimizing the local ratio

The abstract problem we are to solve is the following: Given a sequence $d = (d_1, d_2, \ldots, d_\Delta)$ corresponding to the degrees of vertices in $\mathrm{UN}(u)$, find weights $\hat{w} = (\hat{w}_1, \hat{w}_2, \ldots, \hat{w}_\Delta)$ minimizing the local ratio of $\hat{w}$.

In order to evaluate the goodness of a given model $\hat{w}$ we first need an upper bound on $\hat{w}(S)$, where S is the schedule produced by ALR. For this we use Lemma 5.1 and the fact that the values of labels assigned in Line 7 of ALR can only decrease with time.

Definition 5.1. $\mathrm{UB}(d, \hat{w}) = \sum_{i \in [\Delta]} \hat{w}_i \, (d_i + \Delta - 1)$.

Similarly, we need a lower bound on $\hat{w}(A)$, where A can be any schedule. Note that A must schedule the edges from $\mathrm{UN}(u)$ to u at different time slots; this induces a total order on $\mathrm{UN}(u)$, which we denote by the permutation $\sigma : [\Delta] \to [\Delta]$. Notice that vertex i cannot finish earlier than $\sigma(i)$, nor earlier than d_i since all edges incident on i must be scheduled before it finishes.

Definition 5.2. $\mathrm{LB}(d, \hat{w}) = \min_{\sigma:[\Delta]\to[\Delta]} \sum_{i \in [\Delta]} \hat{w}_i \max\{d_i, \sigma(i)\}$.

It follows from the above discussion that $\hat{w}(S) \leq \mathrm{UB}(d, \hat{w})$ and $\mathrm{LB}(d, \hat{w}) \leq \hat{w}(A)$ for all A. Hence, the minimum local ratio for d can be expressed as a function of UB and LB.

Definition 5.3. *Let* $\rho(d) = \inf_{\hat{w}} \dfrac{\mathrm{UB}(d, \hat{w})}{\mathrm{LB}(d, \hat{w})}$ *be the minimum local ratio of* d.

We now turn our attention to the problem of computing a model $\hat{w}$ with a local ratio that achieves $\rho(d)$. This can be done using the program $\min\{\mathrm{UB}(d, \hat{w}) \,|\, \mathrm{LB}(d, \hat{w}) \geq 1, \hat{w} \geq 0\}$, which

can be re-written as a linear program.

$$\begin{aligned}
&\min \ \textstyle\sum_{i\in[\Delta]}(d_i+\Delta-1)\,\hat{w}_i \\
&\text{subject to} \\
&\textstyle\sum_{i\in[\Delta]}\hat{w}_i\,\max\{d_i,\sigma(i)\}\geq 1 \qquad \forall\sigma:[\Delta]\to[\Delta] \qquad (5.1) \\
&\hat{w}_i\geq 0 \qquad \forall\, i\in[\Delta] \qquad (5.2)
\end{aligned} \tag{LP1}$$

Clearly, LP1 computes a model $\hat{w}$ with local ratio $\rho(d)$. Even though LP1 is exponentially large, it can be solved in polynomial time using the ellipsoid method—the separation oracle involves solving a minimum assignment problem. The ellipsoid method, however, is not practical, so below we derive a more succinct formulation.

$$\begin{aligned}
&\min \ \textstyle\sum_{i\in[\Delta]}(d_i+\Delta-1)\,\hat{w}_i \\
&\text{subject to} \\
&\textstyle\sum_{i\in[\Delta]}(y_i-z_i)\geq 1 \qquad (5.3) \\
&y_i-z_j\leq \max(d_i,j)\,\hat{w}_i \qquad \forall\, i,j\in[\Delta] \qquad (5.4) \\
&y_i,z_i,\hat{w}_i\geq 0 \qquad \forall\, i\in[\Delta] \qquad (5.5)
\end{aligned} \tag{LP2}$$

The idea behind LP2 is to replace the cost of the assignments in (5.1) with their dual lower bound. If $(\hat{w},y,z)$ is a feasible solution of LP2 then $\hat{w}$ is a feasible solution of LP1, since for any assignment σ,

$$1\leq\sum_{i\in[\Delta]}(y_i-z_i)\leq\sum_{i\in[\Delta]}(y_i-z_{\sigma(i)})\leq\sum_{i\in[\Delta]}\hat{w}_i\,\max\{d_i,\sigma(i)\}.$$

Since the polytope of the assignment problem is integral [30, Chapter 5.3], the converse also holds. In other words, for a fixed $\hat{w}$, there exist y and z obeying (5.4) such that

$$\min_{\sigma}\sum_{i\in[\Delta]}\hat{w}_i\,\max\{d_i,\sigma(i)\}=\sum_{i\in[\Delta]}(y_i-z_i).$$

This finishes the description of ALR from Figure 5.1. Namely, in each iteration of the labeling stage, Line 5 must solve LP2 to find the best model for the degree sequence of nodes in $\mathrm{UN}(u)$.

Definition 5.4. *Let* $\rho = \sup_d \rho(d)$ *and* $\rho_\Delta = \max_{d:|d|=\Delta} \rho(d)$.

Theorem 5.1. ALR *is a* ρ*-approximation for the data migration problem and this is tight.*

The proof that the algorithm is a ρ-approximation is similar to that of Lemma 5.2. Let S be the schedule produced by ALR. For every model $\hat{w}$ used by ALR, by definition, S is ρ-approximate with respect to $\hat{w}$. Since the input weight function is a linear combination of these models S is a ρ-approximation. The tightness claim follows from the next two lemmas.

Lemma 5.3. *For any* Δ, $\rho_\Delta < \rho_{2\Delta}$.

Proof. Let d be such that $\rho_\Delta = \rho(d)$. Then define $d'_{2i-1} = d'_{2i} = 2d_i$. Let $(\hat{w}', y', z')$ be an optimal solution of LP2 for d' with cost $\rho(d')$. Define $y_i = y'_{2i-1} + y'_{2i}$, $z_i = z'_{2i-1} + z'_{2i}$ and $\hat{w}_i = 2(\hat{w}'_{2i-1} + \hat{w}'_{2i})$. Then $(\hat{w}, y, z)$ is a feasible solution for d with cost strictly less than $\rho(d')$. Thus $\rho_\Delta = \rho(d) < \rho(d') \leq \rho_{2\Delta}$. □

Hence, for our lower bound we only need to worry about large values of Δ. Let d be a degree sequence of length Δ such that $\rho_\Delta = \rho(d)$. For technical reasons, assume that $d_i < \Delta$ (the local ratio is smaller in this case) and $d_i > 1$ for $i \geq 2$ (otherwise we can use a stronger upper bound). We show that for large Δ, the algorithm can produce solutions arbitrarily close to ρ_Δ.

Lemma 5.4. *Suppose that Line 5 of* ALR *always chooses the model* $\hat{w}$ *when the vertices in* $\mathrm{UN}(u)$ *have degrees* $d_1, \ldots, d_\Delta$, *where* $d_i > 1$ *for* $i \geq 2$ *and* $d_i < \Delta$ *for all* i. *Then the algorithm can produce a schedule with cost* $\frac{\mathrm{UB}(d,w)}{\mathrm{LB}(d,w)}\left(1 - \frac{1}{\Delta}\right)$ *times the optimum.*

Proof. Consider the instance in Figure 5.2: a tree with four levels. The ith node in the second level has weight of $\hat{w}_i$, nodes in other levels have weight zero. The root has degree Δ; the ith node in the second level has degree d_i, i.e., it has $d_i - 1$ children; nodes in the third level have degree Δ.

