Exact, Approximate, and Online Algorithms for Optimization Problems Arising in DVD Assignment

by

James Ross Pearson

A thesis
presented to the University of Waterloo
in fulfilment of the
thesis requirement for the degree of
Master of Mathematics
in
Combinatorics and Optimization

Waterloo, Ontario, Canada, 2009

Abstract

Zip.ca is an online DVD **rental** company that faces two major operational problems: calculation of the assignment of DVDs to customers every thirty minutes throughout the day and purchasing of new inventory in regular intervals.

In this thesis, we model these two problems and develop algorithms to solve them. In doing so, we encounter many theoretical problems that are both applicable to Zip's operations and intrinsically interesting problems independent of the application.

First, we note that the assignment problem facing Zip is inherently in an online setting. With returns of DVDs being processed throughout the day, the dataset is constantly changing. Although the ideal solution would be to wait until the end of the day to make decisions, physical work load capacities prevent this. For this reason we discuss two online problems, online 0-1 budgeted matching and the budgeted Adwords auction. We present a $\frac{1}{2\frac{w_{max}}{w_{min}}}$-competitive algorithm for the online 0-1 budgeted matching problem, and prove that this is the best possible competitive ratio possible for a wide class of algorithms. We also give a $(1-\frac{S+1}{S+e})$-competitive algorithm for the budgeted Adwords auction as the size of the bids and cost get small compared to the budgets, where S is the ratio of the highest and lowest ratios of bids to costs.

We suggest a linear programming approach to solve Zip's assignment problem. We develop an integer program that models the B-matching instance with additional constraints of concern to Zip, and prove that this integer program belongs to a larger class of integer programs that has totally unimodular constraint matrices. Thus, the assignment problem can be solved to optimality every thirty minutes. We additionally create a test environment to check daily performance, and provide real-time implementation results, showing a marked improvement over Zip's old algorithm.

We show that Zip's purchasing problem can be modeled by the matching augmentation problem defined as follows. Given a graph with vertex capacities and costs, edge weights, and budget C, find a purchasing of additional node capacity of cost at most C that admits a B-matching of maximum weight. We give a PTAS for this problem, and then present a special case that is polynomial time solvable that still models Zip's purchasing problem, under the assumption of uniform costs.

We then extend the augmentation idea to matroids and present matroid augmentation, matroid knapsack, and matroid intersection knapsack, three NP-hard problems. We give an FPTAS for matroid knapsack by dynamic programming, PTASes for the other two, and demonstrate applications of these problems.

Acknowledgements

First and foremost, I thank my supervisor Jochen Könemann. Without his direction and insight, none of my work would have been possible. His passion to teach carries over to his role of supervision, making every interaction a valuable experience.

I must also thank the wonderful people at Zip.ca. Their generous hospitality during our visits was much appreciated. I must thank them, in particular, for their patience and understanding during times when my studies took precedence over the project.

I thank Deeparnab Chakrabarty for his interest in my research and valuable discussions about algorithmic ideas included in this thesis.

I am forever in debt to the denizens of DC 3144 and many other graduate students in the department. Their passion for games and ability to bring an air of levity to any situation made for a very enjoyable graduate experience. I feel I must also thank Tom Lehmann, undoubtedly the single most important contributor to our recreational experience.

Finally, I would like to thank my parents and fiancée for their endless support and guidance...not in areas pertaining to this thesis, but in life's much more important matters.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

A deep synergy often exists between academia and industry. Many of the most interesting theoretical problems develop naturally from real applications and many ideas which, at their conception, were purely theoretical, end up being central in solving large-scale real-world problems. This exchange is fruitful for both sides; academia is endowed with a seemingly endless supply of important, relevant, and challenging problems to resolve and industry is able to develop new technologies and solve practical problems.

In this thesis, we consider a DVD rental company Zip.ca and two problems of great importance to its operations: the assignment of DVDs to users and the purchasing of new inventories of DVDs. As we model these problems, we encounter many interesting theoretical problems. While the solutions to some of these problems have direct benefits and applications to Zip, others are interesting at a much more intrinsic level. The intent of this thesis is dual: provide applicable solutions to Zip and pose and suggest solutions to related theoretical problems.

1.1 Introduction to Zip

Zip.ca is an online mail-order DVD rental company. Customers of Zip have available to them many different plans which differ mainly in the number of DVDs that can be possessed at any time. Customers may keep a rented DVD for as long as they like; however, only once a disc is returned can it be replaced by another. Each customer maintains at all times a ranked list of titles that the customer would like to receive.

At any given time, Zip has a set of users who have fewer DVDs than their plans permit. We will say that such users have open slots to fill. Additionally, Zip has inventories of available DVDs in each of its distribution centres (DCs) across the country. Once or twice a day, the DCs receive additional inventory in the form of returns. Over the course of each day, Zip must generate assignments of DVDs to open slots to give to the mailrooms of the DCs for shipping. To obtain an optimal set of assignments over any time period (a day, for example), it would be best to wait until the total inventory for the period was known, and then compute all assignments. However, due to the physical constraints of work capacity in the mailrooms, assignments must be produced in a more continuous manner over the course of the time period. Currently, Zip calculates a new set of assignments every thirty minutes. The size of these assignments is generally much larger than the work capacity of the mailrooms over this time, allowing for better work efficiency by mailroom staff. (We do not want staff to waste time searching for one obscure, rarely-shipped DVD that is stored away when in the same time could process twenty readily available discs.)

The quality of the assignment that Zip produces can be measured by many different metrics. Four main metrics are currently in use:

1. Percentage of slots left open at the end of the day (or equivalently, number of slots filled)

2. Number of slots filled with DVDs of rank one

3. Number of slots filled with DVDs of rank ten or better

4. Number of shipments made from a user's home DC (the DC to which the user returns DVDs and typically closest geographically). Shipments that are not from the home DC are referred to as cross-shipments.

The reason for these metrics being of importance to Zip is fairly obvious. The first three are important for customer satisfaction; customers want to receive the number of DVDs for which they have paid and they want them to be highly-ranked on their lists. The importance of the fourth metric is slightly less obvious. It is important since a DVD shipped from a nearby DC will spend less time in the mail and get to the user sooner, at a cheaper cost. Additionally, the DVD is back in Zip's inventory for reassignment sooner. However, the direct impact of improving this metric is hard to measure. Zip seeks daily to maximize an overall metric, comprised of a combination of these four metrics.

Other issues not captured in this overall metric are also of importance to Zip. One main issue is servicing recently neglected customers. If a user has recently received titles of poor rank or been left with open slots for long periods, the user is more likely to leave Zip. Thus, Zip maintains a service index for each user to measure recent service, and attempts to cater to those with high service indices.

Currently, Zip uses many different algorithms and heuristics to arrive at assignments every thirty minutes. Users are segmented into groups based on service index and assignments are calculated one group at a time, using a type of shortest path algorithm. In this stage, no importance is given to avoiding cross-shipping. Additionally, for the assignments calculated in the first half of the day, only assignments involving DVDs ranked in the top ten are made. This heuristic is done to attempt to avoid making bad assignments early in the day before a large portion of the day's information is available. After the assignments are calculated, a local swap procedure is executed to attempt to decrease cross-shipping. Finally, since there is a physical time period associated with running the algorithm, it is possible that the mailroom has prepared some assignments for shipping while the algorithm was running, creating a discrepancy in the data. Any assignments involving either an open slot or inventory that no longer exists must be removed. This process is referred to as scrubbing.

This series of procedures may produce good assignments every thirty minutes, but it is difficult to derive any sort of performance guarantee. It is difficult to even see what is really happening during the process or which areas have greatest potential for improvement. Our goal in this area is to improve Zip's assignment process.

An additional problem faced by Zip is that of purchasing new inventory. Zip must constantly purchase DVDs, both to keep up to date with new releases and to replace or bolster inventory of older titles. The most logical goal in purchasing new DVDs is to do so in the way that increases the value of the daily metric most significantly, but it can be difficult to predict the impact of purchasing decisions. Our goal in this second area is to provide a mechanism to Zip with which it can make reasonable purchasing decisions.

1.1.1 Applied results

We suggest improvements for both the assignment and purchasing aspects of Zip's operations. On the assignment side, this improvement will come in two forms. First, we modify the metric that Zip uses to evaluate its assignments in such a way that it captures more information. Specifically, it will incorporate the service indices of users and make a distinction between titles of rank eleven and those of rank 500 (neither of which is true of the original metric). Secondly, we create a new algorithm for Zip that, rather then apply a series of heuristics and patches, simply solves one optimization problem to optimality every thirty minutes. Solving a single problem and reaching an optimal solution is appealing for obvious reasons; Zip has a guarantee that, under a certain metric, there does not exist a better solution for the time period under consideration.

Using the new algorithm and targeting the new metric in actual real-time operations at Zip yielded much improved results. The average percentage of slots filled with rank one discs increased from 15.7% to 20.1% and the average percentage of users with open slots at the end of the day decreased from 5.3% to 2.4%, both dramatic increases.

For the purchasing problem, we show that a suitably defined instance of matching augmentation, a problem related to Zip's assignment problem that we present later in this thesis, models finding the budgeted purchasing decision that maximizes the increase of Zip's metric over a given time period. Although we show that this problem is NP-hard, we additionally give a polynomial-time solvable special case that also encodes the purchasing problem, under the assumption that all DVDs have the same price. We conclude by speculating how such an algorithm could be used to make long-term purchasing decisions.

In the following sections, we discuss some basic concepts needed to present and solve the wide range of theoretical problems discussed in this thesis. We also present the theoretical results that we obtain.

1.2 Approximation Algorithms

Given an instance I of a maximization problem M, we are interested in finding an optimal solution for I *efficiently*. An algorithm for M is considered efficient if it returns a solution in time polynomial in the size of the input for any instance. We will consider problems belonging to one or both of the complexity classes P and NP.

The problem M is in P if there exists a polynomial time algorithm that computes the optimal solution given any instance of M. The class NP contains all problems in P, but also many other, likely harder, problems. Specifically, it contains a set of NP-complete problems, problems which are unlikely to also be in P. A problem is called NP-hard if it is, roughly speaking, at least as hard as every problem in NP. For such problems, if we require an algorithm that runs in time polynomial in the input size, it is likely that we will have to settle for *approximately optimal* solutions.

Definition 1.2.1. *Given an instance I of a maximization problem M, let OPT_I be the value of the optimal solution. A polynomial time algorithm A is called a* ***c-approximation*** *algorithm if for every instance I of M, A outputs a solution of value at least OPT_I/c, for constant $c \geq 1$.*

For some NP-hard problems, although we cannot obtain optimal solutions in polynomial time, we can obtain arbitrarily good approximations.

Definition 1.2.2. *A **polynomial time approximation scheme (PTAS)** for an instance I of an NP-hard maximization problem is an algorithm A that, given any $\varepsilon > 0$, returns a solution of value at least $(1-\varepsilon)OPT_I$ in time polynomial in the size of the input.*

Definition 1.2.3. *A **fully polynomial time approximation scheme (FPTAS)** is a PTAS that runs in time polynomial in the size of the input and in $1/\varepsilon$.*

Definition 1.2.4. *A **pseudopolynomial time** algorithm is an algorithm that runs in time polynomial in the* numeric value *of the input (which is exponential in the length of the input). An NP-hard problem is called **weakly NP-hard** if it admits a pseudopolynomial time algorithm. An NP-hard problem that does admit such an algorithm is called **strongly NP-hard**.*

Remark 1.2.5. *If an NP-hard problem has an FPTAS, then it is weakly NP-hard. Conversely, a strongly NP-hard problem does not have an FPTAS.*

We will now consider an example of an NP-hard problem that is relevant to this thesis. The Knapsack problem is defined as follows:

Given items $S = \{1,..,n\}$, weights $w \in \mathbb{Q}^S$, costs $c \in \mathbb{Q}^S_+$, and budget $C \in \mathbb{Q}$, find $S' \subseteq S$ of maximum weight $w(S') := \sum_{e\in S'} w_e$ such that $c(S') \leq C$.

More specifically, the Knapsack problem is weakly NP-hard; it does admit an FPTAS [13].

1.3 Graphs

Many of the problems we consider in this thesis are best modeled by graphs. In this section, we define a graph and the graph concepts needed for the problems we will consider.

Definition 1.3.1. *A **graph** $G := (V,E)$ is a set V of vertices and set E of two-element subsets of V called edges. For $e = \{v_1, v_2\}$, we say e is **incident** with both v_1 and v_2. We denote by $\delta(v)$ the set of all edges incident with v. We call G **bipartite** if there exists a bipartition of V into V_1 and V_2 such that for each edge $e \in E$, e is incident with one vertex from each of V_1 and V_2. (In this case, we will often write $G = (V_1 \cup V_2, E)$)*

Definition 1.3.2. *A **subgraph** $G' = (V', E')$ of $G = (V,E)$ is a graph, where $V' \subseteq V$, $E' \subseteq E$, and for every $e = \{v_i, v_j\} \in E'$, $v_i, v_j \in V'$.*

Definition 1.3.3. *A v_1v_k-**walk** in a graph $G = (V,E)$ is an ordered list*

$$(v_1, e_1, v_2, e_2, .., e_{k-1}, v_k),$$

*where $v_i \in V$ $\forall 1 \leq i \leq k$ and $e_j = \{v_j, v_{j+1}\} \in E$ $\forall 1 \leq j \leq k-1$. If there is a walk between every two vertices in G, we say that G is **connected**. A v_iv_j-**path** is a v_iv_j-walk with no repeated vertices. A **cycle** is a path that begins and ends with the same vertex. A graph with no cycles is called **acyclic**.*

Definition 1.3.4. *A **tree** is a graph $G = (V,E)$ with no cycles where $|V| = |E| - 1$. A **spanning tree** of a graph $G = (V,E)$ is a subgraph $G' = (V,E')$ that is a tree.*

Given weights on the edges of a graph, we can consider the problem of finding a spanning tree of maximum weight. This problem has received much attention and can be solved very efficiently [15]. A harder problem, the *budgeted spanning tree problem*, is presented below:

Given a graph $G = (V,E)$, weights $w \in \mathbb{Q}^E$, costs $c \in \mathbb{Q}^E_+$, and budget $C \in \mathbb{Q}$, find a spanning tree T of maximum weight $w(T)$ such that $c(T) \leq C$.

This problem is shown to be NP-hard in [21].

Definition 1.3.5. *Given a graph $G = (V,E)$ and capacities $B \in \mathbb{Z}^V_+$, a B-**matching** $M \subseteq E$ is a subset of edges such that for every $v \in V$, the number of edges in M incident with v is at most B_v. If the number of edges is equal to B_v, we say that v is **covered** by M; all other vertices are called **uncovered**. If $B_v = 1$ $\forall v \in V$, we will simply call M a **matching**.*

As in the case of spanning trees, we can consider the problem of finding a B-matching of maximum weight. This problem can also be solved efficiently [8]. Once again, we consider the problem of adding a budget constraint to the problem. Consider the *budgeted matching problem*:

Given a graph $G = (V,E)$, weights $w \in \mathbb{Q}^E$, costs $c \in \mathbb{Q}^E_+$, and budget $C \in \mathbb{Q}$, find a matching M of maximum weight $w(M)$ such that $c(M) \leq C$.

This problem is easily seen to be NP-hard (budgeted matching on a set of isolated edges is knapsack), and a PTAS is presented in [1].

Finally, we consider the notion of *directed* graphs.

Definition 1.3.6. *A **directed graph** $D = (V,E)$ is a set V of nodes and set E of two-element* ordered *subsets of V called arcs. For $e = (v_1, v_2)$, we say that e goes from v_1 to v_2. We define $\delta^+(v_1)$ to be the set of arcs going to v_1 and $\delta^-(v_1)$ to be the set of arcs coming from v_1.*

We will consider the *maximum weight st-flow problem* defined on directed graphs, which can be solved efficiently using an algorithm by Ford and Fulkerson [10]:

Given a directed graph $D = (V,E)$, weights $w \in \mathbb{Q}^E$, capacities $u \in \mathbb{Q}^E_+$, a source node s, and a sink node t, find $x \in \mathbb{Q}^E$ such that

- *for all $e \in E, 0 \leq x_e \leq u_e$*
- *for all $v \in V \setminus \{s,t\}$, $x(\delta^+(v)) = x(\delta^-(v))$*
- *$w^T x$ is maximized*

We also present a theorem regarding the existence of integral solutions to the maximum weight *st*-flow problem.

Theorem 1.3.7 ([5]). *If u is integral and there exists a maximum st-flow, then there exists a maximum st-flow that is integral.*

1.3.1 Graph results

Much interest has been paid in recent years to the generalizing of problems by adding budget constraints. Given any maximization problem with a set $\mathscr{F}$ of possible solutions and some budget B, the budgeted form of the maximization problem is to find the solution $S \in \mathscr{F}$ of maximum weight such that the *cost* of S is at most B.

Much of the interest received by budgeted optimization problems can be attributed to the fact that adding a budget constraint often greatly increases the difficulty of solving the problem. Maximum weight matching and maximum weight spanning tree are two such examples. Polynomial time algorithms exist for solving both unbudgeted versions, but as mentioned in the previous section, both budgeted versions are NP-hard.

In 1990, Papadimiriou and Yannakakis gave a framework for multiobjective optimization problems that can be applied to budgeted optimization problems [19]. In this paper, they discuss the notion of a *Pareto curve*. A Pareto curve is a set of solutions where, given any solution in the set, moving to a solution of higher value in any objective requires a decrease in the value of at least one other objective. In some sense, the Pareto curve maps out the set of extreme solutions. An *ε-approximate Pareto curve* is a set of polynomial size in the size of the instance and in $1/\varepsilon$ that approximates a Pareto curve. Papadimitriou and Yannakakis show that if there is a pseudopolynomial algorithm for the exact version of the optimization problem (i.e. find a solution of exactly a given value or determine that none exist), then there is an FPTAS for the construction of an ε-approximate Pareto curve. In the setting of budgeted maximization problems, constructing such a curve using two objectives, weight and cost, it is possible to obtain a solution within a $(1-\varepsilon)$ factor of the optimal solution of cost at most a $(1+\varepsilon)$ factor of the budget. This gives an FPTAS for the problem with slight budget violation. Problems for which this can be done include shortest path, maximum spanning tree [2], and maximum matching [18].

We consider a budgeted generalization of the max-weight B-matching problem. In this version, we are given a cost on each of the *vertices*, and a total budget for the purchase of additional vertex capacity. The objective of this problem is to augment the instance in a way that results in the new instance containing a B-matching of maximum weight. Formally, the *matching augmentation problem* is defined as follows:

Given a graph $G=(V,E)$, weights $w \in \mathbb{Q}^E$, capacities $B \in \mathbb{Z}_+^V$, costs $c \in \mathbb{Q}_+^V$, and a budget $C \in \mathbb{Q}$, find $y \in \mathbb{Z}_+^V$ and $M \subseteq E$ such that:

- $c^T y \leq C$
- *M is a $(B+y)$-matching*
- *M is of maximum weight $w(M)$*

We will also consider a special case of matching augmentation. We call the special case where G is bipartite with $V = V_1 \cup V_2$ and $c_v = \infty \; \forall v \in V_1$, $c_v = 1 \; \forall v \in V_2$ *one-sided bipartite matching augmentation with uniform costs*.

We consider these two problems because they are extremely useful for modeling Zip's purchasing problem. In this thesis, we show that matching augmentation is NP-hard, and proceed to prove the following two theorems:

Theorem 1.3.8. *The matching augmentation problem admits a PTAS*

Theorem 1.3.9. *There exists a polynomial time algorithm for one-sided bipartite matching augmentation with uniform costs.*

We can also consider a version of matching augmentation where, rather than purchasing additional vertex capacity, we purchase 'passes' for the edges. Instead of paying for violations in degree constraints by paying vertex costs, we pay to remove edges from contributing to the vertex loads. We can define the *edge-cost matching augmentation* problem as follows:

Given a graph $G=(V,E)$, weights $w \in \mathbb{Q}^E$, capacities $B \in \mathbb{Z}^V$, costs $c \in \mathbb{Q}_+^E$, and a budget $C \in \mathbb{Q}$, find $Y \subseteq M \subseteq E$ such that:

- $c^T Y \leq C$
- $M \setminus Y$ *is a B-matching*
- M *is of maximum weight* $w(M)$

In the thesis, we prove the following theorem:

Theorem 1.3.10. *The edge-cost matching augmentation problem on bipartite graphs admits a PTAS.*

1.4 Online algorithms

To this point, we have only discussed problems where the entire instance is given to us before our algorithms must make any decisions. In many applications, this is far from reality; information comes in pieces over a period of time. Rarely in these situations do we have the luxury of being able to wait until the entire instance becomes known. The assignment problem facing Zip is one such example. Zip must compute many assignments each day, most of which are computed before all DVD availability for the day is known.

In this section, we present a model for online problems. All of the problems we consider in this thesis will be modeled by bipartite graphs, so we will extend bipartite graphs into this online setting.