Consider an execution of the algorithm that chooses the root in the first iteration, as a result, all nodes in the second level get a label of Δ. In the next $\sum_i (d_i - 1)$ iterations the leaves are labeled $\Delta - 1$. Finally, the remaining nodes get a label less than $\Delta - 1$. Now consider a node in the third level, note that the children are labeled $\Delta - 1$, while its parent is labeled Δ. Therefore,

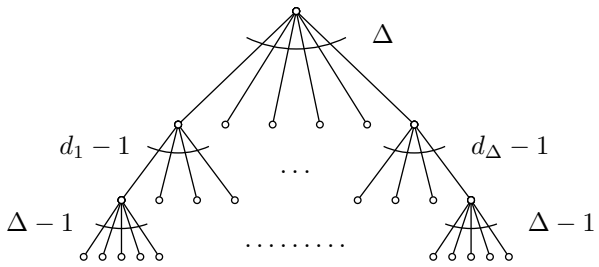


Figure 5.2: Tight instance for ALR.

the edges between the third and forth level will be scheduled before those between the second and the third level. Also assume the edges incident on the root are scheduled from right to left. As a result, the ith node in the second level finishes by $d_i + \Delta - 2$. On the other hand, the optimal solution has cost precisely $\mathrm{LB}(d, \hat{w})$. Therefore, the approximation ratio is precisely

$$\frac{\sum_{i\in[\Delta]} \hat{w}_i \, (d_i + \Delta - 2)}{\mathrm{LB}(d, \hat{w})} = \frac{\mathrm{UB}(d, \hat{w}) - \sum_{i\in[\Delta]} \hat{w}_i}{\mathrm{LB}(d, \hat{w})} \geq \alpha \left(1 - \frac{1}{\Delta}\right).$$

□

As a corollary, we get that the analysis of Section 4.3 is essentially tight since the model $\hat{w}_i = 1$ has a local ratio of $3 - \frac{2}{\Delta+1}$ for the degree sequence $d_i = i$.

However, if we use LP2 to find the best model then the approximation factor becomes ρ, which by Lemmas 5.3 and 5.4 is tight. It only remains to bound ρ. Somewhat surprisingly, a precise characterization in terms of the Golden ratio $\phi = \frac{1+\sqrt{5}}{2} \approx 1.618$ can be derived analytically.

Theorem 5.2. $\rho = 1 + \phi$.

The next section is devoted to proving this theorem. Figure 5.3 shows ρ_Δ for small values of Δ obtained through exhaustive search.

5.4 A tight bound for ρ

We start by showing that $\rho \leq 1 + \phi$. In a sense, we need to argue that every degree sequence d has a good model. Recall that the best model can be found with the linear program LP2. At first glance this may seem like an obstacle since we are essentially treating LP2 as a black box;

Δ	ρ_Δ	Δ	ρ_Δ
1	1	20	2.4453
2	1.5	30	2.5006
3	1.7273	40	2.5275
4	1.9310	50	2.5447
5	2.0115	60	2.5556
6	2.1042	70	2.5667
7	2.1863	80	2.5728
8	2.2129	$\vdots$	$\vdots$
9	2.2589		
10	2.2857	∞	$1+\phi$

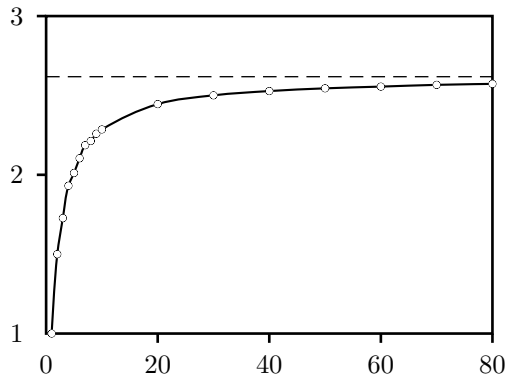


Figure 5.3: Experimental evaluation of $\rho_\Delta = \rho_{d:|d|=\Delta}\rho(d)$.

however, we can exploit linear duality to show the upper bound.

The idea is to replace LP2 with its dual problem LP3 given below. By the Strong Duality Theorem the optimal solution of LP2 and LP3 have the same cost.

$$\max\ \alpha$$

subject to

$$\begin{aligned}
\textstyle\sum_{j\in[\Delta]} x_{ij} &\geq \alpha && \forall\, i \in [\Delta] \\
\textstyle\sum_{i\in[\Delta]} x_{ij} &\leq \alpha && \forall\, j \in [\Delta] \\
\textstyle\sum_{j\in[\Delta]} \max(d_i, j)\, x_{ij} &\leq d_i + \Delta - 1 && \forall\, i \in [\Delta] \\
x_{ij}, \alpha &\geq 0 && \forall\, i, j \in [\Delta]
\end{aligned} \tag{LP3}$$

Therefore, $\rho(d)$ equals the cost of the optimal solution of LP3 for the sequence d. Recall that $\rho_\Delta = \max_{d:|d|=\Delta} \rho(d)$. Suppose we modify LP3 by letting $d_1, \ldots, d_\Delta$ be variables in $[\Delta]$. The result is a mathematical program for ρ_Δ, albeit a non-linear one.

$$\max\ \alpha$$

subject to

$$\begin{aligned}
\textstyle\sum_{j\in[\Delta]} x_{ij} &\geq \alpha && \forall\, i \in [\Delta] && (5.6)\\
\textstyle\sum_{i\in[\Delta]} x_{ij} &\leq \alpha && \forall\, j \in [\Delta] && (5.7)\\
\textstyle\sum_{j\in[\Delta]} \max(d_i, j)\, x_{ij} &\leq d_i + \Delta - 1 && \forall\, i \in [\Delta] && (5.8)\\
x_{ij}, \alpha &\geq 0 && \forall\, i, j \in [\Delta] && (5.9)\\
d_i &\in [\Delta] && \forall i \in [\Delta] && (5.10)
\end{aligned} \tag{NLP$_\Delta$}$$

The plan is to show that NLP$_\Delta$ is upperbounded by $1 + \phi$. To that end, let us first derive some structural properties about the solutions for NLP$_\Delta$.

Lemma 5.5. *There is an optimal solution (x, d, α) for* NLP$_\Delta$ *such that for all i:*

i) $d_i = \min\{k \mid \sum_{j=1}^{k} x_{i,j} \geq 1\}$,

ii) $x_{i,j} = 0$ *for all* $j \leq d_i - 2$, *and*

iii) If $x_{i,d_i-1} \neq 0$ *then* $x_{k,d_i} = 0$ *for all* $k < i$.

Proof. Suppose $d_i < \min\{k \mid \sum_{j=1}^{k} x_{i,j} \geq 1\}$. If d_i obeys (5.8) then we can increment d_i by 1, since this increases the left hand side of (5.8) by less than 1 and its right hand side by 1. Similarly if $d_i > \min\{k \mid \sum_{j=1}^{k} x_{i,j} \geq 1\}$ we can safely decrease d_i without violating feasibility. Thus, from now on we can assume that i) is always satisfied.

First sort the rows of x so that $d_1 \leq d_2 \leq \ldots \leq d_\Delta$. The plan is to modify x row by row until ii) and iii) are satisfied. For the base case $i = \Delta$. Suppose there exists $k < d_i$ such that $x_{i,k} > 0$. That means $x_{i,d_i} < \alpha$. By (5.8) and the fact that $i = \Delta$, there must be an $i' < i$ such that $x_{i',d_i} > 0$. Let $\epsilon = \min\{x_{i,k}, x_{i',d_i}\}$. Decrease $x_{i,k}$ and x_{i',d_i} by ϵ, and increase x_{i,d_i} and $x_{i',k}$ by ϵ. Note that the update does not affect constraint (5.8) for i or i'. However, it may decrease

$d_{i'}$ (as function of x) in which case we must resort the rows. The update is repeated until $x_{i,k} = 0$ for all $k < d_i$.

Assuming that rows in $[i+1, \Delta]$ obey ii) and iii), we show how to modify the ith row. The idea is very similar to the base case, the only difference is that if there exists $k < d_i$ such that $x_{i,k} > 0$ then the fact that $x_{i,d_i} < \alpha$ is not enough to conclude the existence of $i' < i$ such that $x_{i',d_i} > 0$ since the remaining non-zero entries in the d_ith column may be in rows $i' > i$. However, if this is the case then by iii) we have $x_{i',d_i-1} = 0$ for $i' > i$ since $x_{i,d_i} > 0$. Thus, we can safely do the update for x_{i,d_i-1} until $x_{i,k} = 0$ for all $k \leq d_i - 2$. Also note that if we have to switch from x_{i,d_i} to x_{i,d_i-1} then $x_{i',d_i} = 0$ for $i' < i$. Putting all together we get properties ii) and iii) for the ith row. □

Lemma 5.6. *For any Δ, the objective value of* NLP_Δ *is upperbounded by* $(1+\phi) + \frac{1}{\Delta-1}$.

Proof. Let (x, d, α) be an NLP_Δ solution satisfying Lemma 5.5. Our goal is to show that $\alpha \leq 1+\phi+\frac{1}{\Delta-1}$. Let k the largest index such that $(\phi-1)k < d_k$. If $k = \Delta$ then consider constraint (5.8) for $i = \Delta$

$$\sum_{j\in[\Delta]} \max\{d_\Delta, j\} x_{i,j} \leq d_\Delta + \Delta - 1,$$

$$\alpha\, d_\Delta \leq d_\Delta + \Delta. \tag{5.11}$$

Where (5.11) follows from (5.7). Rearranging the terms in (5.11) we get

$$\alpha \leq \frac{\Delta}{d_\Delta} + 1 < \frac{\Delta}{(\phi-1)\Delta} + 1 = \phi + 1.$$

Let us consider the case $k < \Delta$. Adding up constraints (5.8) for all i such that $k < i \leq \Delta$

$$\sum_{i=k+1}^{\Delta} \sum_{j=1}^{\Delta} \max\{d_i, j\}\, x_{i,j} \leq \sum_{i=k+1}^{\Delta} (d_i + \Delta - 1),$$

$$\sum_{j=d_k}^{d_k+\Delta-k-1} \alpha\, j \leq \sum_{i=k+1}^{\Delta} (\phi-1)i + (\Delta-k)(\Delta-1). \tag{5.12}$$

Where (5.12) follows from the fact that $x_{i,j} = 0$ for all $i > k$ and $j < d_k$ (by Lemma 5.5), every row and column of x add up to α (by constraints (5.6) and (5.7)), and $d_i \leq (\phi-1)i$ for all $i > k$

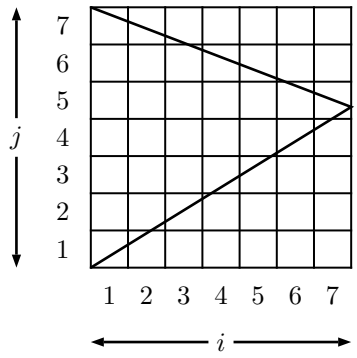


Figure 5.4: How to construct a solution for NLP_Δ.

(by definition of k). Simplifying (5.12) using $d_k > (\phi - 1)k$, we get

$$\alpha \leq \frac{(\phi-1)(k+1+\Delta)+2(\Delta-1)}{2(\phi-1)k+\Delta-k-1}.$$

The right hand side of (5.4) is maximized for $k = \Delta - 1$. Thus,

$$\alpha \leq \frac{(\phi-1)2\Delta+2(\Delta-1)}{2(\phi-1)(\Delta-1)} = \frac{2(\phi\Delta-1)}{2(\phi-1)(\Delta-1)} = (1+\phi)+\frac{1}{\Delta-1}. \tag{5.13}$$

Which is precisely what we needed.

□

By Lemmas 5.3 and 5.6, we get that $\rho \leq 1 + \phi$. The next lemma finishes the proof of Theorem 5.2 by showing that $\rho \geq 1 + \phi$.

Lemma 5.7. *For every Δ, the objective value of NLP_Δ is lowerbounded by $(1+\phi)\left(1-\frac{3}{\Delta}\right)$.*

Proof. The plan is to construct a feasible solution (x, d, α) for NLP_Δ with $\alpha = (1+\phi)\left(1-\frac{3}{\Delta}\right)$. Since the cost of the optimal solution can only be larger than this, the lemma follows.

Imagine drawing on the Cartesian plane a Δ by Δ grid, and lines $l_1 = (\phi - 1)x$ and $l_2 = \Delta - (2 - \phi)x$. See Figure 5.4 for a picture of the grid for $\Delta = 7$. Define the cell (i, j) to be the square $[i-1, i] \times [j-1, j]$. Suppose the intersection of cell (i, j) and l_1 defines a segment of length L then we set $x_{i,j} = c\,L\,\frac{\phi}{\sqrt{3-\phi}}$, where c is a constant, which will be chosen shortly to make the solution feasible. Similarly, if the intersection of cell (i, j) and l_2 has length L then we

set $x_{i,j} = c\,L\,\frac{1}{\sqrt{6-3\phi}}$. It is easy to check that for every j and i:

$$\sum_{j' \in [\Delta]} x_{i,j'} = \sum_{i' \in [\Delta]} x_{i',j} = c\,(1+\phi).$$

Hence, due to constraints (5.6) and (5.7) the cost of the dual solution is $\alpha = c\,(1+\phi)$. Let $d_i = \lceil (\phi-1)i \rceil$. It only remains to find the largest c that satisfying (5.8). To that end we demand for every $i \in [\Delta]$ that

$$\lceil (\phi-1)i \rceil\, c\,\phi + \lceil \Delta - (2-\phi)(i-1) \rceil\, c \leq \lceil (\phi-1)i \rceil + \Delta - 1.$$

It can be verified that the above inequality holds for $c = 1 - \frac{3}{\Delta}$. □

5.5 Generalizations

Throughout the paper we have assumed that the transfer graph G is simple. In practice G is typically a multigraph, so we now show how to modify the description of ALR given in Figure 5.1 to handle multigraphs. Let $E(u,v)$ denote the set of edges between u and v. First, in Line 4 of ALR we choose a vertex u maximizing $\Delta = \sum_{v \in \mathrm{UN}(u)} |E(u,v)|$. To compute the model $\hat{w}$ we create a degree sequence $d'_1, \ldots d'_\Delta$ by making $|E(u,v)|$ copies of d_v for each $v \in N(u)$. Solve LP2 to get weights $\hat{w}'$, and then set $\hat{w}(v)$ to be the sum of $\hat{w}'_i$ for the indices i induced by d_v. These two modifications make the discussion on the approximation ratio for ALR to carry over to multigraphs.