We will consider a graph $G = (U \cup V, E)$, but at the onset of the algorithm, neither V nor E is known to us. Furthermore, we will assume that the vertices of V are ordered; without loss of generality, let the ordering be $(v_1, v_2, .., v_n)$. One at a time, starting with v_1, vertex v_i and edges $E \cap \delta(v_i)$ will be presented, along with any information associated with the edges (weights, for example). At this time, our algorithm must irrevocably decide to include at most one $e \in E \cap \delta(v_i)$ in the solution. Once this decision is made, all edges not chosen are discarded and chosen edges can never be removed from the solution.

Online problems are typically hard to solve. Since decisions are being made without knowing the entire instance, it is almost impossible to design an algorithm that can guarantee an optimal solution for any instance of a problem. As in the case of *NP*-hard problems, we will settle for approximately optimal solutions.

Definition 1.4.1. *Given an instance I of an online maximization problem M, let OPT_I be the value of the optimal (offline) solution. An online algorithm A for M is called **c-competitive** if for every instance I of M, A returns a solution of value at least OPT_I/c.*

We now give two examples of online problems that will be discussed in this thesis. The first problem we consider is *online matching*. This problem is defined as follows:

Given a bipartite graph $G = (U \cup V, E)$, where V, E arrive online, find a matching M of maximum size $|M|$.

In 1990, Karp, Vazirani, and Vazirani gave an extremely simple $1/2$-competitive algorithm for online matching that simply picks any edge any time one is available [14]. They show that this is the best competitive ratio possible for a deterministic algorithm. However, in the same paper, they give a randomized algorithm with competitive ratio of $1-1/e$ (where the competitive ratio of a randomized algorithm is defined as the maximum over all instances of the ratio of the optimal value to the expected value of the solution returned by the algorithm). Their algorithm fixes a random permutation of the known side of the bipartition and makes all decisions according to the ranking in this permutation. The authors show that no randomized algorithm can achieve a better competitive ratio. Since this paper, related problems have received much attention. One such problem is the Adwords auction.

The Adwords auction is a problem that models the application of the selling of advertising space in search engines. Every time an internet user enters a keyword, a set of buyers bid for advertising spots available in the search results page. Each buyer has a daily budget, and as the auctioneer, we obtain profit equal to the minimum of each buyer's budget and the sum of all of the bids made by the buyer in auctions the buyer won. This problem is clearly best modeled as being online as we have no information pertaining to which search queries will be entered.

We can model the Adwords auction as an online graph problem, where U is known and V, E arrive online:

Let U be the set of buyers, V be the set of advertising slots (often called products), and E be the set $U \times V$. Let $n = |U|$ and $m = |V|$. Given bids $b \in \mathbb{R}^E$ and budgets $B \in \mathbb{R}^U$, find $M \subseteq E$ such that:

- *for each $j \in V$, $\sum_{e \in M \cap \delta(j)} b_e \leq 1$*
- *$\sum_i \min\{B_i, \sum_{e \in M \cap \delta(i)} b_e\}$ is maximized*

In 2007, Mehta, Saberi, Vazirani, and Vazirani gave a deterministic algorithm for this problem that achieves a $1-1/e$ competitive ratio when the size of the bids is small compared to the budgets[17]. In the same paper, they show that no randomized algorithm can do better. In 2008, Buchbinder gave a primal-dual algorithm that achieves the same ratio, an algorithm discussed in detail later in

Section 2.2 [4]. Birnbaum and Mathieu gave a survey of online matching, the Adwords auction, and several related problems, and provided a simple proof of the original result in Karp et al.[3].

1.4.1 Online results

In this thesis, we consider generalizing both online matching and the Adwords auction by adding budget constraints. As mentioned in Section 1.3.1, adding budget constraints to maximization problems has been of much interest in recent years. However, to the author's knowledge, little work has been done regarding budgeted online problems. The Adwords auction has a budget for each buyer, but these are not explicit constraints. As the size of the bids get small compared to the budgets, it is not hard to make the constraints explicit without sacrificing the competitive ratio.

In addition to adding budgets to the problems, we also consider the more general weighted version of online matching:

Given a graph $G = (U \cup V, E)$, weights $w \in \mathbb{R}^E$, costs $c \in \{0,1\}_+^E$ and budget $C \in \mathbb{R}$, find a matching M of maximum weight $w(M)$ with cost $c(M) \leq C$.

The budgeted Adwords auction is defined as follows:

Let U be the set of buyers, V be the set of products, and E be the set $U \times V$. Let $n = |U|$ and $m = |V|$. Given bids $b \in \mathbb{R}^E$, budgets $B \in \mathbb{R}^U$, costs $c \in \mathbb{R}_+^E$, and cost budget $C \in \mathbb{R}$ find $M \subseteq E$ such that:

- *for each $j \in V$, $\sum_{e \in M \cap \delta(j)} b_e \leq 1$*
- *$c(M) \leq C$*
- *$\sum_i \min\{B_i, \sum_{e \in M \cap \delta(i)} b_e\}$ is maximized*

In this thesis, we prove the following two results.

Theorem 1.4.2. *There exists a deterministic polynomial time algorithm for the online 0-1 budgeted matching problem that returns a solution of weight within a $\frac{1}{2\frac{w_{max}}{w_{min}}}$-factor of the optimal solution. Furthermore, this is the optimal ratio for a wide class of algorithms.*

Notice that for the unweighted case, this algorithm is $1/2$-competitive, matching the bound given in [14] for deterministic algorithms, despite the addition of a budget constraint.

Theorem 1.4.3. *There exists a polynomial time algorithm for the budgeted Adwords auction that returns a solution of weight within a* $(1-\frac{S+1}{S+e})$*-factor of the optimal solution, as the size of the bids and costs get small compared to the budgets, where* S *is the ratio of the highest and lowest ratios of* b_e *to* c_e*.*

1.5 Linear Programming

A linear program (P) can be expressed as follows:

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

In (P) above, A is an $m \times n$ matrix and c, b are vectors.

Definition 1.5.1. *A solution* x *is called* ***feasible*** *for (P) if* $Ax \leq b$ *and* $x \geq 0$*.*

Definition 1.5.2. *A feasible solution* x^* *is called* ***optimal*** *if* $c^T x^* = \max\{c^T x \text{ s.t. } Ax \leq b,\ x \geq 0\}$*.*

Each linear program has associated with it a *dual* linear program. For a linear program (P) of the form above, the dual (D) is expressed as follows:

$$\begin{aligned} \min \quad & b^T y \\ \text{s.t.} \quad & A^T y \geq c \\ & y \geq 0 \end{aligned} \tag{D}$$

The following theorem gives a relationship between feasible solutions to (P) and (D).

Theorem 1.5.3. *(*Weak duality*) If* x *is feasible for (P) and* y *is feasible for (D), then* $c^T x \leq b^T y$*.*

The weak duality theorem states that any feasible solution to (D) is an upper bound for any feasible solution to (P). This fact can be very useful in designing approximation algorithms, as shown by the following corollary:

Corollary 1.5.4. *If* x *is feasible for (P),* y *is feasible for (D), and* $c^T x \geq \frac{b^T y}{\alpha}$*, then* $c^T x \geq \frac{c^T x^*}{\alpha}$*.*

In many applications, it makes no sense to consider solutions x for (P) that are not integral. Thus, we can define an *integer program* that requires that certain variables take on only integral variables. Consider the integer program (IP) below:

$$\begin{aligned} \max \quad & c^T x \\ \text{s.t.} \quad & Ax \le b \\ & x \in \mathbb{Z}_+^n \end{aligned} \tag{IP}$$

Although these integer programs can lead to more realistic models, they are often hard to solve. While linear programs can be solved in polynomial time (using the Ellipsoid Method), integer programs are in general NP-hard to solve [6]. Thus, we often approximate integer programs by relaxing the integrality constraints. We call (P) the *linear programming relaxation* of (IP).

Now consider (P2) of the following form:

$$\begin{aligned} \max \quad & c^T x \\ \text{s.t.} \quad & Ax = b \\ & x \ge 0 \end{aligned} \tag{P2}$$

Where A is an $m \times n$ matrix as before, $m \le n$, and $rank(A) = m$.

Definition 1.5.5. *A **basis** of (P2) is a set $B \subseteq \{x_1, .., x_n\}$ with $|B| = m$ such that the columns A_B of A corresponding B are linearly independent. The solution $x = A^{-1}b$ is called the **basic feasible solution corresponding to** B.*

Remark 1.5.6. *Any linear program can be expressed in the form of (P2), so the notion of bases and basic feasible solutions is not limited to any specific form of linear program.*

Definition 1.5.7. *Let x be feasible for (P). We say x is an **extreme point** if there does not exist any y such that both $x + y$ and $x - y$ are feasible for (P).*

Fact 1.5.8. *x is an extreme point if and only if it is a basic feasible solution for some basis*

Definition 1.5.9. *We call two extreme points of a linear program **adjacent** if their bases share all but one variable in common.*

1.5.1 Lagrangian relaxation

We conclude our background work on linear programs with a brief discussion of Lagrangian relaxations. Consider the integer program (IP') below obtained by adding an additional constraint to (IP):

$$\begin{aligned} \max \quad & c^T x && \text{(IP')} \\ \text{s.t.} \quad & Ax \leq b \\ & d^T x \leq B && (1) \\ & x \in \mathbb{Z}^n_+ \end{aligned}$$

We have discussed many problems where the addition of a budget constraint such as (1) makes the problems hard to solve. The method of Lagrangian relaxation allows us to, in some ways, 'remove' this troublesome constraint. We consider taking (1) and 'lifting' it into the objective function. Consider the integer program LR(λ):

$$\begin{aligned} \max \quad & c^T x + \lambda(B - d^T x) && \text{(LR(}\lambda\text{))} \\ \text{s.t.} \quad & Ax \leq b \\ & x \in \mathbb{Z}^n_+ \end{aligned}$$

We call LR(λ) the *Lagrangian relaxation* of (IP') for some $\lambda \geq 0$. Notice that any solution feasible for (IP') is feasible for LR(λ), for any λ.

Definition 1.5.10. *The* ***Lagrangian weight*** $w_\lambda(e)$ *of a variable* x_e *is its objective coefficient in the Lagrangian relaxation.*

In the case of LR(λ), $w_\lambda(e) = c_e - \lambda d_e$.

Although there exists considerable theory regarding Lagrangian relaxation, we will require only a few basic properties for the purpose of this thesis.

Theorem 1.5.11. *Let* x^* *be optimal for (IP') and* x^*_λ *be optimal for LR(λ) with* $\lambda \geq 0$. *Then* $c^T x^* \leq c^T x^*_\lambda + \lambda(B - d^T x^*_\lambda)$.

Proof. Consider any x feasible for (IP'). Since x satisfies $d^T x \leq B$, $c^T x \leq c^T x + \lambda(B - d^T x)$. The result follows from the observation that x is feasible for LR(λ). □

Corollary 1.5.12. *If* x^*_λ *is optimal for LR(λ) and* $d^T x^*_\lambda = B$, *then* x^*_λ *is optimal for (IP').*

1.5.2 An integer program for Zip

The most significant applied result of this thesis is the development of a new algorithm to solve instances of Zip's assignment problem every thirty minutes. The algorithm that we created is conceptually extremely simple.

Zip's assignment problem can be formulated as an integer program.

The new assignment algorithm consists of solving one integer program to optimality. Although we have stated that in general it is hard to solve integer programs, a result later in the thesis shows that the linear program relaxation of this integer program is an exact formulation, and can thus be solved to obtain integral solutions.

1.6 Matroids

Let S be a ground set of elements and $\mathscr{I}$ be a set of subsets of S. We will call any $I \in \mathscr{I}$ *independent* (and all other subsets *dependent*).

Definition 1.6.1. *We call $\mathscr{M} = (S, \mathscr{I})$ a **matroid** if the following three properties hold:*

(M0) $\emptyset \in \mathscr{I}$

(M1) If $I' \subseteq I \subset S$ and $I \in \mathscr{I}$, then $I' \in \mathscr{I}$

(M2) If $I, J \in \mathscr{I}$ and $|I| < |J|$, then there exists $e \in J \setminus I$ such that $I \cup \{e\} \in \mathscr{I}$

We now define several concepts regarding matroids.

Definition 1.6.2. *Given a matroid $\mathscr{M} = (S, \mathscr{I})$ and $A \subseteq S$:*

- *A **basis** of A is an inclusion-wise maximal independent subset $B \subseteq A$.*
- *The **rank** of A is $r(A) := \max\{|B| : B \text{ a basis of } A\}$*
- *A **circuit** of M is a minimally dependent subset $C \subseteq S$*

Alternatively, we can define a matroid as follows:

Definition 1.6.3. *$\mathscr{M} = (S, \mathscr{I})$ is a **matroid** if (M0), (M1), and the following additional property hold:*

(M3) For all $A \subseteq S$, every basis of A has the same cardinality

We now give some useful properties of matroids.

Lemma 1.6.4. *Given a matroid $\mathscr{M} = (S, \mathscr{I})$:*

1. *(*Deletion*) For every $S_0 \subseteq S$, $\mathscr{M} - S_0 := (S', \mathscr{I}')$ is a matroid, where $S' := S \setminus S_0$ and $\mathscr{I}' := \{I \in \mathscr{I} : I \cap S_0 = \emptyset\}$.*
2. *(*Contraction*) For every $I \in \mathscr{I}$, $\mathscr{M}/I := (S', \mathscr{I}')$ is a matroid, where $S' := S \setminus I$ and $\mathscr{I}' := \{S_0 \subseteq S \setminus I : S_0 \cup I \in \mathscr{I}\}$.*
3. *(*Truncation*) For every $q \in \mathbb{N}$, $\mathscr{M}^q := (S, \mathscr{I}^q)$ is a matroid, where $\mathscr{I}^q := \{I \in \mathscr{I} : |I| \leq q\}$.*
4. *(*Extension*) For every D, $D \cap S = \emptyset$, $\mathscr{M} + D := (S', \mathscr{I}')$ is a matroid, where $S' := S \cup D$ and $\mathscr{I} := \{S_0 \subseteq S \cup D : S_0 \cap S \in \mathscr{I}\}$.*
5. *(*Union*) For any other matroid $\mathscr{M}_2 = (S_2, \mathscr{I}_2)$, $\mathscr{M}' := (S', \mathscr{I}')$ is a matroid, where $S' := S \cup S_2$ and $\mathscr{I}' := \{I_1 \cup I_2 : I_1 \in \mathscr{I}, I_2 \in \mathscr{I}_2\}$.*

An additional property of matroids that will extremely useful in this thesis is that bases can be found by linear programming. Specifically, consider the max-weight independent set problem:

Given a matroid $\mathscr{M} = (S, \mathscr{I})$ and weights $w \in \mathbb{Q}^S$, find an independent set of maximum weight.

This problem can be formulated as the following linear program, where there is a variable x_e for each $e \in S$:

$$\begin{aligned} \max \quad & w^T x \\ \text{s.t.} \quad & x(A) \leq r(A) \; \forall A \subseteq S \\ & x \geq 0 \end{aligned}$$

Fact 1.6.5 ([20]). *The above linear program always has an integral optimal solution.*

Before we can use the above fact, we must discuss matroid models. Note that a matroid can contain exponentially many independent sets. Thus, determining if a given set is independent is potentially a hard problem. If testing for independence is hard, then certainly the max-weight independent set problem is also hard, as are any problems which require as solutions independent sets. Thus, it is usually assumed when working with matroids that we are provided with an independence oracle. This makes matroid problems tractable and interesting to consider. For the remainder of this thesis, we will assume access to an such an independence oracle.

Corollary 1.6.6. *The max-weight independent set problem can be solved in polynomial time.*

Matroids are very useful in modeling certain problems. One such matroid is the *graphic matroid*, which can be used to model many graph problems. We now define the graphic matroid.

Definition 1.6.7. *Given $G = (V,E)$, the **graphic matroid** of G is $\mathscr{M} = (E,\mathscr{I})$, where $\mathscr{I} = \{E' \subseteq E : E'$ contains no cycles$\}$.*

Remark 1.6.8. *In the graphic matroid for $G = (V,E)$, where G is connected, the set of bases of E is the set of spanning trees of G.*

One other matroid that will be used in the thesis is the uniform matroid.

Definition 1.6.9. *Given a ground set S and an integer C, the **uniform matroid** of S of size C is $\mathscr{M} = (S,\mathscr{I})$, where $\mathscr{I} = \{S' \subseteq S : |S'| \leq C\}$.*

We will now consider some well-studied matroid problems that will be useful in this thesis. First is the matroid intersection problem. This problem is similar to the maximum weight independent set problem, except we now have two matroids, and must find a set independent in both. Formally, the matroid intersection is defined as follows:

Given matroids $\mathscr{M}_1 = (S,\mathscr{I}_1)$ and $\mathscr{M}_2 = (S,\mathscr{I}_2)$ and weights $w \in \mathbb{Q}^S$, find $I \in \mathscr{I}_1 \cap \mathscr{I}_2$ of maximum weight $w(I)$.

As in the case of max-weight independent set, there exists a polynomial time algorithm to solve matroid intersection [9].

As in earlier sections, we also consider budgeted versions of these two problems:

- *Budgeted independent set*: Given a matroid $\mathscr{M} = (S,\mathscr{I})$, weights $w \in \mathbb{Q}^S$, costs $c \in \mathbb{Q}^S_+$, and a budget $C \in \mathbb{Q}$, find $I \in \mathscr{I}$ of maximum weight such that $c(I) \leq C$.
- *Budgeted matroid intersection*: Given matroids $\mathscr{M}_1 = (S,\mathscr{I}_1)$ and $\mathscr{M}_2 = (S,\mathscr{I}_2)$, weights $w \in \mathbb{Q}^S$, costs $c \in \mathbb{Q}^S_+$, and a budget $C \in \mathbb{Q}$, find $I \in \mathscr{I}_1 \cap \mathscr{I}_2$ of maximum weight $w(I)$ such that $c(I) \leq C$.

Budgeted independent set is *NP*-hard, as the budgeted spanning tree problem on G can be solved by finding a budgeted independent set of the graphic matroid corresponding to G. Berger et al. give a PTAS for the budgeted matroid intersection problem in [1]. These two results imply that both budgeted problems are *NP*-hard and admit a PTAS, but the status of the existence of an FPTAS is not known for either.

1.6.1 Matroid results

As mentioned in the previous section, matroids are quite powerful for modeling problems. Graphic matroids model spanning trees in graphs. Matroids quite often generalize existing combinatorial objects and can be used to encode many problems. This is particularly true of matroid intersection. In this thesis, we define several matroid problems with the purpose of encoding some of the ideas behind previous problems with matroids. First, we consider the *matroid augmentation* problem, a problem that in many ways is the matroid equivalent of the edge-cost matching augmentation problem. Rather than being given a graph and buying 'passes' for some edges, we are given an independent set of a matroid, and are buying additional elements such that the resulting set has as large a maximum weight independent set as possible. Formally, the matroid augmentation is defined as follows:

Given a matroid $\mathscr{M} = (S, \mathscr{I})$, $A \subseteq S$, weights $w \in \mathbb{Q}^S$, costs $c \in \mathbb{Q}^S_+$, and budget $C \in \mathbb{Q}$, find $A' \subseteq S$ such that $c(A') \leq C$ and the weight of the max-weight independent set $I \subseteq (A \cup A')$ is maximized.

In this thesis, we prove the following result.

Theorem 1.6.10. *The matroid augmentation problem admits a PTAS.*

A problem related to matroid augmentation is *matroid knapsack*. In the matroid augmentation problem, we are given $A \subseteq S$ as our starting set, much as we are given starting capacities in the matching augmentation problem. In the matroid knapsack problem, we are free to select any $A \in \mathscr{I}$ as our starting set. We then must purchase additional elements to achieve a set of maximum weight. The name of this problem arises from these two aspects; given our choice of a basis of the matroid, the optimal solution is obtained by solving an instance of knapsack on the remaining elements. The matroid knapsack problem is as follows:

Given a matroid $\mathscr{M} = (S, \mathscr{I})$, weights $w \in \mathbb{Q}^S$, costs $c \in \mathbb{Q}^S_+$, and a budget $C \in \mathbb{Q}$, find $A' \subseteq A \subseteq S$ such that $c(A') \leq C$, $A \setminus A' \in \mathscr{I}$, and $w(A)$ is maximized.

We can think of this problem as trying to find a set that *almost* independent; we must pay for any elements that cause a violation of independence. We prove the following result about matroid knapsack.

Theorem 1.6.11. *The matroid knapsack problem admits an FPTAS.*

The FPTAS mentioned in Theorem 1.6.11 is obtained by scaling the instance and applying a dynamic programming algorithm.