Chapter 6

Dual Fitting

In this last chapter we study the data migration problem with the objective to minimize the sum of completion times of edges. We define the notion of strongly minimal schedule and give an efficient algorithm to compute this schedule in bipartite graphs. We use dual fitting to prove that these schedules are $\sqrt{2}$-approximate. To show that our analysis is almost tight, we provide an instance with a strongly minimal schedule that is a 1.375-factor away from optimum.

6.1 Overview

The problem of scheduling the edges of a graph to minimize the sum of their completion times can be cast as an edge coloring problem: Given $G = (V, E)$ we want to partition the edge set E into matchings $M_1, \ldots, M_k$ as to minimize $\sum_i i\,|M_i|$. Indeed, this problem is also known as minimum sum edge coloring.

A schedule $M_1, \ldots, M_k$ is said to be *minimal* if every matching M_i is maximal with respect to $G \setminus \cup_{j<i} M_j$. Bar-Noy et al. [6] showed that *any* minimal schedule is 2-approximate. The main result of this chapter is to identify a stronger minimality requirement that results in a better approximation guarantee.

Data migration in bipartite graphs is equivalent to a variant of open shop scheduling in which we want to minimize the sum of operation completion times. Marx [82] showed that the problem is APX-hard. Gandhi et al. [37] showed that using the sum-coloring algorithm of Halldórsson et al. [52] one can obtain a 1.796 approximation guarantee.

In Section 6.2 we define the notion of *strongly minimal* schedule. Using dual fitting we prove that any strongly minimal schedule is $\sqrt{2}$-approximate. We also show that a strongly minimal schedule always exist for bipartite graphs and that they can be computed in polynomial time. In

Section 6.2.1 we show that our analysis is almost tight by giving an example on which the algorithm gives a 1.375-approximate solution. Finally, in Section 6.2.2 studies the integrality gap of the LP relaxation used in our proofs.

6.2 Strongly minimal schedules

Let us define the notion of strongly minimal schedule and prove that it is $\sqrt{2}$-approximate.

Definition 6.1. *A schedule $M_1, \ldots, M_k$ of G is strongly minimal if, for all $1 \leq b \leq k$, the b-matching $\cup_{i \leq b} M_i$ is maximal with respect to G.*

Theorem 6.1. *Any strongly minimal schedule is $\sqrt{2}$-approximate.*

Proof. The high level idea of the proof is to *assign* every edge to at least one of its endpoints. Each vertex is responsible for paying for the cost of the edges assigned to it. In order to pay for this cost each vertex charges a lower bound on the completion time of the edges assigned to it.

Let $(u, v) \in M_i$, we say endpoint u is *full* if u is matched in all $M_{j<i}$. We consider the endpoints of edges in M_1 to be full. Notice that every edge $(u, v) \in M_i$ must have at least one full endpoint, otherwise $\cup_{j<i} M_j + (u, v)$ would be a valid $(i-1)$-matching, which contradicts the fact that the schedule is strongly minimal. If both endpoints of (u, v) are full then the edge is *half-assigned* to u and v. Otherwise the edge is *fully-assigned* to the one full endpoint.

Every vertex u is responsible for the cost of edges assigned to it. If an edge is half-assigned to u, then u pays for half of its completion time; if the edge is fully-assigned then u pays in full. Let s_1 and s_2 be the number of half-assigned and fully-assigned edges to u respectively. Notice that all edges assigned to u must belong to M_j for some $j \leq s_1 + s_2$. Think of u as paying $\frac{1}{2}$ of the completion time of *all* edges assigned to it, plus an additional $\frac{1}{2}$ for the fully-assigned edges, which in the worst case will be scheduled the latest,

$$u \text{ must pay} \leq \frac{1}{2} \sum_{i=1}^{s_1+s_2} i + \frac{1}{2} \sum_{i=s_1+1}^{s_1+s_2} i.$$

Vertex u will pay this amount by charging the completion time (in the optimal solution) of the edges assigned to it. Fully-assigned edges are charged a soon-to-be-determined ρ factor, and

half-assigned edges are charged $\frac{\rho}{2}$. This constitutes u's budget. How fast can the optimal solution possibly schedule these edges?

$$u\text{'s budget} \geq \frac{\rho}{2}\sum_{i=1}^{s_1+s_2} i + \frac{\rho}{2}\sum_{i=1}^{s_2} i.$$

Notice that every edge is charged at most to an extent of ρ: fully-assigned edges are charged ρ once, from a single endpoint, and half-assigned edges are charged $\frac{\rho}{2}$ twice, once from each endpoint. Thus, strongly greedy schedules are ρ-approximate. The discrepancy between the upper and lower bound on the completion times of edges assigned to u is due to fully-assigned edges which are scheduled the latest in the upper bound, and the earliest in the lower bound. We need to determine the smallest ρ such that u's budget is enough to cover u's payment, namely

$$\frac{(s_1+s_2)(s_1+s_2+1)}{4} + \frac{(1s_1+s_2+1)s_2}{4} \leq \rho\frac{(s_1+s_2)(s_1+s_2+1)}{4} + \rho\frac{s_2(s_2+1)}{4}.$$

Or equivalently,

$$(s_1+s_2)^2 + (2s_1+s_2)s_2 \leq \rho(s_1+s_2)^2 + \rho s_2^2 + (\rho-1)(s_1+2s_2).$$

Let $\alpha = \frac{s_2}{s_1+s_2}$, since $\rho > 1$ the above follows provided

$$\frac{1+2\alpha-\alpha^2}{1+\alpha^2} \leq \rho.$$

The left hand side is maximized for $\alpha = \sqrt{2}-1$, which yields $\sqrt{2} \leq \rho$ □

Strongly minimal schedules are not guaranteed to exist for general graphs, for example a triangle does not allow a strongly minimal schedule. Nevertheless, we can still show that in bipartite graphs they always exist and can be computed in polynomial time. The bipartite, is an interesting and nontrivial case: It is a variant of the open shop scheduling problem in which we want to minimize the sum of completion time of operations [37]. The problem is APX-hard [82] and the best known approximation guarantee for it is 1.796 [37].

Theorem 6.2. *The procedure* FIND STRONGLY MINIMAL *is a* $\sqrt{2}$*-approximation for minimizing the sum of completion times of unit length operations in open shop scheduling.*

```
FIND STRONGLY MINIMAL(G)
1  for i ← Δ to 1 do
2      M_i ← a matching incident to all vertices of G with degree i
3      G ← G \ M_i
4  return M_1, M_2, ..., M_Δ
```

Figure 6.1: Computing a strongly minimal schedule.

Proof. In each iteration, the procedure FIND STRONGLY MINIMAL computes a matching incident to the maximum degree vertices of G and removes the matching from G. This continues until all edges have been removed. The matchings found are then scheduled in reverse order. Because the degree of G decreases by one with each iteration, the algorithm finishes after Δ iterations, here Δ is the degree of the original graph.

Let us argue that the schedule found is strongly minimal. Let $e \in M_i$ and $b < i$, we want to show that e cannot be added to $\cup_{j \leq b} M_j$ without violating the b-matching property. Let G' be the remaining graph when M_i was computed. One of the endpoint of e must have degree i in G', let u be that endpoint. After removing M_i the degree of u becomes $i - 1$, and thus u must be matched in M_{i-1}. In general u will be matched in all $M_{j<i}$. Therefore, the degree of u in $\cup_{j \leq b} M_j$ is b, which in turn means the b-matching is maximal with respect to e.

In bipartite graphs a matching incident to all the maximum degree vertices always exists and can be computed in polynomial time. Together with Theorem 6.1, this finishes the proof. □

6.2.1 An almost tight example

While at first sight the analysis of the approximation factor of strongly minimal schedules may seem too pessimistic, it turns out it is almost tight. Consider the following bipartite graph with vertices $u_1, \ldots, u_n$ on one side and vertices $v_1, \ldots, v_n$ on the other side of the bipartition.

There is an edge $(u_i, v_j) \in E$ if and only if $i \leq j$.

It is not difficult to show that the optimal schedule uses matchings

$$O_k = \{(u_i, v_{i+k-1}) \,|\, i \leq n-k+1\}$$

and has cost $\sum_{i=1}^{n} i\,(n-i+1) = \frac{1}{6}n^3 + 3n^2 + 2n$.

Now suppose we run FIND STRONGLY MINIMAL. Initially the maximum degree vertices are u_1 and v_n, and the algorithm finds the matching M_n consisting of $(u_1, v_{\frac{n}{2}})$ and $(u_{\frac{n}{2}+1}, v_n)$. After removing M_n the maximum degree vertices are u_1, u_2, v_{n-1}, and v_n. In general the algorithm may find, for $\frac{n}{2} < k \leq n$,

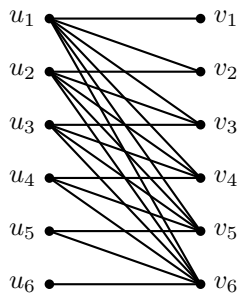
$$M_k = \{(u_i, v_{i+k-\frac{n}{2}-1}) \,|\, i \leq n-k+1\} \cup \{(u_{j-k+\frac{n}{2}+1}, v_j) \,|\, j \geq k\}.$$

After these matchings are removed from the graph we are left with a complete bipartite graph on $u_1, \ldots u_{\frac{n}{2}}$ and $v_{\frac{n}{2}+1}, \ldots v_n$, thus $|M_k| = \frac{n}{2}$ for all $1 \leq k \leq \frac{n}{2}$. Therefore, the cost of this strongly minimal schedule is $\frac{11}{48}n^3 + \frac{5}{8}n^2 + \frac{1}{3}n$. Figure 6.2 shows the input graph along with the optimal solution and the above strongly minimal schedule.

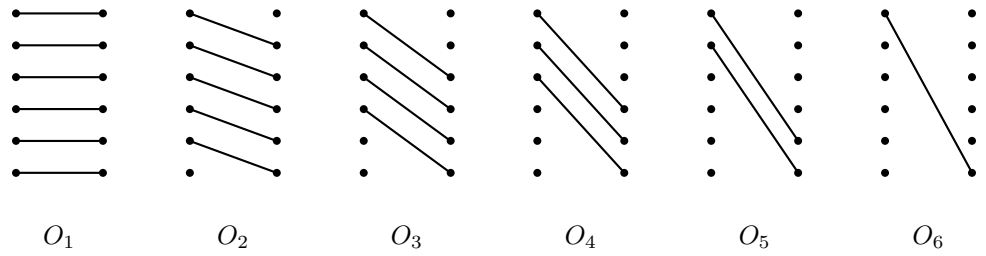
The ratio of the cost of the optimal and strongly minimal solutions approaches 1.375 as $n \to \infty$. Compare this to the approximation guarantee of $\sqrt{2} \approx 1.414$ obtained in Theorem 6.1.

6.2.2 Integrality gap

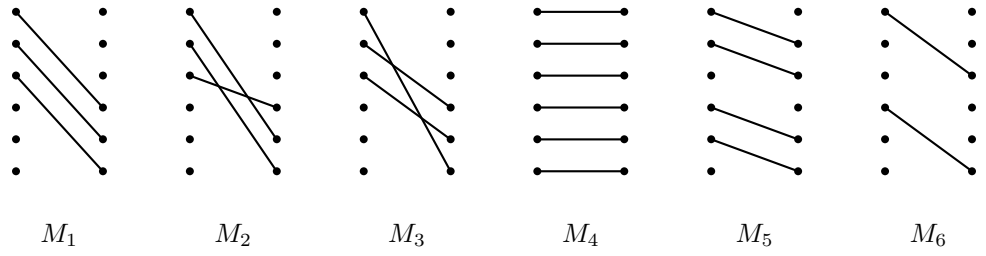
Let us now study the inherent limitations of the lower bounding technique used to prove Theorem 6.1. The lower bound used there can be generalized as follows: For any subset of edges $S(u)$ incident on a vertex u, we know that *any* feasible schedule must spend at least $\frac{|S(u)|(|S(u)|+1)}{2}$ time on these edges. We can charge the cost incurred by this set of edges, a factor $y_{S(u)} \geq 0$. If for every edge e the total charge $\left(\sum_{S(u)\,:\,e \in S(u)} y_{S(u)}\right)$ on e is at most 1, then $\sum_{S(u)} \frac{|S(u)|(|S(u)|+1)}{2}\, y_{S(u)}$ offers a lower bound on the cost an optimal schedule. The best such lower bound corresponds to the optimal solution of the following dual linear program.



a) The instance for $n = 6$.



b) The optimal schedule.



c) A strongly minimal schedule.

Figure 6.2: Construction of the almost tight instance for FIND STRONGLY MINIMAL.

$$\max \sum_{\substack{u \in V \\ S(u) \subseteq E(U)}} \frac{|S(u)|(|S(u)|+1)}{2} y_{S(u)}$$

subject to

$$\sum_{S(u): e \in S(u)} y_{S(u)} \leq 1 \qquad \forall e \in E \qquad (6.1)$$

$$y_{S(u)} \geq 0 \qquad \forall u \in V, S(u) \subseteq E(u)$$

Indeed, the proof of Theorem 6.1 can be viewed as a case of dual-fitting in which constraint (6.1) is violated a $\sqrt{2}$ factor. To determine how good a lower bound the dual offers, we derive the primal LP and study its integrality gap.

Theorem 6.3. *The integrality gap of the LP below is at least $\frac{4}{3}$ in general graphs and at least $\frac{10}{9}$ in bipartite graphs.*

$$\min \sum_{e \in E} C_e$$

subject to

$$\sum_{e \in S} C_e \geq \frac{|S(u)|(|S(u)|+1)}{2} \qquad \forall u \in V, S(u) \subseteq E(u) \qquad (6.2)$$

$$C_e \geq 0 \qquad \forall e \in E$$

Proof. For general graphs, consider a triangle. The optimal solution schedules one edge at the time, and incurs a cost of 6. The LP can schedule all edges at $C_e = 1.5$, with a cost of 4.5. Thus, the integrality gap for this graph is $\frac{4}{3}$.