Finally, we generalize the matroid knapsack problem to the setting of matroid intersection. As mentioned earlier, matroid intersection has many applications. The problem we present here, a generalization of matroid intersection and of matroid knapsack, has even greater modeling power. The *matroid intersection knapsack* problem is as follows:

Given matroids $\mathscr{M}_1 = (S, \mathscr{I}_1)$ and $\mathscr{M}_2 = (S, \mathscr{I}_2)$, weights $w \in \mathbb{Q}^S$, costs $c \in \mathbb{Q}^S_+$, and a budget $C \in \mathbb{Q}$, find $A' \subseteq A \subseteq S$ such that $c(A') \leq C$, $A \setminus A' \in \mathscr{I}_1 \cap \mathscr{I}_2$, and $w(A)$ is maximized.

We obtain the following result:

Theorem 1.6.12. *The matroid intersection knapsack problem admits a PTAS.*

Furthermore, we show the applicability of the matroid intersection knapsack problem by modeling the bipartite version of edge-cost matching augmentation as a matroid intersection knapsack problem and obtaining a PTAS.

1.7 Miscellaneous

In this section, we introduce some additional concepts that will be needed in this thesis.

Definition 1.7.1. *Given a set S, a* ***partial order*** *on S is a binary relation '$\leq$' such that for all $a, b, c \in S$:*

- *(*reflexivity*) $a \leq a$*
- *(*antisymmetry*) if $a \leq b$ and $b \leq a$ then $a = b$*
- *(*transitivity*) if $a \leq b$ and $b \leq c$ then $a \leq c$*

We call S along with the partial order a ***partially ordered set*** *(*or ***poset****).*

Remark 1.7.2. *In all of the applications in this thesis, we will use the relation 'subset'. In other words, $A \leq B$ if $A \subseteq B$*

We require a few definitions regarding posets.

Definition 1.7.3. *A* ***chain*** *in a poset S is an ordered list $(s_1, .., s_t) \subseteq S$ such that $\forall\ 1 \leq j \leq k \leq t, s_j \leq s_k$.*

Definition 1.7.4. *An* ***antichain*** *in a poset S is a set $\{s_1, .., s_t\} \subseteq S$ such that $\forall\ 1 \leq j, k \leq t, s_j \not\leq s_k$ and $s_k \not\leq s_j$.*

Finally, we can present a theorem relating the sizes of chains and antichains in any poset.

Theorem 1.7.5 (Dilworth's Theorem [7])**.** *In a poset S, the maximum size of an antichain equals the minimum number of chains in any partition of S into chains.*

We conclude this section with the topic of total unimodularity.

Definition 1.7.6. *A matrix A is called **totally unimodular** if every square submatrix of A has determinant 0 or ± 1.*

Total unimodularity is an important concept in linear programming for several reasons. First, the linear programming formulation of many simple problems has a totally unimodular constraint matrix. Second, we know how to solve such problems efficiently, as displayed by the following theorem.

Theorem 1.7.7 (Hoffman and Kruskal's Theorem [12])**.** *Let A be an integral constraint matrix of the linear program (P). Then A is totally unimodular if and only if for each integral b, (P) has integral optimal solutions.*

Remark 1.7.8. *If we can model a problem arising from (P) with integral constraint matrix A as a maximum weight st-flow problem with integral capacities b, then Theorems 1.3.7 and 1.7.7 imply that A is totally unimodular.*

Definition 1.7.9. *Given a graph $G = (V,E)$, the **vertex-edge incidence matrix** of G is a $|V| \times |E|$ matrix with a one in the (i, j) position if vertex v_i is incident with edge e_j, and zero otherwise.*

Fact 1.7.10 ([12])**.** *A graph $G = (V,E)$ is bipartite if and only if the vertex-edge incidence matrix of G is totally unimodular*

1.7.1 Total unimodularity results

A final result included in this thesis relates to showing that a certain class of linear programs have totally unimodular constraint matrices. Of particular significance is that the integer program we formulated to solve Zip's assignment problem belongs to this class. This implies the following corollary.

Corollary 1.7.11. *The linear program relaxation for the integer programming formulation used in Zip's assignment algorithm is exact, and thus has integral optimal solutions.*

This implies that we can solve the integer program used in Zip's assignment algorithm.

Problem	Approximation factor of algorithm
Budgeted online matching	$\frac{1}{2^{\frac{w_{max}}{w_{min}}}}$-factor
Budgeted Adwords auction	$(1-\frac{S+1}{S+e})$-factor*
Matching augmentation	PTAS
$\hookrightarrow$ 1-sided bip. uniform case	polytime solvable
Matroid augmentation	PTAS
Matroid knapsack	FPTAS
Matroid intersection knapsack	PTAS

Table 1.1: Summary of theoretical results, where * denotes an asymptotic result as bids, costs get small compared to budgets

1.8 Summary of contributions and results

In this thesis, we have two areas of contribution. On the applied side, we give an exact linear programming formulation of Zip's assignment problem, showing that the assignment problem can be solved efficiently. We also show that an instance of matching augmentation can be used to model Zip's purchasing problem. Since this problem is NP-hard, we show that a polynomial time solvable special case still has the ability to model Zip's purchasing problem, if we assume that all DVDs have equal cost.

We summarize the theoretical algorithmic results contained in this thesis in Table 1.1.

1.9 Thesis outline

This thesis begins in Chapter 2 by providing in detail algorithms for budgeted matroid intersection and the Adwords auction, both of which will be modified later in the thesis to solve related problems.

In Chapter 3, we discuss all of the applied aspects of this thesis, and in doing so, attempt to motivate all of the subsequent theoretical work. Results of actual algorithm implementation are discussed. In Chapter 4 we consider the online nature of Zip's operations, pose two problems that have reasonable modeling power, and provide online algorithms for each. In Chapter 5, the problem of solving the integer program modeling Zip's assignment process is resolved by giving a total unimodularity result of a class of linear programs.

Chapter 6 contains the bulk of our theoretical results. Inspired by Zip's purchasing problem, the matching augmentation problem is discussed, along with a tractable special case particularly relevant to Zip. We then transfer the idea behind the augmentation problem to the setting of matroids, where we define and give approximation algorithms for several problems.

Chapter 2

Algorithms

We will present two algorithms in this chapter that we will modify in the thesis to solve related problems.

2.1 Algorithm for budgeted matroid intersection

The following PTAS for budgeted matroid intersection is presented in [1].

Let the two matroids in the problem be $\mathscr{M}_1 = (S, \mathscr{I}_1)$ and $\mathscr{M}_2 = (S, \mathscr{I}_2)$. The budgeted matroid intersection problem can be expressed as the following integer program:

$$\begin{array}{rll} \max & w^T x & \qquad\qquad\qquad (IP_{BMI}) \\ \text{s.t.} & x(T) \le r_1(T) \quad \forall T \subseteq S & \\ & x(T) \le r_2(T) \quad \forall T \subseteq S & \\ & c^T x \le C & \\ & x \in \mathbb{Z}_+^S & \end{array}$$

First, we form the Lagrangian relaxation LR(λ) of (IP_{BMI}) by lifting the budget constraint into the objective function:

$$\begin{array}{rll} \max & w^T x + \lambda (C - c^T x) & \qquad\qquad\qquad (\text{LR}(\lambda)) \\ \text{s.t.} & x(T) \le r_1(T) \quad \forall T \subseteq S & \\ & x(T) \le r_2(T) \quad \forall T \subseteq S & \\ & x \in \mathbb{Z}_+^S & \end{array}$$

Denote by λ^* the value of λ that minimizes the optimal value of the resulting linear program. We would like to find λ^* for LR(λ). This can be done in polynomial time using Megiddo's parametric search technique whenever LR(λ) can be solved in polynomial time [22], which is the case here. The idea behind Megiddo's parametric search technique is that we simultaneously simulate the algorithm that solves LR(λ^*) and compute λ^*. During the technique, the Lagrangian weight $w_\lambda(e) = w_e - \lambda c_e$ of each $e \in S$ will be represented by a function of the form $a + \lambda b$. Whenever the algorithm needs to compare two such weights, it computes the value of λ that makes the two equal, and then determines if this value is greater than or less than λ^*; We can determine which is larger by solving one additional Lagrangian relaxation for a fixed value of λ. By the end of the procedure, the output will be used to determine λ^*. The reader should see [16] for a detailed discussion of the technique.

Once we have found λ^*, we compute two solutions $X_1, X_2 \in \mathscr{I}_1 \cap \mathscr{I}_2$ such that $c(X_1) \le C \le c(X_2)$. This is done by solving LR($\lambda^* + \varepsilon$) and LR($\lambda^* - \varepsilon$) for sufficiently small $\varepsilon > 0$. Intuitively, this works because by increasing λ^* slightly, we increase the contribution of the lifted budget term, giving the optimal solution incentive to be of smaller cost.

Note that for $i \in \{1,2\}$, we have that

$$w_\lambda(X_i) + \lambda C \ge w_\lambda(X^*) + \lambda C \ge w_\lambda(X^*) + \lambda c(X^*) = OPT, \tag{2.1.1}$$

where X^* is an optimal solution to the budgeted matching instance.

Notice that neither X_1 nor X_2 will contain an element of negative Lagrangian weight. Thus, for any $e \in X_1 \cup X_2$, $w_\lambda(e) = w_e - \lambda c_e \ge 0$. Thus,

$$w_{max} \ge w_e \ge \lambda c_e. \tag{2.1.2}$$

If X_1 and X_2 have different cardinalities, we extend the two matroids by adding $||X_1| - |X_2||$ dummy elements to the ground set of weight and cost zero, and add them to the smaller of X_1 and X_2. We then truncate the two matroids to all independent sets of size $q = |X_1| = |X_2|$. Thus, X_1 and X_2 are maximum weight common bases of each of the two truncated matroids.

Before proceeding, we require some definitions and a lemma.

Definition 2.1.1. *The* ***common basis polytope*** *P of the matroid intersection problem is the polyhedron defined by all of the constraints of LR(λ^*). The* ***optimal face*** *of P is the convex hull of all optimal solutions to LR(λ^*).*

Definition 2.1.2. *Given a matroid $\mathscr{M} = (S, \mathscr{I})$, $X \in \mathscr{I}$, and $Y \subseteq S$, the* ***exchangeability graph*** *of $\mathscr{M}$ with respect to X and Y is the bipartite graph $ex_{\mathscr{M}}(X,Y) := (X \setminus Y \cup Y \setminus X, H)$, where $H = \{(x,y) : x \in X \setminus Y, y \in Y \setminus X, X \setminus \{x\} \cup \{y\} \in \mathscr{I}\}$.*

Lemma 2.1.3 (Exchangeability Lemma)**.** *Given $X \in \mathscr{I}$ and $Y \subseteq S$, if $ex_{\mathscr{M}}(X,Y)$ has a unique perfect matching, then $Y \in \mathscr{I}$.*

Lemma 2.1.4 (Gasoline Lemma)**.** *Given a sequence of k real values $a_0,...,a_{k-1}$ of total value $\sum_{j=0}^{k-1} a_j = 0$, there is an index $i \in \{0,1,...,k-1\}$ such that for any $0 \leq h \leq k-1$, $\sum_{j=i}^{i+h} a_{j(mod k)} \geq 0$.*

The remainder of the algorithm proceeds in two steps. First, we will find two adjacent solutions in the common basis polytope of $\mathscr{M}_1$ and $\mathscr{M}_2$ also of optimal cost under the Lagrangian weights. These two solutions will contain only elements in $X_1 \cup X_2$ and will maintain the fact that one is under budget and one exceeds the budget. We then compute the desired approximation from the two adjacent common bases using the Gasoline Lemma.

Lemma 2.1.5 ([1])**.** *Assume we have two matroids $\mathscr{M}_1 = (S,\mathscr{I}_1), \mathscr{M}_2 = (S,\mathscr{I}_2)$ and two common bases $X_1, X_2 \in \mathscr{I}_1 \cap \mathscr{I}_2$. Then X_1 and X_2 are adjacent extreme points in the common bases polytope if and only if the following conditions hold:*

1. *The exchangeability graph $ex_{\mathscr{M}_1}(X_1,X_2)$ has a unique perfect matching M_1.*
2. *The exchangeability graph $ex_{\mathscr{M}_2}(X_1,X_2)$ has a unique perfect matching M_2.*
3. *The union $M_1 \cup M_2$ forms a cycle.*

Corollary 2.1.6 ([1])**.** *Let $\mathscr{M}_1 = (S,\mathscr{I}_1), \mathscr{M}_2 = (S,\mathscr{I}_2)$ be two matroids. Moreover, let $Z \in \mathscr{I}_1 \cap \mathscr{I}_2$ and $Z \subseteq X_1 \cap X_2$. Then X_1 and X_2 are adjacent extreme points in the common basis polytope of $\mathscr{M}_1$ and $\mathscr{M}_2$ if and only if $X_1 \setminus Z$ and $X_2 \setminus Z$ are adjacent extreme points in the common basis polytope of $\mathscr{M}_1/Z$ and $\mathscr{M}_2/Z$.*

Since the new common bases that we find will always be a subset of $X_1 \cup X_2$, we delete all elements $S' = S \setminus (X_1 \cup X_2)$. We now present the lemma that shows we can always make progress towards adjacent common bases.

Lemma 2.1.7 ([1])**.** *There is a polynomial-time algorithm that, if X_1 and X_2 are not adjacent extreme points, finds a third maximum-weight common basis A with respect to w_{λ^*}, such that $X_1 \neq A \neq X_2$ and $X_1 \cap X_2 \subset A \subset X_1 \cup X_2$.*

We will reproduce here the proof of this lemma given in [1] as it makes explicit the polynomial-time algorithm used.

Proof. Let $Z = X_1 \cap X_2$. Without loss of generality, let $X_1 \setminus X_2 = \{s_1,..,s_r\}$ and $X_2 \setminus X_1 = \{t_1,..,t_r\}$. For $1 \leq i,j \leq r$, define $\mathscr{M}_1^{ij} := \mathscr{M}_1/Z - \{x_i,y_j\}$ and $\mathscr{M}_2^{ij} := \mathscr{M}_2/Z - \{x_i,y_j\}$.

Consider the following polynomial-time algorithm. For every $1 \leq i, j \leq r$, compute A_{ij}, a maximum Lagrangian weight common basis of $\mathscr{M}_1^{ij}$ and $\mathscr{M}_2^{ij}$. If there exists A_{ij} satisfying $|A_{ij}| = r$ and $w_\lambda(A_{ij}) = w_\lambda(X_1/Z)$, then $A = A_{ij} \cup Z$ is the desired third basis. (Note that $X_1 \neq A \neq X_2$ since x_i and y_j are not present in $\mathscr{M}_1^{ij}$ and $\mathscr{M}_2^{ij}$.)

If there does not exist such an A_{ij}, then no common basis A of $\mathscr{M}_1$ and $\mathscr{M}_2$ with the desired properties exists. Assume by contradiction that there is such an A. Choose i, j such that $x_i, y_j \notin A$. Then $A \setminus Z$ is a common basis of $\mathscr{M}_1^{ij}$ $\mathscr{M}_2^{ij}$. Hence $w_\lambda(A_{ij}) \geq w_\lambda(A \setminus Z)$, since A_{ij} is such a maximum Lagrangian weight common basis. Also, $|A_{ij}| = |A \setminus Z| = r$, and thus $A_{ij} \cup Z$ is a common basis of $\mathscr{M}_1$ and $\mathscr{M}_2$, implying that $w_\lambda(A_{ij} \cup Z) \leq w_\lambda(A)$. Hence $w_\lambda(A_{ij}) \leq w_\lambda(A \setminus Z)$. Thus $w_\lambda(A_{ij}) = w_\lambda(A \setminus Z) = w_\lambda(X \setminus Z)$, a contradiction. □

At all times, $X_1 \setminus X_2$ and $X_2 \setminus X_1$ are maximum Lagrangian weight common bases of $\mathscr{M}_1/(X_1 \cap X_2)$ and $\mathscr{M}_2/(X_1 \cap X_2)$. Once the process terminates, there is no other maximum-weight common basis A' of these two, since otherwise $A' \cup (X_1 \cap X_2)$ would have been found by the algorithm in Lemma 2.1.7. Since $X_1 \setminus X_2$ and $X_2 \setminus X_1$ are the only two maximum-weight common bases, the optimal face of the common basis polytope of the contracted matroids is the convex hull of these two points. Thus, they are adjacent on the optimal face of the common basis polytope of $\mathscr{M}_1/(X_1 \cap X_2)$ and $\mathscr{M}_2/(X_1 \cap X_2)$. Thus, by Corollary 2.1.6, X_1 and X_2 are adjacent in the common basis polytope of $\mathscr{M}_1$ and $\mathscr{M}_2$.

If $c(X_1) = C$ or $c(X_2) = C$, we are done by Theorem 1.5.12. Assume $c(X_1) < C < c(X_2)$. Without loss of generality, assume that $X_1 \setminus X_2 = \{s_1, ..., s_r\}$ and $X_2 \setminus X_1 = \{t_1, ..., t_r\}$.

We may now proceed to find our approximation.

Lemma 2.1.8 ([1]). *Given X_1, X_2 as above, there is a polynomial time algorithm which computes X' such that $X' \in \mathscr{I}_1 \cap \mathscr{I}_2$, $c(X') \leq C$, and $w(X') \geq opt - 2w_{max}$.*

Proof. Since X_1 and X_2 are adjacent in the common basis polytope, by Lemma 2.1.5 we have unique perfect matchings $M_1 = \{s_1t_1, ..., s_r, t_r\}$ in $ex_{\mathscr{M}_1}(X_1, X_2)$ and $M_2 = \{t_1s_2, t_2, s_3, ..., t_r, s_1\}$ in $ex_{\mathscr{M}_2}(X_1, X_2)$, and corresponding cycle $(s_1, t_1, s_2, t_2, ..., s_r, t_r)$ in the union of the two matchings. For $1 \leq j \leq r$, assign edge s_jt_j a weight $\delta_j := w_\lambda(t_j) - w_\lambda(s_j)$ and all other edges weight 0. Since X_1 and X_2 have the same Lagrangian weight, $\sum_{j=1}^r \delta_j = 0$. By the Gasoline Lemma, there exists an edge of the cycle such that all partial sums of the weights around the cycle starting at this edge are non-negative. Without loss of generality, assume s_1t_1 is such an edge.

Find the largest $k \leq r$ such that $c(X_1) + \sum_{j=1}^{k} (c(t_j) - c(s_j)) \leq C$. Since $c(Y_2) > C$, we have $k < r$, and by construction, $c(X_1) + \sum_{j=1}^{k} (c(t_j) - c(s_j)) > C - c(t_{k+1}) + c(s_{k+1})$.

We now show that the solution $X' = X_1 \setminus \{s_1, ..., s_{k+1}\} \cup \{t_1, ..., t_k\}$, satisfies the claim.

Because of how we chose k,

$$C - c_{max} \leq C - c_{t_{k+1}} < c(X') \leq C \tag{2.1.3}$$

By the Gasoline Lemma, we have

$$w_\lambda(X') \geq w_\lambda(X_1) - w_\lambda(s_{k+1}) \geq w_\lambda(X_1) - w_{max}. \tag{2.1.4}$$

Next, we prove that $X' \in \mathscr{I}_1 \cap \mathscr{I}_2$. Consider $X' \cup \{s_{k+1}\}$. Its symmetric difference with X_1 is $\{s_1, .., s_k, t_1, .., t_k\}$. Recall $s_i t_i$ is an edge of M_1 for $i \leq k$. Notice that

- $s_i \in X_1 \setminus (X' \cup \{s_{k+1}\})$
- $t_i \in X' \cup \{s_{k+1}\} \setminus X_1$
- $X_1 \setminus s_i \cup t_i \in \mathscr{I}_1$

Thus, $s_i t_i$ is also an edge of $ex_{\mathscr{M}_1}(X_1, X' \cup \{s_{k+1}\})$, so this graph has a perfect matching. This perfect matching must be unique, since otherwise M_1 would not be unique in $ex_{\mathscr{M}_1}(X, Y)$. Thus, by the Exchangeability Lemma, $X' \cup \{s_{k+1}\} \in \mathscr{I}_1$. Similarly, $X' \cup \{s_1\} \in \mathscr{I}_2$, so $X' \in \mathscr{I}_1 \cap \mathscr{I}_2$.

We now must bound the weight of X':

$$\begin{aligned}
w(X') &= w_{\lambda^*}(X') + \lambda^* c(X') \\
&= w_{\lambda^*}(X') + \lambda^* C - \lambda^*(C - c(X')) \\
&\geq w_{\lambda^*}(X_1) + \lambda^* C - w_{max} - \lambda^* c_{t_{k+1}} && \text{(by (2.1.3) and (2.1.4))} \\
&\geq w_{\lambda^*}(X_1) + \lambda^* C - 2w_{max} && \text{(by (2.1.2))} \\
&\geq OPT - 2w_{max},
\end{aligned}$$

where the final inequality follows from 2.1.1. □

Theorem 2.1.9 ([1]). *The budgeted matroid intersection problem admits a PTAS.*

Proof. Let $\varepsilon \in (0,1)$. If the optimum solution X^*contains fewer than $p := \lceil 2/\varepsilon \rceil$ elements, we can guess X^* by brute force in time $O(m^p) = O(m^{O(1/\varepsilon)})$, giving a PTAS. Otherwise, we guess the p elements X_H^* of largest weight in X^* by iterating over all possible choices. We contract the matroid by these elements and delete from the matroids all elements of weight larger than any of the contracted elements. We also decrease the budget accordingly. The maximum weight of any remaining element is

$$w'_{max} \leq w(X_H^*)/p \leq \varepsilon w(X_H^*)/2, \tag{2.1.5}$$

since w'_{max} is at most the weight of the least-weight element contracted, which is at most the average weight of all contracted elements. Additionally, $X_L^* := X^* \setminus X_H^*$ is an optimum solution for the intersection of the new matroids. We compute an independent set X' in the intersection using the algorithm in Lemma 2.1.8 of weight $w(X') \geq w(X_L^*) - 2w'_{max}$. We return the solution $X = X_H^* \cup X'$.

The algorithm runs in time $O(m^{p+O(1)}) = O(m^{O(1/\varepsilon)})$. Finally,

$$\begin{aligned} w(X) &= w(X_H^*) + w(X') \\ &\geq w(X_H^*) + w(X_L^*) - 2w'_{max} \\ &\geq w(X^*) - \varepsilon w(X_H^*) && \text{(by (2.1.5))} \\ &\geq (1-\varepsilon) w(X^*). \end{aligned}$$

□

2.2 Algorithm for Adwords auction

Recall that in the Adwords auction, a set of n buyers are bidding on a set of m advertising slots (products). For each product j, each buyer i supplies a bid b_{ij}. We cannot collect more than B_i profit from each buyer i.

The following $(1-1/e)$-competitive algorithm is presented in [4].

The linear programming formulation for the Adwords auction problem is as follows:

Algorithm 1 Primal-dual algorithm for Adwords

```
1: Initially $x_i := 0 \quad \forall i$
2: Upon arrival of product $j$, allocate to buyer $i$ that maximizes $b_{ij}(1-x_i)$
3: if $x_i < 1$ then
4:     Set $y_{ij} \leftarrow 1$
5:     Set $z_j \leftarrow b_{ij}(1-x_i)$
6:     Set $x_i \leftarrow x_i(1+\frac{b_{ij}}{B_i}) + \frac{b_{ij}}{(c-1)B_i}$
7: end if
```

$$\begin{aligned}
\max \quad & \sum_{j=1}^{m}\sum_{i=1}^{n} b_{ij}y_{ij} && \text{(P)}\\
\text{s.t.} \quad & \sum_{i=1}^{n} y_{ij} \le 1 \quad \forall 1 \le j \le m \\
& \sum_{j=1}^{m} b_{ij}y_{ij} \le B_i \quad \forall 1 \le i \le n \\
& y \ge 0
\end{aligned}$$

The dual of this linear program is as follows:

$$\begin{aligned}
\min \quad & \sum_{i=1}^{n} B_i x_i + \sum_{j=1}^{m} z_j && \text{(D)}\\
\text{s.t.} \quad & b_{ij}x_i + z_j \ge b_{ij} \quad \forall 1 \le i \le n, 1 \le j \le m \\
& x, z \ge 0.
\end{aligned}$$

Intuitively, we can think of x_i as a function of the fraction of buyer i's budget that has been used to obtain products, and z_j as the price of product j.

We consider a primal-dual algorithm that, rather than simply award each item to the highest bidder, simultaneously attempts to give products to a buyer with a high bid *and* plenty of budget remaining. The purpose of the x_i variables in the algorithm will be to ensure that those with large budgets remaining are favoured. Specifically, each x_i variable is exponential in the fraction of the budget B_i used.

The value of c above is determined later. Let $R_{max} = \max_{i,j}\left\{\frac{b_{ij}}{B_i}\right\}$ be the maximum ratio of any buyer's bid to its budget.

Theorem 2.2.1 ([4]). *Algorithm 1 is $(1-1/c)(1-R_{max})$-competitive, where $c = (1+R_{max})^{\frac{1}{R_{max}}}$. As $R_{max} \to 0$, the competitive ratio tends to $(1-1/e)$.*

Proof. We must prove that the algorithm produces a feasible dual solution and almost feasible primal solution, and that in each iteration, $\Delta D \leq (1+\frac{1}{c-1})\Delta P$, where ΔP and ΔD are the changes in value of the primal and dual solutions, respectively, over the iteration.

Dual feasibility: Consider the dual constraint corresponding to buyer i and product j. If $x_i \geq 1$, then the constraint is satisfied. Otherwise, the algorithm sets z_j to $b_{i'j}(1-x_i')$, where i' is the buyer i maximizing $b_{ij}(1-x_i)$. This guarantees that our constraint is satisfied at this point in the algorithm. In future iterations, x_i may only increase, so feasibility will be maintained.

Comparison of primal and dual values: In iterations where $x_i \geq 1$ for i maximizing $b_{ij}(1-x_i)$, there is no change in the primal or dual objective. Whenever the algorithm updates the primal and dual solutions, the change in the primal objective is b_{ij}. The change in the dual objective is:

$$B_i \Delta x_i + z_j = b_{ij}x_i + \frac{b_{ij}}{c-1} + b_{ij}(1-x_i) = b_{ij}\left(1+\frac{1}{c-1}\right)$$

Primal (near) feasibility: We never update the primal solution for buyers satisfying $x_i \geq 1$. We prove that for any buyer i, that if $\sum_j b_{ij}y_{ij} \geq B_i$, then $x_i \geq 1$. This is done by proving the following claim:

Claim 2.2.2 ([4])**.**

$$x_i \geq \frac{1}{c-1}\left(c^{\frac{\sum_j b_{ij}y_{ij}}{B_i}} - 1\right)$$

Proof. We prove the claim by induction on the iterations of the algorithm. Initially, it is true trivially. We are only concerned with iterations where buyer i obtains a product, say k. In such an iteration:

$$\begin{aligned}
x_i' &= x_i\left(1+\frac{b_{ik}}{B_i}\right)+\frac{b_{ik}}{(c-1)B_i} \\
&\geq \frac{1}{c-1}\left(c^{\frac{\sum_{j\neq k} b_{ij}y_{ij}}{B_i}}-1\right)\left(1+\frac{b_{ik}}{B_i}\right)+\frac{b_{ik}}{(c-1)B_i} && (2.2.1) \\
&= \frac{1}{c-1}\left[c^{\frac{\sum_{j\neq k} b_{ij}y_{ij}}{B_i}}\left(1+\frac{b_{ik}}{B_i}\right)-1\right] \\
&\geq \frac{1}{c-1}\left[c^{\frac{\sum_{j\neq k} b_{ij}y_{ij}}{B_i}}c^{\left(\frac{b_{ik}}{B_i}\right)}-1\right] && (2.2.2) \\
&= \frac{1}{c-1}\left[c^{\frac{\sum_{j} b_{ij}y_{ij}}{B_i}}-1\right]
\end{aligned}$$

Inequality 2.2.1 follows from the induction hypothesis and inequality 2.2.2 follows since for any $0 \leq x \leq y \leq 1$, $\frac{ln(1+x)}{x} \geq \frac{ln(1+y)}{y}$. Consider choosing $c = (1+R_{max})^{\frac{1}{R_{max}}}$. Then for $x = \frac{b_{ik}}{B_i}$ and $y = R_{max}$, we have that

$$ln(c) = \frac{ln(1+y)}{y} \leq \frac{ln(1+x)}{x}.$$

Thus, $c^x \leq 1+x \leq 1+y$, giving $c^{\left(\frac{b_{ik}}{B_i}\right)} \leq 1+\frac{b_{ik}}{B_i}$ as desired. □

So as soon as a buyer's budget is exhausted, we stop allocating products to the buyer. Thus, at most one iteration exists where i receives j but is charged less than b_{ij}. Therefore, for each i, $\sum_j b_{ij}y_{ij} \leq B_i + \max_j\{b_{ij}\}$, and thus the profit extracted from i is at least

$$\left[\sum_j b_{ij}y_{ij}\right]\frac{B_i}{B_i+\max_j\{b_{ij}\}} \geq \left[\sum_j b_{ij}y_{ij}\right](1-R_{max}).$$

Thus, we lose at most a $(1-R_{max})$ factor of the profit.

The competitive ratio of the algorithm follows by Corollary 1.5.4. □

Chapter 3

Zip's Assignment and Purchasing Problems

In this chapter, we discuss suggestions for the solutions to Zip's assignment and purchasing problems. The two main shortcomings of Zip's current DVD assignment process are the inability of the overall performance metric to capture all of the relevant statistics of the daily assignment and the inefficiency and opaqueness of the assignment algorithm. We will discuss both of these and offer suggestions as to how they could be remedied.

Once we have resolved these problems, we will encounter a new problem: the testing of our new algorithm and metric. Since it is impossible to judge the quality of a daily metric by considering individual thirty-minute runs, we need a methodology for ensuring, or at least gaining a reasonable amount of confidence, that our algorithm will perform well over an entire day. To gain this assurance, we construct a mailroom simulation environment in which we test our algorithm. The simulation methodology and results are discussed. Finally, we compare results of real-time runs of the new algorithm with control over Zip's operations with the average results of Zip's old algorithm.

We formulate a problem that models Zip's purchasing problem and notice that it is an instance of matching augmentation. We further provide a special case that is polynomial-time solvable and discuss the development of a purchasing algorithm based around this methodology.

3.1 Metric improvement

We first attempt to improve Zip's daily metric. First, consider the evaluation of the quality of the DVDs sent to users. Currently, Zip uses a combination of the number of assignments of rank one and the number of assignments of rank ten or better. Assume that in the metric, these two are given weights w_1 and w_2, respectively. This means that a rank one disc contributes $w_1 + w_2$ to the metric, ranks two through ten contribute w_2, and ranks eleven and worse contribute nothing. This valuation seems extremely arbitrary. Is a rank ten disc that much better than a rank eleven disc? Are rank two and rank nine equally desirable? Or 11 and 500? It seems that such a discrete valuation is undesirable. It is not difficult to replace this current metric by a continuous one. Intuitively, the metric should be decreasing and exhibit diminishing changes (i.e. there is more of a difference between rank five and six than between fifteen and sixteen). We propose that a disc of rank r should have contribution proportional to $\frac{1}{r^c}$, where $c \geq 1$ is a parameter than Zip can set as desired. As c increases, we put more importance on highly desirable titles and almost no importance on those towards the bottom of the rankings. To keep the contribution of a rank one assignment the same as in the previous metric, we can include this rank metric in the overall metric with weight $w_1 + w_2$.

The second change to the metric will be the inclusion of service indices. Currently, Zip does not include these indices in their metric, but does use this information to divide the users with open slots into groups. This grouping is extremely similar to what Zip originally did with the DVD ranks; an arbitrary division is made and users receive one of a few discrete levels of service. In particular, an assignment is calculated for those in the top division, ignoring everyone else. Using the remaining available inventory, an assignment is calculated for the second division, and so on. This procedure is poor not only because of the discrete levels of service, but also because it involves creating multiple assignments in isolation. This can lead to suboptimal results.

The change to the metric will be similar to the change in the rank metric; we will enforce a more continuous range of service quality. We propose that for each possible assignment of disc j to user i, we calculate the contribution to the metric as before (using the new rank metric). This value should then be multiplied by $s_i^{1/d}$, where $s_i \in [0,1]$ is the service metric and $d \geq 0$ is a parameter that Zip can set as desired. A higher value of s_i denotes that user i should receive better service. As d increases, the range of users that receive good service increases and severely neglected users are not treated much differently than mildly neglected users.

Now that we have a more complete metric, we will consider an algorithm for maximizing the metric every thirty minutes.

3.2 Algorithm improvement

Note that for the time being, we will consider the individual assignment runs as offline problems. Two algorithmic ideas for modeling Zip's assignment problem as being online are discussed in detail in Chapter 4. It would be interesting to consider each thirty-minute period as a single node arriving online and the set of assignments we can pick as the incident edges. This may lead to an improved daily result if implemented carefully and is an interesting future direction.

The assignment problem facing Zip is essentially an instance of *B*-matching on a bipartite graph with some additional constraints. Consider the following construction:

1. For each user i with an open slot, create a vertex u_i with capacity equal to the number of open slots owned by i.
2. For each title j and each DC k in which there are copies of j available, create a vertex v_{jk} with capacity equal to the number of copies of j available at k.
3. For each u_i, v_{jk}, create an edge e_{ijk} of weight equal to the contribution of the assignment of title j to user i shipped from DC k to the metric.

If this were the entire instance, we could solve this easily by linear programming. However, the problem has several other constraints that must be encoded. These include:

1. A user i may receive at most $list_{ij}$ copies of title j, where $list_{ij} = 1$ if j is ranked by i, and $list_{ij} = 0$ otherwise.
2. A DC k can handle at most cap_k assignments (since we may wish to limit the size of the assignment we give to any one mailroom, to enforce that only the best quality assignments are filled).
3. The total number of assignments is at most $totCap$ (since we may wish to limit the total assignment size, for reasons similar to those for the limits on individual DCs).

This problem can be formulated as the following integer program, where we denote by I, J, and K the set of users, titles, and DCs, indexed by i, j, and k, respectively.

$$\begin{aligned}
\max \quad & \sum_{ijk} w_{ijk} x_{ijk} && (IP_{Zip}) \\
& \sum_{jk} x_{ijk} \leq slots_i \quad \forall i \in I && (1) \\
& \sum_{i} x_{ijk} \leq avail_{jk} \quad \forall j \in J, k \in K && (2) \\
& \sum_{k} x_{ijk} \leq list_{ij} \quad \forall i \in I, j \in J && (3) \\
& \sum_{ij} x_{ijk} \leq cap_k \quad \forall k \in K && (4) \\
& \sum_{ijk} x_{ijk} \leq totCap && (5) \\
& x_{ijk} \in \{0,1\} \quad \forall i \in I, j \in J, k \in K
\end{aligned}$$

In Theorem 5.1.2, we will see that the canonical linear programming relaxation of (IP_{Zip}) is integral. Thus, we can solve the integer program efficiently by simply solving the linear programming relaxation to optimality.

We can also arrive at this result more directly. Recall by Remark 1.7.8 that if we can express (IP_{Zip}) as a maximum weight *st*-flow problem with capacities coming from the right-hand side of (IP_{Zip}), the constraint matrix of (IP_{Zip}) is totally unimodular. This gives the desired result from the previous paragraph. Alternatively, it implies that we can solve the underlying problem by solving the resulting maximum weight *st*-flow problem, which can also be done efficiently.

Consider the following construction of a maximum weight *st*-flow instance illustrated in Figure 3.1:

Add

- source node s
- node s' corresponding to constraint (5) and an arc (s, s') with weight 0 and capacity $totCap$.
- node k and an arc (s', k) with weight 0 and capacity cap_k for each constraint k of (4).
- node jk and an arc (k, jk) with weight 0 and capacity $avail_{jk}$ for each constraint jk of (2).
- sink node t

- node i and an arc (i,t) with weight 0 and capacity $slots_i$ for each constraint i of (1).
- node ij and an arc (ij,i) with weight 0 and capacity $list_{ij}$ for each constraint ij of (3).

Finally, add an arc (jk,ij) with weight w_{ijk} and infinite capacity, for all $i \in I$, $j \in J$, $k \in K$.

Solving this maximum weight *st*-flow problem is equivalent to solving (IP_{Zip}), where the flow across arc (jk,ij) in the flow instance corresponds to the value of variable x_{ijk} in the linear program. The correspondence of solutions in the two instances is easy to see. If we wish to increase the flow along an arc (jk,ij), we must also increase the flow along arcs (s,s'), (s',k), (k,jk), (ij,i), and (i,t) in order to maintain flow conservation. This corresponds to increasing the size of the total assignment, the size of the assignment from DC k, and so on. In fact, the excess capacity of any arc in the flow instance (except for those of the form (jk,ij)) represents the slackness of the corresponding constraint in the linear program.

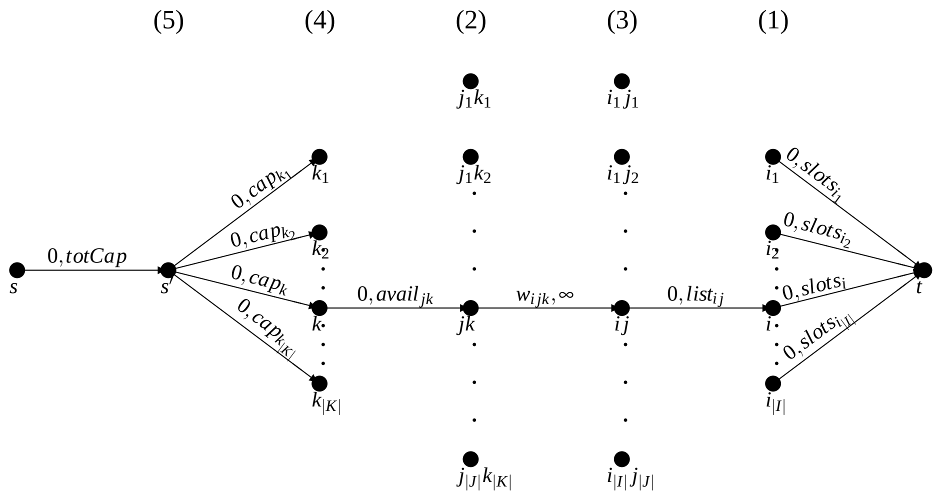

Figure 3.1: Construction of a maximum weight *st*-flow instance to solve (IP_{Zip}) (arc e labelled with w_e, c_e)

Now that we have an algorithm that will efficiently produce solutions that will optimize any metric of Zip's choosing in every thirty-minute period, we now have to test the ability of the algorithm to produce good daily results.

3.3 Algorithm testing

We have shown that we are capable of solving Zip's assignment problem every thirty minutes efficiently. However, as mentioned earlier, we have no guarantee that we will be producing desirable results for the entire day. It is entirely possible that the assignments we are selecting early in the day are not part of the optimal solution; in fact, choosing optimal partial assignments early in the day could be greatly damaging the daily assignment by making the daily optimal solution unobtainable. We illustrate this possibility in Figure 3.2. This may be unavoidable, but it is desirable to test if the assignments we produce are at least close to optimal; at the very least, we would like to test if we are making an improvement over Zip's previous results. It is easy to do this for each assignment run throughout the day; in fact, it is unnecessary, since we are solving for the optimal solution. However, it is not possible to compare daily results. To do this, the new algorithm would need to have control over the mailrooms of the DCs. However, this would mean giving the new algorithm control over operations before it has been tested, the exact scenario we are trying to avoid.

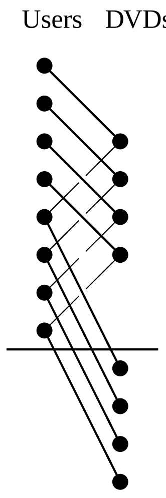

Figure 3.2: By selecting the assignments marked with broken lines before the bottom four vertices arrive, we miss out on the assignment marked with thick lines

To solve this testing problem, we constructed a simulation environment for Zip's mailroom. We were provided by Zip with the status of the database given to the assignment algorithm every thirty minutes and the value of their metrics for the day. By comparing status changes between adjacent runs, we were able to obtain, for each thirty minute period, a reasonable approximation of the DVDs that were returned into inventory and the slots that became open. In order to make fair comparisons with Zip's performance, we made the following assumptions:

1. The work capacity for a DC for a time period is taken to be the actual work done by Zip's mailroom during the time period that day. Note that this means we cannot hope to fill more slots than Zip, and are thus simply testing the quality of the assignment.

2. The mailroom processes assignments uniformly at random over all the possible assignments given to it.

3. The run time of the algorithm is five minutes (Our LP algorithm runs in significantly less time, but for comparison of assignment quality, the amount of scrubbing necessary should be kept constant. Recall from the introduction that scrubbing is the process of removing assignments that are no longer possible to fill due to slots or inventories being utilized during the physical runtime of the algorithm.)

4. We used the original metric so that we could compare our performance directly with the results obtained by Zip (it would make no sense to compare results if we attempted to maximize two different quantities)

Additionally, in each period, we compute an assignment of size at most three times the mailroom capacity. This is a reasonable compromise between giving a large assignment, resulting possibly in suboptimal DVDs being chosen for shipment, and giving a small assignment, resulting in decreased mailroom efficiency. For each assignment we produce, we will group the assignment according to the location of the DVDs being assigned. For the assignments in each DC k, we will randomly select a subset of the assignments to be processed. We further divide this chosen subset into R_k and S_k, where S_k represents the assignments that will be processed during the physical runtime of the next iteration of the algorithm. Thus, the assignments in S_k may be subject to scrubbing. In the simulator, each assignment run proceeds as follows according to Algorithm 2.

In addition to our and Zip's results, we computed the optimal assignment for the day. This is done by simply assuming the mailroom has infinite capacity and waiting until the last time period of the day to produce any assignments.