For the bipartite case (our example is in fact a tree) consider a spider with three legs of length two. The graph is shown on the right in Figure 6.3 along with the edge completion times of an optimal schedule (in black) and of the optimal LP solution (in gray). Optimum schedules three edges in M_1, two in M_2 and one in M_3, with a total cost of 10. On the other hand, the LP solution manages to schedule all edges in two rounds, with a total cost of 9. Thus the integrality

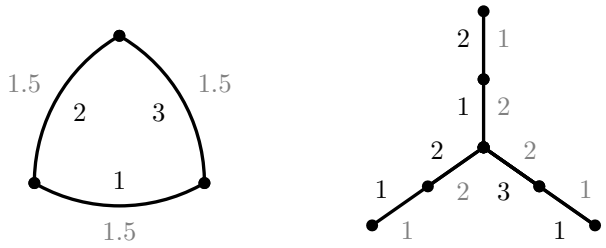


Figure 6.3: Integrality gap examples for general and bipartite instances for $\min \sum_e C_e$. Integral finishing times appear in black; fractional finishing times appear in gray.

gap for bipartite graphs is at least $\frac{10}{9}$. □

6.2.3 Limitations of strongly minimal schedules

We conclude this section with a note on the limitations of strongly minimal schedules. One common generalization of our scheduling problem is to minimize the weighted sum of completion times. In this setting the proof of Theorem 6.1 does not go through as we make crucial use of the fact that the edges have uniform weight.

It would be natural to hope that the following slight modification of FIND STRONGLY MINIMAL would produce good schedules: Instead of finding any matching incident to the maximum degree vertices, find one with minimum weight. Unfortunately, the following bipartite example shows that strongly minimal schedules are just not suited for the weighted case. Take a path of length four and replace each edge with a copy of $K_{t,t}$. The edges in the first and the last $K_{t,t}$ have weight 1, and the ones in the middle have weight 0. The optimal solution schedules the first and the last $K_{t,t}$ in the first t rounds and the remaining edges are scheduled in the next $2t$ rounds, with a total cost of $t^2(t+1)$. On the other hand, any strongly minimal solution can schedule at most t edges with weight 1 per round, thus incurring a total cost of $t^2(2t+1)$. The ratio of the cost of the two solutions approaches 2 as $t \to \infty$.

Chapter 7

Conclusions

Undoubtedly, the primal-dual schema is a powerful technique for designing approximation algorithms. The results in this thesis illustrate, however, that there is still room for new algorithmic ideas within the basic schema, and that sometimes in order to fully realize its potential we must look into more sophisticated uses of it. One aspect of the primal-dual schema that is not yet fully understood is the apparent asymmetry in complexity of the dual update for exact and approximation algorithms. As noted by Williamson [100], the dual update of many exact primal-dual algorithms involve increasing and decreasing variables, while nearly every approximation algorithm only increases the dual variables. Perhaps more sophisticated dual updates could lead to better approximations.

Our results in Chapter 2 suggest that Lagrangian relaxation is a powerful technique for designing approximation algorithms for partial covering problems, even though the black-box approach may not be able to fully realize its potential. Perhaps requiring additional properties from the α-LMP algorithm could lead to better-than-$\frac{4}{3}\alpha$ approximations for partial cover in general. For example, we could ask that the primal solutions around the threshold value λ^* be related to a single dual solution through relaxed complementary slackness conditions. We note that this particular strategy would fail since the lower bound example from Section 2.3 can be adapted to fit this more restrictive framework. More specifically, a dual solution can be constructed such that the solutions returned by the algorithm around λ^* obey complementary slackness conditions (and relaxed primal complementary slackness when $\alpha > 1$). We leave as an open problem to explore this direction of research.

It would be interesting to extend our study on the strengths and limitations of Lagrangian relaxation to other domains. The obvious candidate is the k-median problem. Jain and Vazirani

[65] designed a 2α-approximation for k-median using as a black box an α-LMP approximation for facility location. Later, Jain et al. [66] gave a 2-LMP approximation for facility location. Is the algorithm in [65] optimal in the sense of Theorem 2.1? Can the algorithm in [66] be turned into a 2-approximation for k-median by exploiting structural similarities when combining the two solutions?

The class of totally unimodular matrices is probably the most important subclass of balanced matrices. An open problem in the area of partial cover is to establish good approximations for partial totally unimodular cover (P-TUC). The element-set incidence matrix for the instance used in Section 2.4.4 is both totally balanced and totally unimodular, so the lowerbound on the integrality gap applies for P-TUC as well. Does the upper bound of Theorem 2.3 also hold for P-TUC?

In Chapter 4 we showed the first purely combinatorial algorithms for data migration to minimize the sum of vertex completion times via a more sophisticated procedure for constructing the primal (integral) solution instead of primal complementary slackness. Perhaps this approach could lead to combinatorial algorithms for other scheduling problems.

In Chapter 5 we introduced the notion of adaptive local ratio in the design of a $1+\phi$ approximation for the data migration problem to minimize the sum of vertex completion times. It would interesting to find other examples where optimizing the local ratio leads to better approximations. Another interesting problem is whether there is a combinatorial algorithm for finding an optimal model for ALR.

Bibliography

[1] E. Anderson, J. Hall, J. Hartline, M. Hobbes, A. Karlin, J. Saia, R. Swaminathan, and J. Wilkes. An experimental study of data migration algorithms. In *Proceedings of the 5th Workshop on Algorithm Engineering (WAE)*, 2001.

[2] A. Archer, R. Rajagopalan, and D. B. Shmoys. Lagrangian relaxation for the k-median problem: New insights and continuity properties. In *Proceedings of the 11th Annual European Symposium on Algorithms (ESA)*, 2003.

[3] S. Arora and G. Karakostas. A $2 + \epsilon$ approximation algorithm for the k-MST problem. In *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 754–759, 2000.

[4] E. Balas and M. Padberg. Set partitioning: A survey. *SIAM Review*, 18(4):710760, 1976.

[5] N. Bansal, L. Fleischer, T. Kimbrel, M. Mahdian, B. Schieber, and M. Sviridenko. Further improvements in competitive guarantees for qos buffering. In *Proceedings of the 14th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 196–207, 2004.

[6] A. Bar-Noy, M. Bellare, M. M. Halldórsson, H. Shachnai, and T. Tamir. On chromatic sums and distributed resource allocation. *Information and Computation*, 140:183–202, 1998.

[7] A. Bar-Noy, R. Bar-Yehuda, A. Freund, J. S. Naor, and B. Schieber. A unified approach to approximating resource allocation and scheduling. *Journal of the ACM*, 48(5):1069–1090, 2001.

[8] R. Bar-Yehuda. One for the price of two: A unified approach for approximating covering problems. *Algorithmica*, 27(2):131–144, 2000.

[9] R. Bar-Yehuda. Using homogeneous weights for approximating the partial cover problem. *Journal of Algorithms*, 39(2):137–144, 2001.

[10] R. Bar-Yehuda and S. Even. A linear time approximation algorithm for approximating the weighted vertex cover. *Journal of Algorithms*, 2:198–203, 1981.

[11] R. Bar-Yehuda and S. Even. A local-ratio theorem for approximating the weighted vertex cover problem. *Annals of Discrete Mathematics*, 25:27–46, 1985.