Algorithm 2 Simulator

```
1: Initially, set $R_k,S_k = \emptyset \ \forall$ DCs $k$, $\mathscr{A} = \emptyset$
2: **while** not end of day **do**
3:     Update database with discs and slots that have become available in the last thirty minutes
4:     Calculate assignment $A = \{A_k\}$ using our LP algorithm
5:     Scrub $A$ to remove conflicts with $S_k \ \forall k$; remove $S_k$ from the database.
6:     For each DC $k$, select uniformly at random $R_k,S_k \subset A_k$, $R_k \cap S_k = \emptyset$, such that $|R_k| = \frac{5cap_k}{6}$, $|S_k| = \frac{1cap_k}{6}$. Add $R_k,S_k$ to $\mathscr{A}$.
7:     Remove $R_k$ from the database $\forall k$
8: **end while**
9: Return $\mathscr{A}$
```

Metric	Zip	LP-Alg	No-Scrub	Optimal
Slots Filled	8,779	8,540	8,505	9,082
Top 10	6,358	5,608	5,446	5,622
Top 1	1,241	1,803	1,761	1,665
Home DC	4,786	6,431	6,466	5,889
Metric	5,569	5,567	5,526	5,877
Metric (no SF)	2,936	3,005	2,975	3,153

Table 3.1: Simulator results

The simulator was used to test an additional linear programming based algorithm. The goal of this alternate algorithm, NoScrub, is to eliminate scrubbing, preserving all computed assignments. The individual assignments are computed in an identical manner to the LP-based assignment algorithm, except that when we calculate the next assignment thirty minutes later, we assume that *every* DVD in the assignment was shipped. The result is that the corresponding slots and DVDs are not present in the next assignment run, so no scrubbing will be necessary; the next assignment is made only using slots and DVDs that were not involved in the previous assignment. Once this new assignment is computed, the piece of the previous assignment that was not shipped is returned to the database.

The results of the testing are summarized in Table 3.1.

Since we have constructed the simulator in such a way that we can never fill more slots than Zip, it is difficult to derive a fair comparison in the value of the slots filled metric. For this reason, we have also included the value of the metric without the slots filled measure. This alternate metric measures, in some respect, the *quality* of the assignment, whereas slots filled measures the *quantity*. We notice

Metric	LP-Alg Day 1	LP-Alg Day 2	Zip-Alg Average of Previous Week
% Top 10	69.3%	73.9%	77.4%
% Top 1	19.6%	20.6%	15.7%
% Home DC	62.2%	62.9%	60.1%
% Users with an open slot at end of day	2.9%	1.8%	5.3 %

Table 3.2: Real-time implementation results

both that the quality of the assignments produced by the two linear programming-based algorithms is better than Zip's results and that our results are not terribly worse than the optimal assignment for the day. Both of these observations are very promising for the performance of our algorithm.

These results, along with on-site stability testing, prompted Zip to switch over to the LP-based algorithm. The quality of the assignments produced in the first two days of activity is summarized in Table 3.2, along with the average results for the previous week seen by Zip before the algorithm switch for comparison.

The new linear programming-based algorithm has made drastic improvements in the percentage of users with an open slot at the end of the day, reducing this percentage by more than half on average over the two days. Having unfilled slots is a main driver of users leaving Zip, so such a dramatic reduction in this number is valuable. The percentage of slots filled with top ranked discs also increased considerably, rising more than four percent on average over the two days. We also decreased cross-shipping slightly, which will be likely to provide improved results in the future as discs are returned sooner. The one decrease came in the percentage of slots filled by discs ranked in the top ten, but this was to be expected; we removed from the daily metric what used to be a sharp threshold at rank-ten discs in favour of a smoother metric.

3.4 DVD Purchasing

The second problem we attempt to resolve is that of purchasing new DVDs. Purchasing must be done regularly in order to accumulate new releases as soon as they are available; the popularity of these titles will be at their peak around this time, so it is extremely important to have inventory of them available. DVDs that are lost, broken, stolen, etc. may have to be replaced if they are still popular. Finally, the popularity of DVDs may increase for other reasons at any time, and Zip may need to bolster their supply.

The most logical way to decide which DVDs to buy seems to be to do so in a way that maximizes the increase in the performance metric of the assignment algorithm. For now, we will assume that the only goal of the purchasing algorithm is, in fact, to increase the quality of the current assignment algorithm run by as much as possible (later in the chapter, we will discuss more long-term goals). This goal is conceptually easy to model. Recall the integer programming formulation of the assignment problem in use by the assignment algorithm.

$$
\begin{aligned}
\max \quad & \sum_{ijk} w_{ijk} x_{ijk} && (IP_{Zip}) \\
& \sum_{jk} x_{ijk} \le slots_i \quad \forall i \in I \\
& \sum_{i} x_{ijk} \le avail_{jk} \quad \forall j \in J, k \in K \\
& \sum_{k} x_{ijk} \le list_{ij} \quad \forall i \in I, j \in J \\
& \sum_{ij} x_{ijk} \le cap_k \quad \forall k \in K \\
& \sum_{ijk} x_{ijk} \le totCap \\
& x_{ijk} \in \{0,1\} \quad \forall i \in I, j \in J, k \in K
\end{aligned}
$$

The effect of purchasing a DVD j (and placing it in DC k) on this integer program is simply an increase of 1 of the value $avail_{jk}$. Thus, we can introduce a variable y_{jk} for each DVD j and DC k and add these to the integer program. However, Zip cannot purchase as many DVDs as they wish; they have a purchasing budget. If we call this purchasing budget C, and let the price of DVD j (at any DC k) be c_{jk}, we can add the constraint

$$\sum_{jk} c_{jk} y_{jk} \le C$$

to the integer program. Since the goal of the purchasing tool is to determine which DVDs would increase the metric most significantly, we will relax the temporal aspect of the process slightly and instead consider simply quality of assignments. Specifically, we will remove the capacity constraints, both on the DCs and overall, under the assumption that we are trying to make as many assignments with high metric value as possible. We will not be concerned if this process requires more than a single thirty-minute period, as it will still be increasing the total number of quality assignments. Finally, we stop differentiating between the DCs. Since we have the ability to place the newly purchased DVDs in any DC we wish, and most of the improvement in the assignment quality will be as a result of these new purchases, not distinguishing between DCs should not detract from the accuracy of the purchasing tool significantly. What remains is the following integer program:

$$\begin{aligned}
\max \quad & \sum_{ij} w_{ij} x_{ij} && (IP_{Purchasing})\\
& \sum_{j} x_{ij} \leq slots_i \quad \forall i \in I && (1)\\
& \sum_{i} x_{ij} \leq avail_j + y_j \quad \forall j \in J && (2)\\
& x_{ij} \leq list_{ij} \quad \forall i \in I, j \in J && (3)\\
& \sum_{j} c_j y_j \leq C && (4)\\
& x_{ij} \in \{0,1\}, y_j \in \mathbb{Z}_+ \quad \forall i \in I, j \in J
\end{aligned}$$

Remark 3.4.1. *The integer program ($IP_{Purchasing}$) specifies an instance of the matching augmentation problem on a bipartite graph, where the constraints of (3) specify the edges of the graph ($ij \in E$ if $list_{ij} = 1$).*

In Section 6.1.2, we give a PTAS for the matching augmentation problem that can be used to approximately solve this purchasing problem. If, instead of having a cost c_j for each DVD, we say that all DVDs have the same price, then ($IP_{Purchasing}$) defines an instance of one-sided bipartite matching augmentation with uniform costs, a special case of matching augmentation that we prove in Section 6.1.4 has a totally unimodular constraint matrix, and is thus solvable in polynomial time. Although the assumption that all DVDs have the same cost is slightly inaccurate, it is not completely unreasonable.

In practice, it seems myopic to make purchasing decisions based on obtaining the greatest possible improvement over one assignment run. One reasonable methodology to follow would be to run the purchasing algorithm over the data for an entire day and determine which additional DVDs would best improve the optimal daily solution. The purchasing algorithm could be run in this manner every day, and a database could be maintained keeping track of which titles are often included in the daily purchasing decision. When it is time to make an actual purchase, the titles that appear in the database most often could be purchased. The implementation of a buying tool for Zip based on the purchasing algorithm described in this section is an interesting direction for future work.

Chapter 4

Online problems

The problem Zip encounters is in some ways a hybrid between a traditional matching problem and an online matching problem. Zip's problem is online in that once a thirty minute period has passed, any unused mailroom capacity is lost forever; thus fewer assignments can be made in total. Also, once assignments are physically filled by the mailroom, they cannot be revoked. However, an edge that 'arrives' can be put in the matching several hours later, assuming both ends are still uncovered. This is not the case in most online problems. Another difference is that in Zip's case, the vertices do not appear online one at a time, but rather in blocks. Despite these differences, a large online component exists in Zip's operations, so it is worth discussing some online models.

In this section, we will discuss two different online problems. The first is simply online matching. The online matching problem is relevant to Zip since it must also make irrevocable decisions without knowledge of the future. Although Zip has the option to wait and collect more information about the future before assigning edges to the matching, doing so comes at the expense of unused mailroom capacity. It is therefore in their best interest to make some assignments as soon as possible, as in the online matching problem.

The second problem is the Adwords auction. The Adwords auction models Zip's situation very well in some aspects. If we think of the bids as the increase in Zip's metric as a result of making the assignment, solving Adwords is similar to solving Zip's assignment problem over a period of time in the more restrictive setting where DVDs arrive one at a time. The budget for each user can be thought of as a mechanism to ensure that many users receive good service, as no one user will be given assignments of value too large. Recall that in the algorithm for the Adwords auction discussed in Section 2.2, buyers with a higher fraction of unused budget were more likely to receive products. Thus, as some users start to receive assignments, it is more likely that other users receive subsequent assignments. Additionally, the budget can be set higher for those users with high service indices to direct the algorithm to serve these users better.

We will consider generalizations of both of these problems, online budgeted matching and the budgeted Adwords auction. Recall that in each, we have added costs to the edges, and require that the solutions we produce have total cost of at most some given bound. The addition of budgets to the problems is particularly relevant in Zip's problem. In the budgeted Adwords auction, for example, if we set the cost of every edge to 1, then we can enforce mailroom capacity constraints on our assignment. If we set the costs to correspond to a user's service index in some way, we can guide the algorithm to provide better service for recently neglected users. The costs and total budget can be used to enforce several different properties, so the problems are worth considering.

4.1 Online 0-1 budgeted matching

Recall the online matching problem:

Given a bipartite graph $G = (U \cup V, E)$ and weights $w \in \mathbb{R}^E$, find a matching M of maximum weight $w(M)$.

We will generalize the online matching problem slightly by adding costs $c_e \in \{0,1\}$ to each of the edges. We will sometimes refer to edges e with $c_e = 1$ as red edges, and with $c_e = 0$ as black edges. We will be able to assign at most C red edges.

This problem can be formulated as the following linear program:

$$\begin{aligned} \max \quad & \sum_e w_e y_e && \text{(P)} \\ \text{s.t.} \quad & \sum_{e \in \delta(v)} y_e \le 1 \quad \forall v \in U \cup V \\ & \sum_e c_e y_e \le C \\ & y \ge 0 \end{aligned}$$

The dual of this linear program is as follows:

$$
\begin{aligned}
\min \quad & \sum_{v \in U \cup V} z_v + Cx && \text{(D)}\\
\text{s.t.} \quad & z_u + z_v + c_{uv}x \geq w_{uv} \quad \forall uv \in E\\
& z, x \geq 0
\end{aligned}
$$

Notice that at the onset of the problem, we do not know the entire primal or dual LP. In fact, the dual has no constraints at first, as none of E is known. We present a primal-dual algorithm for the problem that seeks to maintain both primal and dual feasibility at all times; as the dual constraints appear, we will adjust the z and x variables to ensure the new constraints are satisfied. The algorithm is conceptually extremely simple. When a vertex v arrives online, if there exist black edges incident to v and uncovered neighbours of v, we select one of them. Otherwise, if the are red such neighbours (and we are under budget), we pick one of these.

The subtle aspect of the algorithm is the maintenance of the dual variables. We have a z variable for each vertex. All z variables start as zero, and remain zero until an incident edge uv is selected. At this point, z_u and z_v are both set to w_{uv}. The only wrinkle is that we keep track of the vertices of V that are incident to chosen red edges (done by $\mathscr{C}$ in Algorithm 3). Once we reach our budget, we will need to set the dual variable x to w_{max} to satisfy upcoming dual constraints corresponding to red edges. This increase in x increases the dual objective, potentially ruining our approximation. Thus, once we reach our budget, we will set to zero all components of z corresponding to variables in $\mathscr{C}$ as these constraints will be satisfied by our choice of x. Algorithm 3 makes formal this primal-dual approach.

First, note the following lemmas:

Lemma 4.1.1. *The value of z_u or z_v is non-zero only for covered u or v, and is bounded by w_{max}.*

Proof. The value of any component of z is only ever changed in steps 4,8,12,or 15 of the algorithm.

4: z_u is only increased if u is covered.

8: u is chosen such that it is uncovered and v has just arrived online, so it is uncovered

12: Same as Step 8

15: z_v is fixed at zero, so does not meet the precondition of the lemma

Finally, every time a component of z is modified (with the exception of in Step 15), it is set to an edge weight or a maximum of itself and an edge weight. Thus, w_{max} bounds the value of any component of z. □

Algorithm 3 Primal-dual algorithm for online 0-1 budgeted matching

```
 1: Initially all variables set to 0, $\mathscr{C} = \emptyset$.
 2: Ignore all edges $e$ of weight $w_e \leq 0$
 3: Upon arrival of vertex $v$:
 4: For all covered $u$ adjacent to $v$, if $c_{uv} = 0$ or $x = 0$, set $z_u \leftarrow \max\{z_u, w_{uv}\}$
 5: if $S$, the set of uncovered $u$ adjacent to $v$ with $c_{uv} = 0$, is non-empty then
 6:     Select $u \in S$ such that $w_{uv}$ is maximized
 7:     Set $y_{uv} \leftarrow 1$
 8:     Set $z_v, z_u \leftarrow w_{uv}$
 9: else if $T$, the set of uncovered $u$ adjacent to $v$ with $c_{uv} = 1$, is non-empty, and
    $x = 0$ then
10:     Select $u \in T$ such that $w_{uv}$ is maximized
11:     Set $y_{uv} \leftarrow 1$
12:     Set $z_v, z_u \leftarrow w_{uv}$
13:     Set $\mathscr{C} \leftarrow \mathscr{C} \cup v$
14:     if $|\mathscr{C}| = C$ then
15:         Fix $z_v \leftarrow 0 \; \forall v \in \mathscr{C}$, $x \leftarrow w_{max} := \max_{e \in E} w_e$
16:     end if
17: end if
```

Lemma 4.1.2. *The value of z_u never decreases and the value of z_v never decreases for $v \notin \mathscr{C}$.*

Proof. The values of both z_u and z_v are zero until they are set to some non-negative weight in step 8 or 12. Since the vertices become covered at this point, they are never set in these steps again. For z_u, the value may be changed in step 4, but only increased. For $z_v \notin \mathscr{C}$, the value is never changed again. □

We now prove the competitive factor for our algorithm.

Theorem 4.1.3. *Define $w_{min} := \min_{e \in E : w_e > 0} w_e$. Algorithm 3 is $\frac{1}{2\frac{w_{max}}{w_{min}}}$-competitive for the online 0-1 budgeted matching problem.*

Proof. We must show that after each iteration, both the primal and dual solutions are feasible, and that the total value of the primal solution is within a $\frac{1}{2\frac{w_{max}}{w_{min}}}$ factor of the value of the dual solution.

Primal feasibility: Since we only ever add edges incident to uncovered vertices, we produce a matching. Also, since we set $x = w_{max}$ after adding C red edges, we never add more than C red edges, so the budget is respected.

Dual feasibility: Consider an edge uv arriving online with vertex v. If $c_{uv} = 0$, we have two cases.

- If u is uncovered, then either uv or an edge of larger weight is chosen to the matching, so we set $z_v \geq w_{uv}$, satisfying the dual constraint corresponding to edge uv. By Lemma 4.1.2, z_v never decreases for $v \notin \mathscr{C}$, so this constraint remains satisfied.
- If u is covered, then we immediately set $z_u = \max\{z_u, w_{uv}\}$, so $z_u \geq w_{uv}$. By Lemma 4.1.2, z_u never decreases, so this constraint remains satisfied.

In the case where $c_{uv} = 1$, we have three cases.

- If the budget has been reached, then $x = w_{max}$, satisfying the constraint.
- If we are under budget, and u is uncovered, then as in the $c_{uv} = 0$ case, $z_v \geq w_{uv}$, satisfying the dual constraint. If the budget is reached, then z_v is set to 0, but x is set to w_{max}, so the constraint remains satisfied.
- If we are under budget, and u is covered, then we set $z_u = \max\{z_u, w_{uv}\}$, so $z_u \geq w_{uv}$. By Lemma 4.1.2, z_u never decreases, so this constraint remains satisfied.

Comparison of primal and dual values: Lemma 4.1.1 implies that considering contributions of edges in the matching and their endpoints (and the variable x) will capture all non-zero variables.

Let M be the matching returned by the algorithm. For every $uv \in M$, we have a contribution of $w_{uv} \geq w_{min}$ to the primal value and $z_u + z_v \leq 2w_{max}$ to the dual value. This gives us primal value of at least $|M|w_{min}$ and dual value of at most $2|M|w_{max}$. If we reach the budget during the algorithm, the dual value increases by Cw_{max} due to the increase in x, but C of the z variables are permanently set to 0. Thus at most $2|M| - C$ of the z variables are non-zero, so the dual value is at most $(2|M| - C)w_{max} + Cw_{max} = 2|M|w_{max}$. Thus, our bound of $2|M|w_{max}$ is unchanged. So for any matching M generated by our algorithm, we have that the primal value is at least a $\frac{w_{min}}{2w_{max}}$-factor of the dual value.

The competitive ratio of the algorithm follows by Corollary 1.5.4. □

Remark 4.1.4. *In the case where we would like to find a matching of maximum cardinality (i.e. $w_e = 1\ \forall e \in E$), this algorithm produces a $1/2$-approximation.*

Notice that this matches the upper bound given for deterministic algorithms for online unbudgeted matching given in [14]. In this case, the addition of 0-1 budgets has not decreased the value of the optimal competitive ratio.

We now provide an instance proving that this primal-dual algorithm is optimal for a certain class of algorithms. Consider the instance shown in Figure 4.1.

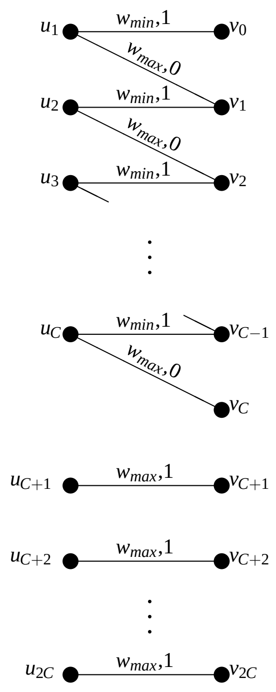

Figure 4.1: Lower bound instance of budgeted 0-1 online matching (edge e labelled with w_e, c_e)

Suppose we have any algorithm A that always selects *some* edge whenever there is one available (of positive weight), and selecting this edge will not exceed the budget. In other words, every time a vertex arrives online that has a non-empty set of uncovered neighbours, the algorithm selects one of these edges to the matching whenever it can do so and still respect the budget (it is irrelevant how the choice of which edge is made).

When we run A on Figure 4.1, it will select edge u_1v_0 when v_0 arrives. When v_1 arrives, the only uncovered neighbour is u_2, so this edge is selected. This continues until A has selected $\{u_iv_{i-1} : 1 \leq i \leq C\}$. At this point, A has selected C red edges, so can not select any of the upcoming edges. The total weight of the matching computed is Cw_{min}. The optimal matching for this instance is $\{u_iv_i : 1 \leq i \leq 2C\}$, which has weight $2Cw_{max}$. This ratio matches the performance guarantee of our primal-dual algorithm.

This example shows that to improve upon the $2\frac{w_{max}}{w_{min}}$ ratio, the algorithm must, in some scenarios, decline to add an edge, a locally suboptimal choice. Designing an algorithm of this nature would almost certainly require randomness, as it is fairly easy, given a deterministic algorithm that declines edges on occasion, to give an instance on which it performs poorly. Even given randomness, it is difficult to decline edges without resulting in poor results for some instances. The design of an algorithm that beats a competitive ratio of $1/2$ is an interesting open problem.

Another interesting algorithmic question is the design of an algorithm that violates the budget by some small amount. It is possible that an algorithm that is allowed some violation could produce better competitive ratios. The design of algorithms for the online 0-1 budgeted matching problem that allow budget violation is another interesting direction.

4.2 Budgeted Adwords auction

Recall the Adwords problem:

Given a bipartite graph $G = (U \cup V, E)$, define $n = |U|$ and $m = |V|$. Given bids $b \in \mathbb{R}^E$ and budgets $B \in \mathbb{R}^U$, find $M \subseteq E$ such that:

- *for each $j \in V$, $\sum_{e \in M \cap \delta(j)} b_e \leq 1$*
- *$\sum_i \min\{B_i, \sum_{e \in M \cap \delta(i)} b_e\}$ is maximized*

We will consider adding a budget constraint to the Adwords problem. In the budgeted case, there will be c_{ij} for each buyer i and product j. We will require that the sum of the c_{ij} of allocated products is at most C.

The linear program formulation for this budgeted problem is as follows:

$$\begin{aligned} \max \quad & \sum_{j=1}^{m}\sum_{i=1}^{n} b_{ij}y_{ij} & \text{(P)} \\ \text{s.t.} \quad & \sum_{i=1}^{n} y_{ij} \leq 1 \quad \forall 1 \leq j \leq m \\ & \sum_{j=1}^{m} b_{ij}y_{ij} \leq B_i \quad \forall 1 \leq i \leq n \\ & \sum_{j=1}^{m}\sum_{i=1}^{n} c_{ij}y_{ij} \leq C \\ & y \geq 0 \end{aligned}$$

The dual of this linear program is as follows:

$$\begin{aligned} \min \quad & \sum_{i=1}^{n} B_i x_i + \sum_{j=1}^{m} z_j + C\alpha && \text{(D)} \\ \text{s.t.} \quad & b_{ij}x_i + z_j + c_{ij}\alpha \geq b_{ij} \quad \forall 1 \leq i \leq n, 1 \leq j \leq m \\ & x, z, \alpha \geq 0. \end{aligned}$$

Consider Algorithm 4, a primal-dual algorithm for the budgeted Adwords problem based on the primal-dual algorithm for unbudgeted Adwords presented in [4] discussed in Section 2.2. The algorithm we present is extremely similar to the algorithm in [4]; in fact the x variables are updated in an identical manner. The ideas behind the algorithm in [4] are extended in a very natural way to include an overall budget. The values of c and d will be chosen later.

Define:

- $R_{max} = \max_{i \in I, j \in J} \frac{b_{ij}}{B_i}$
- $C_{max} = \max_{i \in I, j \in J} \frac{c_{ij}}{C}$
- $S_{max} = \max_{i \in I, j \in J, b_{ij}, c_{ij} \neq 0} \frac{b_{ij}}{c_{ij}}$ (and similarly S_{min})
- $S = \frac{S_{max}}{S_{min}}$

Algorithm 4 Primal-dual algorithm for budgeted Adwords

```
1: Initially $x_i := 0 \quad \forall i$
2: Upon arrival of product $j$, allocate to buyer $i$ that maximizes $b_{ij}(1-x_i) - c_{ij}\alpha$
3: if $b_{ij}(1-x_i) - c_{ij}\alpha > 0$ then
4:     Set $y_{ij} \leftarrow 1$
5:     Set $z_j \leftarrow b_{ij}(1-x_i) - c_{ij}\alpha$
6:     Set $x_i \leftarrow x_i(1+\frac{b_{ij}}{B_i}) + \frac{b_{ij}}{(c-1)B_i}$
7:     Set $\alpha \leftarrow \alpha(1+\frac{c_{ij}}{C}) + \frac{c_{ij}S_{max}}{(d-1)C}$
8: end if
```

Theorem 4.2.1. *Algorithm 4 is* $(1-\frac{\frac{1}{c-1}+\frac{S}{d-1}}{1+\frac{1}{c-1}+\frac{S}{d-1}})(1-R_{max})(1-S \cdot C_{max})$*-competitive, where* $c = (1+R_{max})^{\frac{1}{R_{max}}}$ *and* $d = (1+C_{max})^{\frac{1}{C_{max}}}$*. As* $R_{max}, C_{max} \to 0$*, the competitive ratio tends to* $(1-\frac{S+1}{S+e}) = O(\frac{1}{S})$*.*

Proof. We must show that after each iteration, the dual solution is feasible, the primal solution is (almost) feasible an the change in value of the primal solution is within the bound given above of the change in the value of the dual solution

Dual feasibility: Consider the dual constraint corresponding to buyer i and product j. If $b_{ij}(1-x_i) - c_{ij}\alpha \leq 0$, then the constraint is satisfied. Otherwise, the algorithm sets z_j to $b_{i'j}(1-x'_i) - c_{i'j}\alpha$, where i' is the buyer maximizing $b_{ij}(1-x_i) - c_{ij}\alpha$. This guarantees that our constraint is satisfied at this point in the algorithm. In future iterations, x_i and α may only increase, so feasibility will be maintained.

Primal (near) feasibility: As in Claim 2.2.2, we have that:

$$x_i \geq \frac{1}{c-1}(c^{\frac{\sum_{j\in M} b_{ij}y_{ij}}{B_i}} - 1)$$

when we select $c = (1+R_{max})^{\frac{1}{R_{max}}}$. This implies that whenever $\sum_{j\in M} b_{ij}y_{ij} \geq B_i$, we have that $x_i \geq 1$, and thus we do not allocate any more products to buyer i. Therefore, the budget for each buyer is violated by at most one product. In removing this last product assigned to each buyer, we will reduce our profit by at most a $(1-R_{max})$ factor.

By an identical proof to as in Claim 2.2.2, we have that

$$\alpha \geq \frac{S_{max}}{d-1}(d^{\frac{\sum_{j\in M} c_{ij}y_{ij}}{C}} - 1)$$

when we select $d = (1+C_{max})^{\frac{1}{C_{max}}}$. This implies that whenever $\sum_{ij\in M} c_{ij}y_{ij} \geq C$, we have that $\alpha \geq S_{max}$. Thus, if we do not exceed our overall budget, there is no reduction in our performance ratio. If we do exceed our budget, we exceed it by at most $\max_{ij} c_{ij}$, since once the budget is exceeded, we stop adding items. To cut back on $\max_{ij} c_{ij}$ budget, we may lose as much as $S_{max} \max_{ij} c_{ij}$ profit. Since we have gone over budget, we have gained at least $S_{min}C$ profit, so our total profit is reduced by a factor of at most:

$$\frac{S_{min}C - S_{max}\max_{ij} c_{ij}}{S_{min}C} = 1 - S\cdot C_{max}$$

Comparison of primal and dual values: In iterations where $b_{ij}(1-x_i) - c_{ij}\alpha \leq 0$ for i maximizing this expression, there is no change in the primal or dual objective. Whenever the algorithm updates the primal and dual solutions, the change in the primal objective is b_{ij}. The change in the dual objective is:

$$\begin{aligned}
B_i\Delta x_i + z_j + C\Delta\alpha &= x_i b_{ij} + \frac{b_{ij}}{c-1} + b_{ij} - b_{ij}x_i - c_{ij}\alpha + \alpha c_{ij} + \frac{c_{ij}S_{max}}{d-1} \\
&= b_{ij} + \frac{b_{ij}}{c-1} + \frac{c_{ij}S_{max}}{d-1} \\
&= b_{ij}(1 + \frac{1}{c-1} + \frac{c_{ij}S_{max}}{b_{ij}(d-1)}) \\
&\leq b_{ij}(1 + \frac{1}{c-1} + \frac{S}{d-1})
\end{aligned}$$

Thus, the ratio of the change in the primal and dual objectives is:

$$1 - \frac{\frac{1}{c-1} + \frac{S}{d-1}}{1 + \frac{1}{c-1} + \frac{S}{d-1}}$$

The competitive ratio of the algorithm follows by Corollary 1.5.4. □

Remark 4.2.2. *In the case where $c_{ij} \in \{0,1\}$ $\forall ij \in E$ as in the online budgeted matching instance we considered, $S \leq \frac{b_{max}}{b_{min}}$, and we obtain an asymptotic competitive ratio of $O(\frac{b_{min}}{b_{max}})$ as $R_{max}, C_{max} \to 0$. This is asymptotically equivalent to the result obtained for online 0-1 budgeted matching.*

Chapter 5

Total Unimodularity of the Zip Linear Program

In Section 3.2, we formulated an integer program that would solve the assignment problem. Consider the linear programming relaxation of (IP_{Zip}):

$$\begin{array}{lll}
\max & \sum_{ijk} w_{ijk} x_{ijk} & (P_{Zip}) \\
\text{s.t.} & \sum_{jk} x_{ijk} \leq slots_i \quad \forall i \in I & (1) \\
 & \sum_{i} x_{ijk} \leq avail_{jk} \quad \forall j \in J, k \in K & (2) \\
 & \sum_{k} x_{ijk} \leq list_{ij} \quad \forall i \in I, j \in J & (3) \\
 & \sum_{ij} x_{ijk} \leq cap_k \quad \forall k \in K & (4) \\
 & \sum_{ijk} x_{ijk} \leq totCap & (5) \\
 & x_{ijk} \geq 0 \quad \forall i \in I, j \in J, k \in K &
\end{array}$$

We would like to be able to solve the integer program easily. If we knew that (P_{Zip}) always had an integral optimal solution, we could simply solve the LP to optimality and be left with a solution that is useful to Zip. On the other hand, if optimal solutions to (P_{Zip}) are fractional, more work is required.

In this section, we will show that the constraint matrix of (P_{Zip}) is totally unimodular, and thus (P_{Zip}) will have an integral optimal solution by Theorem 1.7.7 whenever the right hand side is integral (which will always be the case). We will reach this conclusion by proving a result on a generalization of (P_{Zip}).

5.1 A more general linear program

Consider the first block of constraints from (P_{Zip}):

$$\sum_{jk} x_{ijk} \leq slots_i \quad \forall i \in I$$

Notice that for each individual constraint in this block, we specify a certain value for some of our indices (i in this case), and then sum over all values of the other indices (j and k in this case). Each of the five blocks of constraints in (P_{Zip}) has this general form. We will give a general expression for a block of constraints of this form.

Let $I = \{i_1,..,i_n\}$ be a set of indices and let $S \subseteq \{1,..,n\}$, $S^* := \{1,..,n\} \setminus S$. Define $i_S := \{i_k : k \in S_j\}$, the set of indices specified by S. We would like to consider the constraint where we specify a certain value for all of the indices in i_S and then sum over all values of indices in i_{S^*}. Given an index set i_S, define $H(i_S)$ as the set of all possible tuples of values that the indices of i_S can assume. Let $h(i_S)$ be any such set of values. In other words, $H(i_S)$ is the set of all possible realizations of $h(i_S)$. For example, if $S = \{1,5,6\}$, then a possible realization of $h(i_S)$ could be $(i_1 = 3, i_5 = 2, i_6 = 8)$. Given this notation, we can express the desired constraint, where the indices in the set i_S take on the values $h(i_S)$, as

$$\sum_{h(i_{S^*}) \in H(i_{S^*})} x_{h(i_{S^*}),h(i_S)} \leq c_{h(i_S)},$$

where $c_{h(i_S)}$ is the right hand side bound for the constraint where the indices of i_S take on the values $h(i_S)$.

To express an entire block of constraints in this form, we simply have one constraint for each set of possible values $h(i_S)$ of the index set i_S:

$$\sum_{h(i_{S^*}) \in H(i_{S^*})} x_{h(i_{S^*}),h(i_S)} \leq c_{h(i_S)} \quad \forall h(i_S) \in H(i_S)$$

Notice that the above block of constraints is formed by specifying the set S. To arrive at a full generalization of (P_{Zip}), we need to have several such sets. Let $\mathscr{S} = \{S_0,..,S_{m-1}\}$, with $S_j \subseteq \{1,..,n\}$ $\forall 0 \leq j \leq m-1$. The entire constraint set is simply

$$\sum_{h(i_{S^*}) \in H(i_{S^*})} x_{h(i_{S^*}),h(i_{S_j})} \leq c_{h(i_{S_j})} \quad \forall h(i_{S_j}) \in H(i_{S_j}) \quad \forall 0 \leq j \leq m-1$$

The objective function will be a simple weighted sum of all variables which, using this notation, is expressed as

$$\max \quad \sum_{h(I) \in H(I)} w_{h(I)} x_{h(I)}$$

Together with non-negativity constraints, we arrive at our final generalized linear program P($I,\mathscr{S}$).

$$\begin{aligned} \max \quad & \sum_{h(I)\in H(I)} w_{h(I)} x_{h(I)} && \qquad\qquad (\mathrm{P}(I,\mathscr{S})) \\ \text{s.t.} \quad & \sum_{h(i_{S^*})\in H(i_{S^*})} x_{h(i_{S^*}),h(i_{S_j})} \le c_{h(i_{S_j})} \quad \forall h(i_{S_j}) \in H(i_{S_j}) \quad \forall 0 \le j \le m-1 \\ & x \ge 0 \end{aligned}$$

Remark 5.1.1. *$P_{Zip} = P(\{i,j,k\},\{\{1\},\{2,3\},\{1,2\},\{3\},\emptyset\}$, for appropriate values of c, where $S_0 = \{1\}$ corresponds to the index i and gives the set of constraints*

$$\sum_{jk} x_{ijk} \le slots_i \quad \forall i \in I,$$

$S_1 = \{2,3\}$ corresponds to the indices j, k and gives the set of constraints

$$\sum_{i} x_{ijk} \le avail_{jk} \quad \forall j \in J, k \in K,$$

and so on.

We will now state and prove a theorem than gives a sufficient condition for total unimodularity of the constraint matrix of (P($I,\mathscr{S}$)). First, define a partial order on $\mathscr{S}$. Let $S_j \le S_k$ if $S_j \subseteq S_k$. Considering $\mathscr{S}$ along with this partial order as a poset, we have the following result.

Theorem 5.1.2. *If $\mathscr{S}$ can be partitioned into two chains then the constraint matrix A of (P($I,\mathscr{S}$)) is totally unimodular.*

Before proving the theorem, note that since $\emptyset \le \{k\} \le \{j,k\}$ and $\{i\} \le \{i,j\}$, the index set of constraints (5),(4), and (2) form one chain, and the index sets of constraints (1) and (3) form a second chain. Thus, the theorem will prove that the constraint matrix of (P_{Zip}) is totally unimodular.

Proof. Let C_1 and C_2 be the two chains. Let B be any square submatrix of A. We must show that $det(B) \in \{-1,0,1\}$. Notice that no variable appears in two different constraints corresponding to the same set S_j. Without loss of generality, let $C_1 = (S_0,..,S_p)$, where $S_0 \subset .. \subset S_p$. Thus, for any two rows of B, r_j and r_k, that share any variable in common, where r_j comes from a constraint associated with S_j and r_k from S_k, with $0 \le j < k \le p$, we have that $r_k \subset r_j$.

For any two rows r_1 and r_2, we can replace either with $r_2 - r_1$ and only possibly change the sign of the determinant.

Consider the following procedure: Initialize $i := p$. Starting with all rows r_i corresponding to constraints from S_i, replace every row r_j of C_1 that satisfies $r_i \subset r_j$ with $r_j - r_i$. At the end of this iteration, any variable that appears in a row corresponding to a constraint of S_i appears nowhere else in rows corresponding to C_1. Decrement i and repeat. At the end of the procedure, the rows corresponding to C_1 have at most a single one per column. Perform the same procedure on C_2. Call the resulting matrix B'. Note that $|det(B')| = |det(B)|$.

The matrix B' has at most two ones in each column. If any column has no ones, then $det(B') = 0$, and we are done. If any column has a single one, we can expand by minors along this entry, reducing the size of the matrix, and proceed inductively. If every column has exactly two ones, we have by construction that the rows of C_1 and the rows of C_2 each contain a single one per column. Thus, the sum of the rows of C_1 equals the sum of the rows of C_2, implying that $det(B') = 0$. □

Remark 5.1.3. *Under the hypothesis of Theorem 5.1.2, we have that the rows of A are the union of two laminar families.*

A similar proof to the one above shows that any A whose rows are the union of two laminar families is totally unimodular. Such a proof is given in Goeman's lecture notes [11].

As with (IP_{Zip}), if we could model (P($I, \mathscr{S}$)) as a maximum weight *st*-flow problem, we would have both an alternative proof that the constraint matrix is totally unimodular by Remark 1.7.8 and another method to solve such problems efficiently.

The flow instance constructed will be very similar to the instance we constructed to model (IP_{Zip}). Recall that in this construction, we had the source node on one side, followed by nodes corresponding to constraints (5),(4), and (2) (in this order), then nodes corresponding to constraints (3) and (1), and finally the sink node. Note that (5),(4),(2) is one chain, and (3),(1) is the reverse of another chain. The general construction will follow this same format.

As in Theorem 5.1.2, assume that $\mathscr{S}$ can be partitioned into two chains C_1 and C_2. Assume without loss of generality that $C_1 = S_0 \subseteq S_1 \subseteq ... \subseteq S_p$ and $C_2 = S_{m-1} \subseteq S_{m-2} \subseteq ... \subseteq S_{p+1}$. Consider the following construction:

- Add a source s.
- Add a node $h(i_{S_0})$ and an arc $(s, h(i_{S_0}))$ with weight 0 and capacity $c_{h(i_{S_0})}$, for each $h(i_{S_0}) \in H(i_{S_0})$.

- For $1 \leq j \leq p$, add a node $h(i_{S_j})$ and an arc $(h(i_{S_{j-1}}), h(i_{S_j}))$ with weight 0 and capacity $c_{h(i_{S_j})}$, for each $h(i_{S_j}) \in H(i_{S_j})$ (where $h(i_{S_{j-1}})$ is the restriction of $h(i_{S_j})$ to the indices of $i_{S_{j-1}}$).
- Add a sink t.
- Add a node $h(i_{S_{m-1}})$ and an arc $(h(i_{S_{m-1}}), t)$ with weight 0 and capacity $c_{h(i_{S_{m-1}})}$, for each $h(i_{S_{m-1}}) \in H(i_{S_{m-1}})$.
- For $m-1 \geq j \geq p+1$, add a node $h(i_{S_j})$ and an arc $(h(i_{S_j}), h(i_{S_{j+1}}))$ with weight 0 and capacity $c_{h(i_{S_j})}$, for each $h(i_{S_j}) \in H(i_{S_j})$.
- Add an arc $(h(i_{S_p}), h(i_{S_{p+1}}))$ of weight $w_{h(I)}$ and infinite capacity for each $h(I) \in H(I)$. Note that if there exists an index $i \notin S_p, S_{p+1}$, we will construct multiple arcs for each value of $h(I)$, one for each possible value of index i.

As with (IP_{Zip}), the flows along arcs of the form $(h(i_{S_p}), h(i_{S_{p+1}}))$ correspond to values of $x_{h(I)}$, and excess capacities along all other arcs correspond to slackness of the corresponding constraints. Also as before, this instance has the property that each arc of the form $(h(i_{S_p}), h(i_{S_{p+1}}))$ lies along a unique directed st-path. Thus increasing flow along this arc requires an increase over the entire directed path, so there must exist excess capacity along the entire path, or equivalently, slackness in all constraints involving $x_{h(I)}$.

Note that the reverse direction of Theorem 5.1.2 does not hold. Consider $P(\{i,j,k\}, \{\{1,2\},\{2,3\},\{1,3\}\})$, where each of i,j,k takes on exactly two possible values, 1 and 2. Each constraint has exactly two variables. This set cannot be partitioned into two chains, but if we consider the bipartition

$$(\{x_{111}, x_{122}, x_{212}, x_{221}\}, \{x_{112}, x_{121}, x_{211}, x_{222},\})$$

we see that every constraint contains one variable from each set. Therefore, the constraint matrix is the incidence matrix of a bipartite graph, and thus totally unimodular.

A weaker version of the reverse direction does hold:

Theorem 5.1.4. *If $\mathscr{S}$ cannot be partitioned into two chains, and each index can take on at least four distinct values, then the constraint matrix A of $P(I, \mathscr{S})$ is not totally unimodular.*

Proof. By Dilworth's Theorem (Theorem 1.7.5), there exists an antichain C of size three. Without loss of generality, assume $C = \{S_0, S_1, S_2\}$. We will show that A

contains as a submatrix the incidence matrix of a 9-cycle. Since $det(C_9) = 2$, we will have that A is not totally unimodular.

We will now find the nine constraints and variables that correspond to the rows and columns of the 9-cycle. Assume without loss of generality that $\{0,1,2,3\}$ are valid values for each index. Constraint 0 will be the constraint determined by index set S_0, with all indices $i_j, j \in S_0$, set to 0. Constraint k, for $1 \leq k \leq 8$, will be the constraint determined by index set $S_{k \ mod \ 3}$, with each index $i_j, j \in S_{k \ mod \ 3}$, set equal to the value of i_j in constraint $k-1$ if $j \in S_{(k-1) \ mod \ 3}$. If $j \notin S_{(k-1) \ mod \ 3}$, set i_j equal to one greater (*mod* 3) than the value it had the last time it was set in a constraint, or 0 if it has never been previously set.