[12] R. Bar-Yehuda and D. Rawitz. On the equivalence between the primal-dual schema and the local ratio technique. *SIAM Journal on Discrete Mathematics*, 19(3):762–797, 2005.

[13] R. Bar-Yehuda, K. Bendel, A. Freund, and D. Rawitz. Local ratio: A unified framework for approximation algorithms. *ACM Computing Surveys*, 36(4):422–463, 2004.

[14] R. Bar-Yehuda, G. Flysher, J. Mestre, and D. Rawitz. Approximation of partial capacitated

vertex cover, 2007. To appear in ESA.

[15] C. Berge. Balanced matrices. *Mathematical Programming*, 2:19–31, 1972.

[16] B. E. Birnbaum and K. J. Goldman. An improved analysis for a greedy remote-clique algorithm using factor-revealing LPs. In *Proceedings of the 9th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, pages 49–60, 2006.

[17] N. Bshouty and L. Burroughs. Massaging a linear programming solution to give a 2-approximation for a generalization of the vertex cover problem. In *Proceedings of the 15th Annual Symposium on the Theoretical Aspects of Computer Science (STACS)*, pages 298–308, 1998.

[18] S. Chakrabarti, C. A. Phillips, A. S. Schulz, D. B. Shmoys, C. Stein, and J. Wein. Improved scheduling problems for minsum criteria. In *Proceedings of the 23rd International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 646–657, 1996.

[19] M. Charikar and S. Guha. Improved combinatorial algorithms for the facility location and k-median problems. In *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 378–388, 1999.

[20] M. Charikar, S. Khuller, D. M. Mount, and G. Narasimhan. Algorithms for facility location problems with outliers. In *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 642–651, 2001.

[21] N. Christofides and S. Korman. A computational survey of methods for the set covering problem. *Management Science*, 21:591–599, 1975.

[22] F. Chudak, M. X. Goemans, and D. S. Hochbaum. A primal-dual interpretation of recent 2-approximation algorithms for the feedback vertex set problem in undirected graphs. *Operations Research Letter*, 22:111–118, 1998.

[23] F. A. Chudak, T. Roughgarden, and D. P. Williamson. Approximate k-MSTs and k-Steiner trees via the primal-dual method and Lagrangean relaxation. In *Proceedings of the 9th Integer Programming and Combinatorial Optimization Conference (IPCO)*, pages 60–70, 2001.

[24] J. Chuzhoy and J. S. Naor. Covering problems with hard capacities. In *Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 481–489, 2002.

[25] V. Chvàtal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4(3):233–235, 1979.

[26] V. Chvátal. *Linear Programming*. Freeman, 1983.

[27] K. L. Clarkson. A modification of the greedy algorithm for vertex cover. *Information Processing Letters*, 16(1):23–25, 1983.

[28] E. G. Coffman, M. R. Garey, D. S. Johnson, and A. S. LaPaugh. Scheduling file transfers. *SIAM Journal on Computing*, 14(3):744–780, 1985.

[29] M. Conforti, G. Cornuéjols, and K. Vuskovic. Balanced matrices. *Discrete Mathematics*, 306 (19-20):2411–2437, 2006.

[30] W. J. Cook, W. H. Cunningham, W. R. Pullyblank, and A. Schrijver. *Combinatorial Optimization*. Wiley, 1998.

[31] I. Dinur and S. Safra. The importance of being biased. In *Proceedings of the 34th annual ACM symposium on Theory of computing (STOC)*, pages 33–42, 2002.

[32] U. Feige. A threshold of ln n for approximating set cover. *Journal of the ACM*, 45(4): 634–652, 1998.

[33] R. Gandhi and J. Mestre. Combinatorial algorithms for data migration to minimize average completion time. In *Proceedings of the 9th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, pages 128–139, 2006.

[34] R. Gandhi, S. Khuller, and A. Srinivasan. Approximation algorithms for partial covering problems. In *Proceedings of the 11th International Colloquium on Automata, Languages, and Programming (ICALP)*, 2001.

[35] R. Gandhi, M. M. Halldórsson, G. Kortsarz, and H. Shachnai. Improved bounds for scheduling conflicting jobs with minsum criteria. In *Proc. of the 2nd Workshop on Approximation and Online Algorithms (WAOA)*, pages 68–82, 2004.

[36] R. Gandhi, S. Khuller, and A. Srinivasan. Approximation algorithms for partial covering problems. *Journal of Algorithms*, 53(1):55–84, 2004.

[37] R. Gandhi, M. M. Halldórsson, G. Kortsarz, and H. Shachnai. Improved results for data migration and openshop scheduling. *ACM Transactions on Algorithms*, 2(1):116–129, 2006.

[38] R. Gandhi, E. Halperin, S. Khuller, G. Kortsarz, and A. Srinivasan. An improved approximation algorithm for vertex cover with hard capacities. *Journal of Computer and System Sciences*, 72(1):16–33, 2006.

[39] R. Gandhi, S. Khuller, S. Parthasarathy, and A. Srinivasan. Dependent rounding and its applications to approximation algorithms. *Journal of the ACM*, 53(3):324–360, 2006.

[40] R. S. Garfinkel and G. L. Nemhauser. *Perspectives on Optimization*, chapter Optimal set covering: A survey, page 164183. Addison-Wesley Publishing Co., 1972.

[41] N. Garg. Saving an epsilon: a 2-approximation for the k-MST problem in graphs. In *Proceedings of the 37th annual ACM symposium on Theory of computing (STOC)*, pages 396–402, 2005.

[42] N. Garg. A 3-approximation for the minimum tree spanning k vertices. In *Proceedings of the 37hd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 302–309, 1996.

[43] N. Garg, V. V. Vazirani, and M. Yannakakis. Primal-dual approximation algorithms for integral flow and multicut in trees. *Algorithmica*, 18(1):3–20, 1997.

[44] D. R. Gaur, T. Ibaraki, and R. Krishnamurti. Constant ratio approximation algorithms for the rectangle stabbing problem and the rectilinear partitioning problem. *Journal of Algorithms*, 43(1):138–152, 2002.

[45] A. Ghouilla-Houri. Charactérisations des matrices totalment unimodulaires. *C. R. Acad. Sc. Paris*, 254:1192–1193, 1962.

[46] M. X. Goemans and D. P. Williamson. *Approximation Algorithms for NP-Hard Problems*, chapter The primal-dual method for approximation algorithms and its application to network design problems, pages 144–191. PWS Publishing Company, 1997.

[47] D. Golovin, V. Nagarajan, and M. Sing. Approximating the k-multicut problem. In *7th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2006.

[48] R. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45:1563–1581, 1966.

[49] S. Guha, R. Hassin, S. Khuller, and E. Or. Capacitated vertex covering. *Journal of Algorithms*, 48(1):257–270, 2003.

[50] J. Hall, J. Hartline, A. Karlin, J. Saia, and J. Wilkes. On algorithms for efficient data migration. In *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 620–629, 2001.

[51] L. Hall, A. S. Schulz, D. B. Shmoys, and J. Wein. Scheduling to minimize average completion time: Off-line and on-line approximation algorithms. *Mathematics of Operations Research*, pages 513–544, 1997.