An example may help to illustrate this process. Let $S_0 = \{1,2,3\}$, $S_1 = \{1,3,4\}$, $S_2 = \{1,5\}$. Constraint 0 will be the one associated with S_0 with (i_1,i_2,i_3) forced to be $(0,0,0)$. In other words, select as constraint 0 the constraint

$$\sum_{i_4,i_5} x_{0,0,0,i_4,i_5} \leq c_{(i_1=0,i_2=0,i_3=0)}.$$

Constraint 1 will be the constraint associated with S_1 with (i_1,i_3,i_4) forced to be $(0,0,0)$. Note that i_1 and i_3 are 0 since they were 0 in the previous constraint, and i_4 is 0 because it has not been seen yet. Constraint 2 is associated with S_2 with (i_1,i_5) forced to be $(0,0)$. Constraint 3 forces (i_1,i_2,i_3) to be $(0,1,1)$; i_1 is 0 because it was in the previous constraint and i_2, i_3 are both 1 because they were not in the previous constraint, and in their previous appearances, they were both 0.

If we were to apply our iterative rule to constraint 8 to obtain a tenth constraint, it will be exactly constraint 0. This is because any index that is in all three sets will be fixed at 0 for the whole process, and any other index in S_0 will be removed and added three times throughout the nine constraints, so will be 0 again by the tenth constraint. In this sense, the numbering of the constraints is arbitrary; only the ordering is important.

The nine variables are selected as follows. For each consecutive pair of constraints k and $(k+1)$ *mod* 9, select the variable x^k whose indices match any values forced in constraints k and $(k+1)$ *mod* 9. Notice that by construction, we can never have contradictory assignments in the two constraints. For any index i_j not fixed in either constraint, set $i_j = 3$ in x^k.

Claim 5.1.5. *Of the nine chosen constraints, x^k appears in exactly constraints k and $(k+1)$ mod 9*

Once we have proven Claim 5.1.5, we see that the matrix defined by the variables x^k and the constraints k is the vertex-edge incidence matrix of a 9-cycle. Therefore, A is not totally unimodular. □

Proof. (Claim 5.1.5) It is clear that x^k appears in constraints k and $(k+1)$ *mod* 9, as these constraints set index values exactly as they appear in x^k. Let S_j be the set associated with constraint k. Consider any constraint $l \in \{(k+2),..,(k+8)\ mod\ 9\}$.

Assume l is associated with S_j or $S_{(j+1)\ mod\ 3}$. Then constraint l sets the same constraint set as one of constraint k and $(k+1)$ *mod* 9. Since C is an antichain, both S_j and $S_{(j+1)\ mod\ 3}$ contain an element not in $S_{(j-1)\ mod\ 3}$. The index associated with this element will not be set in any constraint associated with $S_{(j-1)\ mod\ 3}$, and thus will be set to a higher value the next time it appears. The increase will be by either 1 or 2 mod 3, so will not be the same as in x^k. Thus, x^k does not appear in this constraint.

Assume l is associated with $S_{(j+2)\ mod\ 3}$. Since C is an antichain, there exists $r \in S_{(j+2)\ mod\ 3} \setminus S_{(j+1)\ mod\ 3}$. Thus, i_r is set to some value $b \in \{0,1,2\}$ in constraint $(k+2)$ *mod* 9 and not set at all in constraint $(k+1)$ *mod* 9. Also, this implies that the value of i_r in constraint $(k+5)$ *mod* 9 is $(b+1)$ *mod* 3. If $r \in S_{(j-1)\ mod\ 3}$, then the r^{th} position of x^k is set to $(b-1)$ *mod* 3. If $r \notin S_{(j-1)\ mod\ 3}$, then the r^{th} position of x^k is set to 3. Since $b \notin \{(b-1)\ mod\ 3,\ 3\}$ and $(b+1)\ mod\ 3 \notin \{(b-1)\ mod\ 3,\ 3\}$, and this index is set in constraints $(k+2)$ *mod* 9 and $(k+5)$ *mod* 9, x^k is not in either constraint. By the same argument, this time using the fact that there exists $s \in S_{(j+2)\ mod\ 3} \setminus S_j$, we see that i_s is set to a value in constraint $(k-1)$ *mod* 9 that is different from the value of i_s in x^k, so x^k does not appear in constraint $(k-1)$ *mod* 9. □

Notice now why we need the fourth possible value for each index. In the case where $r = s$, we would need the value of i_r in x^k to be different from b, $b+1$, and $b-1$ *mod* 3 for the above proof to hold.

Chapter 6

Matching and Matroid Augmentation

In this chapter, we start by considering the matching augmentation problem, a theoretical problem that models the purchasing decisions faced by Zip. We show that the problem is NP-hard in general, but admits a PTAS. After we briefly consider several generalizations, we look at a tractable special case of the problem that retains some modeling power in the situation facing Zip.

The central idea behind the matching augmentation problem is that we are simultaneously searching for an optimal solution and actions that will increase the value of this optimal solution, subject to some resource (money in Zip's case). There is no reason to limit this discussion to the *b*-matching problem. In the second half of the chapter, we extend the idea of instance augmentation to the much more general framework of matroids and discuss several problems in this area.

6.1 Matching augmentation

Recall the matching augmentation problem:

Given a graph $G = (V,E)$, weights $w \in \mathbb{Q}^E$, capacities $B \in \mathbb{Z}^V$, costs $c \in \mathbb{Q}^V_+$, and a budget $C \in \mathbb{Q}$, find $y \in \mathbb{Z}^V_+$ and $M \subseteq E$ such that:

- $c^T y \leq C$
- *M is a $(B+y)$-matching*
- *M is of maximum weight $w(M)$*

The matching augmentation problem can be formulated as the following integer program:

$$
\begin{aligned}
\max \quad & \sum_{e\in E} w_e x_e && (IP_{MA})\\
\text{s.t.} \quad & \sum_{e\in\delta(v)} x_e \le B_v + y_v \quad \forall v\in V\\
& \sum_{v\in V} c_v y_v \le C\\
& x\in\{0,1\}^E, y\in\mathbb{Z}_+^V
\end{aligned}
$$

In the above formulation, $x_e = 1$ corresponds to including edge e in the matching, and the value of y_v corresponds to the additional capacity purchased for vertex v.

6.1.1 Complexity of matching augmentation problem

We will show that matching augmentation is NP-hard by a reduction to knapsack. Consider an instance of knapsack where $S = \{1..n\}$ is the set of items, $w' \in \mathbb{Q}^n$ are the weights of the items, $c' \in \mathbb{Q}_+^n$ are the costs of the items, and $C' \in \mathbb{Q}$ is the capacity of our knapsack. Consider the bipartite graph $G = (U\cup V, E)$ with $|U| = |V| = |S| = n$. For each $i \in S$, we let $B_{u_i} = 1$, $B_{v_i} = 0$, $c_{u_i} = \infty$, and $c_{v_i} = c'_i$. Also, we will join u_i and v_i with an edge e of weight $w_e = w'_i$. Along with a capacity augmentation budget of $C = C'$, this gives an instance of matching augmentation. An optimal solution to this instance would solve the corresponding Knapsack instance.

We now proceed to the construction of a PTAS for matching augmentation.

6.1.2 A PTAS for the matching augmentation problem

Our PTAS for matching augmentation will proceed as follows. First, we will guess the heaviest edges of the optimal solution, and remove all edges of weight larger than a certain threshold. On the remaining problem, we will define an instance of budgeted matching and approximate using the budgeted matching algorithm in [1]. From this approximation we will extract an approximation for the matching augmentation problem.

Let $\mathscr{I}$ be an instance of matching augmentation and let $\varepsilon > 0$ be a constant. Let M^* be optimal for $\mathscr{I}$. If M^* contains at most $p := \lceil \frac{8}{\varepsilon} \rceil$ edges, then we can solve the problem optimally by brute force in time $O(m^p) = O(m^{O(1/\varepsilon)})$ by simply guessing the optimal solution. Thus, assume that M^* contains more than p edges.

We will guess the heaviest p edges M_H^* of M^* by iterating through all possible choices of M_H^* with $|M_H^*| = p$. We remove from our graph the edges in M_H^* and all edges of weight larger than the smallest weight of an edge in M_H^*. For each endpoint v of an edge of M_H^*, we decrease B_v by one if $B_v \geq 1$, and otherwise reduce the budget C by c_v. Let $\mathscr{I}'$ be the resulting matching augmentation instance, with underlying graph $G' = (V, E')$. Note that in $\mathscr{I}'$, the maximum weight of an edge is

$$w'_{max} \leq \frac{w(M_H^*)}{p} \leq \frac{\varepsilon w(M_H^*)}{8}, \tag{6.1.1}$$

since the heaviest remaining edge has smaller weight than the least-weight edge in M_H^*, and thus has smaller weight than the average edge in M_H^*. Also, $M_L^* := M^* \setminus M_H^*$ is optimal for $\mathscr{I}'$.

To find an approximately optimal solution for $\mathscr{I}'$, we will define a suitable budgeted matching instance $\overline{\mathscr{I}'}$ as follows:

For each vertex $v \in V$, we will create $d_v = deg_{G'}(v)$ copies of v, $S_v := \{v_1, ..., v_{d_v}\}$. Let $S_v^1 = \{v_i : i \leq B_v\}$ and $S_v^2 = \{v_i : B_v < i \leq d_v\}$.

For each edge $e = uv \in E'$, we will create two new vertices, t_e^u and t_e^v. For each $u_i \in S_u^1$, add an edge between t_e^u and u_i of weight $\overline{w}_{t_e^u u_i} = w_{uv}/2$ and cost 0. For each $u_i \in S_u^2$, add an edge between t_e^u and u_i of weight $\overline{w}_{t_e^u u_i} = w_{uv}/2$ and cost c_u. (Add similar edges for all $v_i \in S_v$.) Additionally, join t_e^u and t_e^v with an edge of weight $\overline{w}_{t_e^u t_e^v} = w_{uv}/2 + \delta$ and cost 0 ($\delta > 0$ to be set later). This construction is illustrated in Figure 6.1. The total budget C remains as before. Let $\overline{w}_{max}$ be the maximum weight of an edge in $\overline{\mathscr{I}'}$. Note that

$$\overline{w}_{max} \leq w'_{max} \quad \text{for} \quad \delta \leq w_{min}/2. \tag{6.1.2}$$

Lemma 6.1.1. *Let M be an optimal solution for $\overline{\mathscr{I}'}$ and let $e = \{u, v\} \in E'$ be any edge. Then either the edge $t_e^u t_e^v$ is chosen in M or edges $u_i t_e^u$ and $v_j t_e^v$ are chosen in M, for some $u_i \in S_u$, $v_j \in S_v$.*

Proof. Assume $t_e^u t_e^v$ is not chosen in M. Since the cost of this edge is zero, it must have been not chosen because one of t_e^u and t_e^v is covered. Assume without loss of generality that t_e^u is covered. If t_e^v is uncovered, then we can select edge $t_e^u t_e^v$ instead of the edge covering t_e^u and increase the weight of M by δ, a contradiction. Therefore both vertices are covered. Since $t_e^u t_e^v$ is not in M, edges $u_i t_e^u$ and $v_j t_e^v$ are in M, for some $u_i \in S_u$, $v_j \in S_v$. □

In fact, since we can always make local improvements as in the above lemma, we will assume without loss of generality that M has this property for each $e \in E'$.

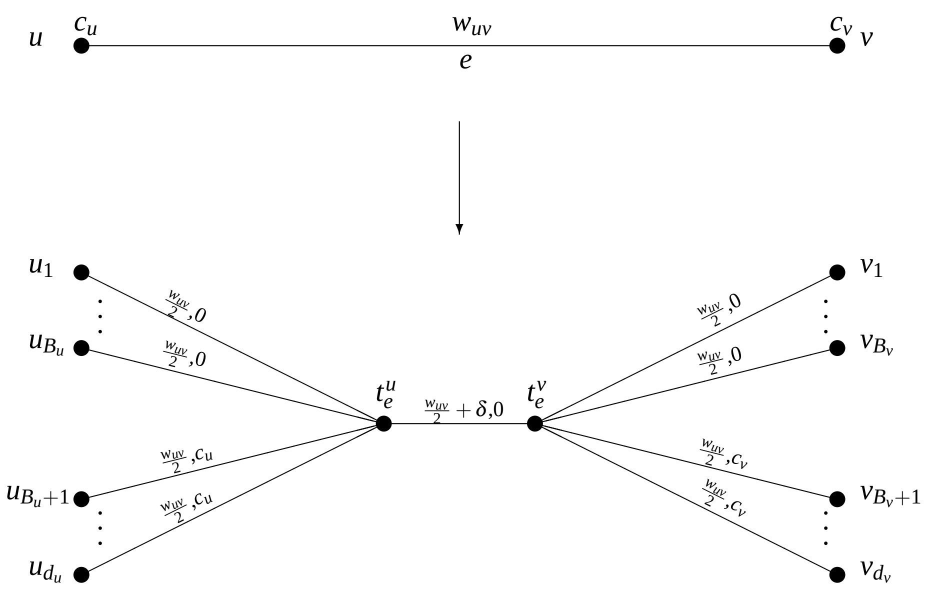

Figure 6.1: Construction of the budgeted matching instance

Given such an M, define $\hat{M}$ to be the set $\{uv \in E' : u_i t_e^u$ and $v_j t_e^v$ are chosen in M, for some $u_i \in S_u$, $v_j \in S_v\}$. Thus, we can see that

$$\overline{w}(M) = w(\hat{M})/2 + w(E')/2 + \delta|E' - \hat{M}|.$$

We will now construct $\tilde{M}_L^*$, a solution in $\overline{\mathscr{I}'}$ that corresponds to the solution M_L^* in $\mathscr{I}'$. In other words, $\hat{\tilde{M}}_L^* = M_L^*$.

For each vertex $v \in V$, arbitrarily order the edges $e \in E'$ incident with v. Let the position of e in this ordering be $v(e)$. Define the solution $\tilde{M}_L^*$ as

$$\{u_{u(e)}t_e^u, v_{v(e)}t_e^v : e = uv \in M_L^*\} \cup \{t_e^u t_e^v : e = uv \in E' \setminus M_L^*\}.$$

After constructing the budgeted matching instance $\overline{\mathscr{I}'}$ from $\mathscr{I}'$ using

$$\delta = \min\left\{\frac{\varepsilon w(M_H^*)}{4|E'|}, \frac{w_{min}}{2}\right\}, \tag{6.1.3}$$

we will use the approximation algorithm for budgeted matching in [1] to find a solution M of weight $\overline{w}(M) \geq OPT - 2\overline{w}_{max} \geq \overline{w}(\tilde{M}_L^*) - 2\overline{w}_{max}$.

Substituting our formula for $\overline{w}$ into the inequality $\overline{w}(M) \geq \overline{w}(\tilde{M}_L^*) - 2\overline{w}_{max}$ gives

$$w(\hat{M})/2 + w(E')/2 + \delta|E' - \hat{M}| \geq w(M_L^*)/2 + w(E')/2 + \delta|E' - M_L^*| - 2\overline{w}_{max}$$

Simplifying, bounding the size of our solutions by $|E'|$, and applying 6.1.2 gives

$$\begin{aligned} w(\hat{M}) &\geq w(M_L^*) - 2\delta(|M_L^*| - |\hat{M}|) - 4\overline{w}_{max} \\ &\geq w(M_L^*) - 2\delta|E'| - 4w'_{max} \qquad (6.1.4) \end{aligned}$$

Finally, let $M := M_H^* \cup \hat{M}$. Then

$$\begin{aligned} w(M) &= w(M_H^*) + w(\hat{M}) \\ &\geq w(M_H^*) + w(M_L^*) - 2\delta|E'| - 4w'_{max} && \text{(by (6.1.4))} \\ &\geq w(M^*) - \varepsilon w(M_H^*)/2 - \varepsilon w(M_H^*)/2 && \text{(by (6.1.3),(6.1.1))} \\ &= w(M^*) - \varepsilon w(M_H^*) \\ &\geq (1-\varepsilon)w(M^*) \end{aligned}$$

This entire algorithm has running time $O(m^{p+O(1)}) = O(m^{O(1/\varepsilon)})$, and is thus a PTAS.

It should be noted that, after using similar filtering techniques and considering the same budgeted matching instance, an FPTAS can be obtained that violates the budget constraint by at most a $(1+\varepsilon)$-factor using the Pareto set methodology of [19] discussed in Section 1.3.1. Finding an FPTAS (with no budget violation) or determining that none exists is an open problem.

6.1.3 Generalizations of matching augmentation

One simple generalization is replacing the constant cost functions $c \in \mathbb{R}^V$ by non-decreasing cost functions. By simply adding different costs to the edges incident to S_v^2 $\forall v \in V$, a non-decreasing cost function can be implemented. Note that if c is not non-decreasing, an optimal solution to the budgeted matching instance will simply use the cheapest edges available and ignore the ordering of the cost function.

A second generalization is to have the cost of additional capacity on a vertex be dependent on which incident edge is being added. Equivalently, we can consider the edges having two costs each, one for each endpoint. This allows us to have different costs to connect different edges to a vertex. Specifically, rather than being given $c \in \mathbb{Q}_+^V$, we will have $c \in \mathbb{Q}_+^{E\times\{1,2\}}$; for each edge $e = \{u,v\}$, we will have two costs, c_e^u and c_e^v, as shown in Figure 6.2.

The PTAS for this generalized version is almost identical to the PTAS for the original. Clearly, in the construction of the budgeted matching instance, the cost of

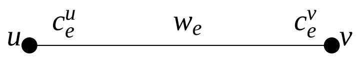

Figure 6.2: Vertex-dependent edge costs in matching augmentation

an edge $t_e^v v_i$, $v_i \in S_v^2$ will now be the cost of connecting edge e to vertex v. The only non-trivial change reflects the fact that in this generalization, it is important which edges pay a connection cost and which are connected using the original capacity available, whereas before, this distinction was not necessary. In the case where additional capacity is purchased at a vertex, it is now desirable to use the original capacity to connect the expensive edges and purchase new capacity to connect the inexpensive edges.

To solve this problem, for each edge that we guess is in M_H^* during the filtering step, we also guess if neither, one, or both of its endpoints are being paid for by vertex capacities (rather than by the budget). Depending on the guess that we make, we will either decrease the available capacity at vertices or decrease the remaining budget available. This makes the running time for the algorithm $O((4m)^{p+O(1)})$ rather than $O(m^{p+O(1)})$, since there are four different ways to connect an edge (a binary decision at each endpoint), so we still have a PTAS.

An interesting related problem is edge-cost matching augmentation, where the costs depend exclusively on the edges, and entire edges must be purchased, not simply the endpoints. The reduction given here is not strong enough to incorporate this case. If the graph is bipartite, this edge-cost case is solvable through techniques discussed later in the chapter; the general case remains open.

6.1.4 Special case: one-sided bipartite matching augmentation with uniform costs

Consider an instance (G, w, c, C) of matching augmentation where $G = (V_1 \cup V_2, E)$ is bipartite, $c_v = \infty \ \forall v \in V_1$ and $c_v = 1 \ \forall v \in V_2$. This encodes the budget constraint that we are permitted to purchase at most C units of additional vertex capacity over one side of the bipartition (and none on the other side). The linear programming relaxation of this problem is as follows:

$$\begin{aligned}
\max \quad & \sum_{e \in E} w_e x_e && (P_{bip}) \\
\text{s.t.} \quad & \sum_{e \in \delta(v)} x_e \leq B_v \quad \forall v \in V_1 && (1) \\
& \sum_{e \in \delta(v)} x_e \leq B_v + y_v \quad \forall v \in V_2 && (2) \\
& \sum_{v \in V_2} y_v \leq C && (3) \\
& x_e \leq 1 \quad \forall e \in E && (4) \\
& x, y \geq 0
\end{aligned}$$

Lemma 6.1.2. *The constraint matrix A of the above linear program is totally unimodular*

Proof. Let M be the submatrix with rows corresponding to all constraints of (1) and (2) and columns corresponding to the x variables. Note that M is totally unimodular as it is simply the vertex-edge incidence matrix of a bipartite graph.

Let A' be any square submatrix of A. If A' has a row corresponding to a constraint of (4), then this row has either zero or one entry of '1'. If the entire row is zeroes, the determinant is zero. If there is exactly one '1', we can calculate the determinant through expansion along this row and proceed inductively.

So assume the rows of A' consist only of constraints from (1),(2), and (3). If there is a column of A' that contains exactly zero or one nonzero entry, we either have $det(A') = 0$ or we can expand along this column and proceed inductively, respectively.

Otherwise, each column has exactly two nonzeros. Consider the columns corresponding to the x variables. Since M is the vertex-edge incidence matrix of a bipartite graph, the rows corresponding to (1) minus the rows corresponding to (2) give the all-zero vector. Since every column of A' has two nonzero entries, the sums of the rows of A' corresponding to (1) and (2) are equal. Looking at the columns of A' corresponding to the y variables, clearly the sum of the rows corresponding to (2) equal the row (3). (Constraint (3) must be present or there would not be two nonzeros in the y-columns). Putting these two together we see that (3) plus the rows of (1) equals the rows of (2), implying that $det(A') = 0$. □

Thus, we can simply solve (P_{bip}) and get an integral solution. This implies that this special case is polynomial-time solvable. This special case is still relevant to Zip, since the structure of G in this case is exactly the structure of their problem;

we have a bipartite graph, and the only purchases that can be made are one side of the bipartition (the side corresponding the supply of DVDs). Although it is slightly inaccurate to set the cost of every DVD to be equal, it is not completely unreasonable.