[52] M. M. Halldórsson, G. Kortsarz, and H. Shachnai. Sum coloring interval graphs and k-claw free graphs with applications for scheduling dependent jobs. *Algorithmica*, 37:187–209, 2003.

[53] E. Halperin. Improved approximation algorithms for the vertex cover problem in graphs and hypergraphs. *SIAM Journal on Computing*, 31(5):1608–1623, 2002.

[54] E. Halperin and A. Srinivasan. Improved approximation algorithms for the partial vertex cover problem. In *Proceedings of the 5th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, pages 161–174, 2002.

[55] R. Hassin and D. Segev. Rounding to an integral program. In *Proceedings of the 4th International Workshop on Efficient and Experimental Algorithms (WEA)*, pages 44–54, 2005.

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

[57] M. Held and R. Karp. The traveling salesman problem and minimum spanning trees: Part II. *Mathematical Programming*, 1:6–25, 1971.

[58] D. S. Hochbaum. Approximation algorithms for the set covering and vertex cover problems. *SIAM Journal on Computing*, 11(3):555–556, 1982.

[59] D. S. Hochbaum. The t-vertex cover problem: Extending the half integrality framework with budget constraints. In *Proceedings of the 1st International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, pages 111–122, 1998.

[60] D. S. Hochbaum, editor. *Approximation Algorithms for NP–hard Problems*. PWS Publishing Company, 1997.

[61] D. S. Hochbaum. *Approximation Algorithms for NP-hard Problems*, chapter Approximating covering and packing problems: set cover, vertex cover, independent set, and related problems, pages 94–143. PWS Publishing Company, 1997.

[62] A. J. Hoffman, A. Kolen, and M. Sakarovitch. Totally-balanced and greedy matrices. *SIAM Journal on Algebraic and Discrete Methods*, 6:721–730, 1985.

[63] H. Hoogeveen, P. Schuurman, and G. Woeginger. Non-approximability results for scheduling problems with minsum criteria. In *Proceedings of the 5th Integer Programming and Combinatorial Optimization Conference (IPCO)*, pages 353–366, 1998.

[64] N. Immorlica, M. Mahdian, and V. S. Mirrokni. Cycle cover with short cycles. In *Proceedings of the 22nd Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 641–653, 2005.

[65] K. Jain and V. V. Vazirani. Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and Lagrangian relaxation. *Journal of the ACM*, 48(2):274–296, 2001.

[66] K. Jain, M. Mahdian, E. Markakis, A. Saberi, and V. V. Vazirani. Greedy facility location algorithms analyzed using dual fitting with factor-revealing LP. *Journal of the ACM*, 50(6): 795–824, 2003.

[67] D. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, 9(3):256–278, 1974.

[68] G. Karakostas. A better approximation ratio for the vertex cover problem. In *Proceedings of the 15th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 1043–1050, 2005.

[69] R. M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[70] S. Khot. On the power of unique 2-prover 1-round games. In *Proceedings of the 34th annual ACM symposium on Theory of computing (STOC)*, pages 767–775, 2002.

[71] S. Khot and O. Regev. Vertex cover might be hard to approximate to within $2-\varepsilon$. In *18th Annual IEEE Conference on Computational Complexity (CCC)*, pages 379–, 2003.

[72] S. Khuller, Y. A. Kim, and Y.-C. J. Wan. Algorithms for data migration with cloning. *SIAM Journal on Computing*, 33(2):448–461, 2004.

[73] S. Khuller, Y.-A. Kim, and A. Malekian. Improved algorithms for data migration. In *Proceedings of the 9th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, pages 164–175, 2006.

[74] Y. Kim. Data migration to minimize the average completion time. *Journal of Algorithms*, 55:42–57, 2005.

[75] A. Kolen. *Location problems on trees and in the rectilinear plane*. PhD thesis, Matematisch Centrum, Amsterdam, 1982.

[76] J. Könemann, O. Parekh, and D. Segev. A unified approach to approximating partial covering problems. In *Proceedings of the 14th Annual European Symposium on Algorithms (ESA)*, pages 468–479, 2006.

[77] J. Könemman and R. Ravi. A matter of degree: Improved approximation algorithms for degree-bounded minimum spanning trees. *SIAM Journal of Computing*, 31:1783–1793, 2002.

[78] H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 41:960–981, 1955.

[79] A. Levin and D. Segev. Partial multicuts in trees. In *3rd Internation Worshop on Approximation and Online Algorithms (WAOA)*, 2005.

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

[81] A. Lubiw. Doubly lexical ordering of matrices. *SIAM Journal on Computing*, 16:854–879, 1987.

[82] D. Marx. Complexity results for minimum sum edge coloring. Unpublished Manuscript, 2004.

[83] N. Megiddo. Combinatorial optimization with rational objective functions. In *Proceedings of the tenth annual ACM symposium on Theory of computing (STOC)*, pages 1–12, 1978.

[84] J. Mestre. A primal-dual approximation algorithm for partial vertex cover: Making educated guesses. In *Proceedings of the 8th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, pages 182–191, 2005.

[85] J. Mestre. Lagrangian relaxation and partial cover: Thinking insider the box. Submitted for publication, 2007.

[86] J. Mestre. Adaptive local ratio. Submitted for publication, 2007.

[87] G. Nemhauser and L. Wolsey. *Integer and Combinatorial Optimization*. Wiley, 1988.

[88] T. Nishizeki and K. Kashiwagi. On the 1.1 edge-coloring of multigraphs. *SIAM Journal on Discrete Mathematics*, 3(3):391–410, 1990.

[89] M. W. Padberg. Covering, packing and knapsack problems. *Annals of Discrete Mathematics*, 4:265287, 1979.

[90] R. Paige and R. E. Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing*, 16:973–989, 1987.

[91] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization*. Dover Publications, Inc., 1998.

[92] O. Parekh and D. Segev. Path hitting in acyclic graphs. In *Proceedings of the 14th Annual European Symposium on Algorithms (ESA)*, pages 564–575, 2006.

[93] M. Queyranne. Structure of a simple scheduling polyhedron. *Mathematical Programming*, 58:263–285, 1993.

[94] M. Queyranne and M. Sviridenko. A $(2 + \epsilon)$-approximation algorithm for generalized preemptive open shop problem with minsum objective. *Journal of Algorithms*, 45:202–212, 2002.

[95] M. Queyranne and M. Sviridenko. Approximation algorithms for shop scheduling problems with minsum objective. *Journal of Scheduling*, 5:287–305, 2002.

[96] P. Sanders and D. Steurer. An asymptotic approximation scheme for multigraph edge coloring. In *Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 897–906, 2005.

[97] A. Schrijver. *Theory of Linear and Integer Programming*. Wiley, 1986.

[98] A. S. Schulz. Scheduling to minimize total weighted completion time: Performance guarantees of lp-based heuristics and lower bounds. In *Proceedings of the 5th Integer Programming and Combinatorial Optimization Conference (IPCO)*, pages 301–315, 1996.

[99] J. P. Spinard. Doubly lexical ordering of dense 0-1 matrices. *Information Processing Letters*, 45:229–235, 1993.

[100] D. P. Williamson. The primal dual method for approximation algorithms. *Mathematical Programming*, 91(3):447–478, 2002.

[101] L. Wolsey. Mixed integer programming formulations for production planning and scheduling problems. In *Invited talk at the 12th International Symposium on Mathematical Programming*, 1985.