We may also model this problem as a maximum weight *st*-flow problem. This will serve as another proof that the constraint matrix is totally unimodular (by Remark 1.7.8) and give an additional algorithm for solving the problem efficiently.

Consider the following construction:

- Create directed graph D from bipartite G by directing every edge e of G from V_1 to V_2, with weight w_e and capacity 1.
- Create a source s and an arc (s,v) of weight 0 and capacity B_v for all $v \in V_1$.
- Create a sink t and an arc (v,t) of weight 0 and capacity B_v for all $v \in V_2$.
- Create a dummy node d, an arc (v,d) of weight 0 and infinite capacity for all $v \in V_2$, and arc (d,t) of weight 0 and capacity C.

The flow on arc (v_1,v_2) in the maximum weight *st*-flow instance corresponds to the value of the variable $x_{v_1v_2}$ in the linear program, and the flow on arc (v_2,d) corresponds to the value of y_{v_2}. Note that at most B_v edges incident to $v \in V_1$ may be chosen, as at most B_v flow can enter (and thus exit) node v in the flow instance. The same is true of vertices $v \in V_2$, except that additional flow can be routed along the arc (v,d). However, at most C units can be routed in this manner over the entire instance, forcing the solution to respect the budget constraint.

We now move away from matching and instead focus on similar types of problems involving matroids. We first present the direct analogue to matching augmentation, and then the related Matroid knapsack problem. We give PTASes for both this problem and a generalization, Matroid intersection knapsack.

6.2 Matroid augmentation

Recall the matroid augmentation problem:

Given a matroid $\mathscr{M} = (S,\mathscr{I})$, $A \subseteq S$, weights $w \in \mathbb{Q}^S$, costs $c \in \mathbb{Q}^S_+$, and budget $C \in \mathbb{Q}$, find $A' \subseteq S$ such that $c(A') \leq C$ and the weight of the max-weight independent set $I \subseteq (A \cup A')$ is maximized.

Notice that this is in some ways the direct analogue of edge-cost matching augmentation for matroids; in the matching version, we wanted to increase the value of the optimal solution in our instance by purchasing additional edges. Consider $\mathscr{M}$ the graphic matroid for $G=(V,E)$. This instance, phrased in terms of graphs, is given a graph $G=(V,E)$ and a subset $A\subseteq E$ of the edges, select additional edges $A'\subseteq E$ of cost $c(A')\leq C$ such that the resulting network has an acyclic subgraph of maximum weight possible. A practical application of this could be the refurbishing of an existing service network; given a certain budget, we want to add to existing infrastructure to increase service by as much as possible.

The matroid augmentation problem can be formulated as the following integer program:

$$\begin{aligned} \max \quad & w^T x \\ \text{s.t.} \quad & x(T)\leq r(T) \quad \forall T\subseteq S \\ & x_e \leq y_e \quad \forall e\in S\setminus A \\ & c^T y \leq C \\ & x,y\in \mathbb{Z}_+^S \end{aligned} \tag{IP_1}$$

The following integer program is equivalent:

$$\begin{aligned} \max \quad & w^T x \\ \text{s.t.} \quad & x(T)\leq r(T) \quad \forall T\subseteq S \\ & c^T x \leq C \\ & x\in \mathbb{Z}_+^S \end{aligned} \tag{IP_2}$$

where we set $c_e=0\ \forall e\in A$ (and unchanged for all $e\notin A$). Thus, the matroid augmentation problem can be reduced to the budgeted independent set problem. The reverse reduction is straightforward; if we set $A=\emptyset$, then solving matroid augmentation solves budgeted independent set. Since solving budgeted independent set would solve budgeted spanning tree (using the graphic matroid), and budgeted spanning tree is NP-hard [21], so is matroid augmentation. However, we easily get a PTAS for matroid augmentation by using the algorithm for budgeted matroid intersection in [1] (the result in [1] for budgeted matroid intersection clearly also applies to budgeted independent set).

6.3 Matroid knapsack

Recall the matroid knapsack problem:

Given a matroid $\mathscr{M} = (S, \mathscr{I})$, weights $w \in \mathbb{Q}^S$, costs $c \in \mathbb{Q}^S_+$, and a budget $C \in \mathbb{Q}$, find $A' \subseteq A \subseteq S$ such that $c(A') \leq C$, $A \setminus A' \in \mathscr{I}$, and $w(A)$ is maximized.

We can formulate the matroid basis knapsack problem as the following integer program:

$$\begin{array}{rll} \max & w^T x & (IP_{MKn}) \\ \text{s.t.} & x(T) \leq r(T) + y(T) \quad \forall T \subseteq S & \\ & c^T y \leq C & \\ & x_e \leq 1 \quad \forall e \in S & \\ & x, y \in \mathbb{Z}^S_+ & \end{array}$$

Notice that there is no reason to ever have $y_e > x_e$ for any $e \in S$. Thus, we can redefine our x variables to be $x - y$ in the above integer program, giving us a more convenient representation:

$$\begin{array}{rll} \max & w^T(x+y) & (IP'_{MKn}) \\ \text{s.t.} & x(T) \leq r(T) \quad \forall T \subseteq S & \\ & c^T y \leq C & \\ & x_e + y_e \leq 1 \quad \forall e \in S & \\ & x, y \in \mathbb{Z}^S_+ & \end{array}$$

This new integer program motivates the reason for calling this problem matroid knapsack; in any locally optimal solution, we choose some $A \in \mathscr{I}$ (at no cost) by setting the values of the x variables and then solve the knapsack problem on the remaining elements. The solution to this knapsack problem is represented by the y variables.

We will once again consider the application of this problem obtained by using the graphic matroid of a connected graph $G = (V, E)$. The problem then becomes the problem of selecting any spanning tree of G at zero cost, and then buying additional edges of the graph subject to a budget. We wish to obtain a network of maximum weight. An application of this problem would be the use by a company of a communication network run by some central authority. The company may be allowed access by the central authority to everything in the network for free as a start-up (i.e. a spanning tree at zero cost), and wishes to gain additional service or connectivity by purchasing access to additional components.

Before considering algorithms for the matroid knapsack problem, we will consider the problem under the uniform cost function.

6.3.1 Special case: uniform costs

Consider the case where $c_e = 1 \; \forall e \in S$. In other words, we can select any C elements to add to our independent set.

Claim 6.3.1. *The feasible solutions of (IP'_{MKn}) in the uniform costs case correspond to the independent sets of a matroid.*

Proof. The possible choices of y of at most C elements form the independent sets of a uniform matroid on the ground set S', where we define S' as a copy of S. In other words, for each $e \in S$, there is $e' \in S'$, and vice versa. Combining this uniform matroid with our original matroid $\mathscr{M}$ by Matroid Union, we see that all feasible solutions to (IP'_{MKn}) are independent sets of this union matroid on ground set $S \cup S'$.

Conversely, given an independent set $I \cup I'$ of the matroid formed by applying matroid union to $\mathscr{M}$ and the uniform matroid, we select x variables according to the elements of I and the y variables according to the elements of I'. $\square$

Since the solutions are simply independent sets of a matroid, we can obtain the optimal solution in polynomial time (by Corollary 1.6.6).

6.3.2 Algorithms for matroid knapsack

We will now consider matroid knapsack for an arbitrary matroid. First, notice that the problem is NP-hard. This can be seen easily by considering the matroid whose only independent set is the empty set. In this case, the problem is exactly knapsack, and thus weakly NP-hard.

First, we show that the problem can be reduced to budgeted matroid intersection, which has a PTAS [1]. We then give a pseudopolynomial-time dynamic programming algorithm, which will lead to an FPTAS.

Since (IP'_{MKn}) already contains the rank constraints for one matroid and a budget constraint, if we can show that the set $x_e + y_e \leq 1 \; \forall e \in S$ defines the independent sets of some matroid, we will be done (note that we will define the cost of selecting x_e as 0 towards the budget).

Claim 6.3.2. *$\mathscr{M}' = (S', \mathscr{I}')$ is a matroid, where $S' = \{e_1, e_2 : e \in S\}$, $\mathscr{I}' = \{A \subseteq S' : \nexists e \in S \text{ s.t. } e_1, e_2 \in A\}$.*

Proof. For any element $e \in S$, consider the matroid $\mathscr{M}_e = (\{e_1, e_2\}, \{\{e_1\}, \{e_2\}, \emptyset\})$. Since $\mathscr{M}_e$ is a matroid for every $e \in S$, $\mathscr{M}'$ is a matroid by Matroid Union. $\square$

Setting $x_e = 1$ if $e \in A_1$ and $y_e = 1$ if $e \in A_2$ $\forall e \in S$, we see that $x_e + y_e \leq 1$ $\forall e \in S \Leftrightarrow (A_1, A_2) \in \mathscr{I}$. Thus, (IP'_{MKn}) is the intersection of two matroids along with a budget constraint, and thus has a PTAS by [1].

We may hope to do better than a PTAS. This PTAS exists for any two general matroids; one of our matroids is extremely structured and quite simple. We exploit this fact to get a dynamic programming algorithm to solve Matroid knapsack.

In our dynamic programming solution, we will solve a series of subproblems. In the subproblem denoted by $[i,W]$, we find a solution to the Matroid knapsack problem on the matroid M restricted to the ground set $\{1,..,i\}$, where $\{1,..,i\}$ are the i heaviest elements of S. Additionally, our solution to $[i,W]$ must be the solution of minimum cost that has weight of at least W. If there are several such solutions, we will require the one with largest weight.

We will solve the problem $[i+1,W]$ by modifying the optimal solutions of one of the problems $[i,W]$ and $[i,W-w_{i+1}]$; using the first will correspond to element $i+1$ not being included in the solution to $[i+1,W]$, whereas using the second will correspond to adding element $i+1$.

Consider the dynamic programming algorithm given by Algorithm 5.

Algorithm 5 Dynamic programming algorithm for Matroid Knapsack

```
1: Order the elements of $S$ by decreasing weight (breaking ties lexicographically).
2: Define $OPT[i,W]$ to be the min-cost solution using elements $\{1,..,i\}$ that has
   weight of at least $W$. If there are many such solutions, select the one of
   largest weight (break further ties by selecting the solution whose largest in-
   dex is smallest).
3: Calculate $OPT[i+1,W]$ as follows:
      • Let $(X,Y) = OPT[i, W - w_{i+1}]$.
      • Let $f$ be the minimum cost element in the circuit $C$ of $X \cup e_{i+1}$ (if $X \cup$
        $e_{i+1}$ is independent, set $f$ to be null).
      • Set $OPT[i+1,W]$ to be either $OPT[i,W]$ or $(X \cup e_{i+1} \setminus f, Y \cup f)$,
        whichever has smaller cost.
```

We will first prove that the above recursive step is correct.

First, notice that the X component of the solution $OPT[i,W]$ is always a basis of $\{e_1,..,e_i\}$ (otherwise, we could add components to X to either increase the weight or decrease the cost, depending on whether the elements added come from Y). If $e_{i+1} \notin OPT[i+1,W]$, then $OPT[i+1,W] = OPT[i,W]$.

Lemma 6.3.3. *If $e_{i+1} \in OPT[i+1,W]$, then $OPT[i+1,W] = (X \cup e_{i+1} \setminus f, Y \cup f)$,* *where* $(X,Y) = OPT[i, W - w_{i+1}]$.

Proof. Assume, for a contradiction, that $OPT[i+1,W] = (X', Y')$ with $c(Y') < c(Y) + c_f$.

If $e_{i+1} \in Y'$, consider $(X', Y' \setminus e_{i+1})$. This solution has weight at least $W - w_{i+1}$. Its cost is

$$c(Y') - c_{i+1} < c(Y) + c_f - c_{i+1} \leq c(Y)$$

where the last inequality follows because f is either null or the minimum cost element in a circuit that contains e_{i+1}, and thus has lower cost. This new solution is of lower cost than (X,Y), a contradiction.

If $e_{i+1} \in X'$, consider the set $D := \{e \in S \setminus X' : X' \cup e \notin \mathscr{I}, X' \cup e \setminus e_{i+1} \in \mathscr{I}\}$. In other words, D is the set of elements that form circuits involving e_{i+1} in X'.

Claim 6.3.4. *If $X \cup e_{i+1}$ contains a circuit C, then $C \cap D \neq \emptyset$*

Proof. Since C is a circuit, $C \setminus e_{i+1}$ is independent in $X' \cup C$. Thus, we can extend $C \setminus e_{i+1}$ to a basis B of $X' \cup C$. Since X' is also a basis of $X' \cup C$, $|B| = |X'|$. This implies that B is also basis of $\{e_1, .., e_{i+1}\}$, since it has the same size as X'. Thus, we can extend $X' \setminus e_{i+1}$ to a basis by adding an element e' of $B \setminus X'$, which is just C. So we have $e' \in C \cap D$, as desired. □

Note that if $X \cup e_{i+1}$ is independent, then $OPT[i+1,W] = (X \cup e_{i+1}, Y)$.

Let f' be the minimum cost edge in $C \cap D$. Note that $f' \in Y'$, as otherwise, $(X' \cup f' \setminus e_{i+1}, Y')$ is a better solution than (X', Y') (since the elements are in decreasing order by weight, and further ties are broken by highest-indexed element in the solution). Consider the solution $(X' \cup f' \setminus e_{i+1}, Y' \setminus f')$. This solution has weight at least $W - w_{i+1}$. Its cost is

$$c(Y') - c_{f'} < c(Y) + c_f - c_{f'} \leq c(Y)$$

where the last inequality holds because f is either null or the minimum cost element of the circuit C, which contains f'. This new solution is of lower cost than (X,Y), a contradiction. □

Assume that we have filled out the entire dynamic programming table (this will be explained in more detail later). To recover the optimal solution, we simply scan across the $OPT[n,:]$ row of the table. At each entry, we check the cost of the solution; when we first encounter a solution of cost greater than C, we step back one column to find our optimal solution. If we never find a solution of cost greater than C, then our optimal solution is simply the $OPT[n,W]$ with W as large as possible (eventually, $OPT[n,W]$ will be empty).

Since the weights can be arbitrary non-negative rationals, this dynamic programming table is not polynomial size (or even finite). To get our FPTAS, we must scale and round our weights so that there are only polynomially many values we must consider, as is done in the standard technique for converting the knapsack dynamic programming algorithm to an FPTAS.

Let the optimal solution be I, of weight $w(I)$. Divide each weight w_e by a value f (to be determined later) and round down to obtain new weights $\overline{w}_e$. Notice that for each e,

$$\frac{w_e}{f} - 1 < \overline{w}_e \leq \frac{w_i}{f}$$

Use the dynamic programming algorithm above to compute an optimal solution $\bar{I}$ for this new instance. Consider the value of $\bar{I}$ under the original weights:

$$\begin{aligned}
w(\bar{I}) &= \sum_{e \in \bar{I}} w_e \\
&\geq f \sum_{e \in \bar{I}} \overline{w}_e \\
&\geq f \sum_{e \in I} \overline{w}_e && (\bar{I} \text{ is optimal}) \\
&\geq f \sum_{e \in I} \left(\frac{w_e}{f} - 1 \right) \\
&= w(I) - f|I| \\
&\geq w(I) - fn \\
&\geq w(I) - \varepsilon w(I) && (\text{Assuming } fn \leq \varepsilon OPT)
\end{aligned}$$

Since $OPT \geq \max_e w_e \geq \frac{1}{n} \sum_{e \in S} w_e$, we can choose $f = \frac{\varepsilon}{n^2} \sum_{e \in S} w_e$. Under this choice of f, the size of our dynamic programming table is $O(n \frac{OPT}{f}) = O(\frac{n}{f} \sum_{e \in S}) = O(\frac{n^3}{\varepsilon})$, so the algorithm runs in time polynomial in n and $1/\varepsilon$.

6.4 Matroid intersection knapsack

We now consider a generalization of the matroid knapsack problem. Instead of requiring that our x variables define an independent set of a matroid, we will require that they define an independent set in the intersection of two matroids, $\mathscr{M}_1 = (S, \mathscr{I}_1)$ and $\mathscr{M}_2 = (S, \mathscr{I}_2)$.

The integer program for the matroid intersection knapsack problem is as follows:

$$\begin{aligned} \max \quad & w^T(x+y) && (IP_{MIKn}) \\ \text{s.t.} \quad & x(T) \le r_1(T) \quad \forall T \subseteq S \\ & x(T) \le r_2(T) \quad \forall T \subseteq S \\ & c^T y \le C \\ & x_e + y_e \le 1 \quad \forall e \in S \\ & x, y \in \mathbb{Z}_+^S \end{aligned}$$

Before proceeding, we will show how the matroid intersection knapsack problem can be used to solve the edge-cost matching augmentation problem on bipartite graphs.

6.4.1 Application to edge-cost matching augmentation

Let $G = (U \cup V, E)$ be the bipartite graph in the edge-cost matching augmentation instance. We will use the set of edges E as the ground set of both of our matroids. The y variables in the matroid intersection knapsack instance will represent the set of edges purchased in the edge-cost matching augmentation instance; this will ensure that we stay under budget. We now must define two matroids such that a set of edges is independent in both matroids exactly when it is a feasible b-matching in the edge-cost matching augmentation instance.

Define $\mathscr{M}_U = (E, \mathscr{I}_U)$, where $\mathscr{I}_U = \{E' \subseteq E : \forall u \in U, |E' \cap \delta(u)| \le B_u\}$, where B_u is the vertex capacity of vertex u specified in the edge-cost matching augmentation instance. Define $\mathscr{M}_V$ similarly.

Lemma 6.4.1. *Solutions to edge-cost matching augmentation on $G = (U \cup V, E)$ and the matroid intersection knapsack instance using $\mathscr{M}_U$ and $\mathscr{M}_V$ are in one-to-one correspondence.*

Proof. Let E' be any set of edges in $\mathscr{I}_U \cap \mathscr{I}_V$ and $Y \subseteq E \setminus E'$ with $c(Y) \le C$. E' satisfies all vertex capacities, and combined with Y, gives a feasible solution to the edge-cost matching augmentation instance. Conversely, let $M \subseteq E$ be any feasible solution to edge-cost matching augmentation. We can decompose M into $X \dot{\cup} Y$, where X is a feasible B-matching and Y is the set of purchased edges with $c(Y) \le C$. X and Y correspond to an independent set and set of edges of weight at most C, respectively, in the matroid intersection knapsack instance. □

Since the objective value of solutions is preserved in the reduction, any result we can obtain for matroid intersection knapsack will carry over to this special bipartite case of edge-cost matching augmentation.

6.4.2 A PTAS for matroid intersection knapsack

We will give a PTAS for the matroid intersection knapsack problem using techniques similar to those used by [1] in the budgeted matroid intersection algorithm presented in Section 2.1.

First, as in [1], we form a Lagrangian relaxation ($LR(\lambda)$) of the problem by lifting the budget constraint into the objective function.

$$\begin{aligned} \max \quad & w^T(x+y)+\lambda(C-c^Ty) && (LR(\lambda)) \\ \text{s.t.} \quad & x(T) \le r_1(T) \quad \forall T \subseteq S \\ & x(T) \le r_2(T) \quad \forall T \subseteq S \\ & x_e + y_e \le 1 \quad \forall e \in S \\ & x,y \in \mathbb{Z}_+^S \end{aligned}$$

Removing the constant λC and rearranging gives ($LR(\lambda)'$) below.

$$\begin{aligned} \max \quad & w^Tx+(w-\lambda c)^Ty && (LR(\lambda)') \\ \text{s.t.} \quad & x(T) \le r_1(T) \quad \forall T \subseteq S \\ & x(T) \le r_2(T) \quad \forall T \subseteq S \\ & x_e + y_e \le 1 \quad \forall e \in S \\ & x,y \in \mathbb{Z}_+^S && (6.4.1) \end{aligned}$$

Remark 6.4.2. *The Lagrangian weight for the variables x_e and y_e corresponding to an element $e \in S$ in LR(λ)' are different; for $x_e = 1$, $w_\lambda(e) = w_e$.*

Without the budget constraint on the y variables, their only restriction is that y_e cannot be 1 if x_e is 1. Thus, any time we have $x_e = 0$ in an optimal solution, we will be free to choose $y_e = 1$. We will want to do this exactly when y_e has nonnegative Lagrangian weight. Thus, when $x_e = 0$, we will have

$$y_e = 1 \Leftrightarrow w_e \ge \lambda c_e. \qquad (6.4.2)$$

To ease later work, we define some notation.

Definition 6.4.3. *For any set $T \subseteq S$, define $T^+ := \{e \in T : w_e \ge \lambda c_e\}$*

Definition 6.4.4. *Given a solution to any of the above integer programs, define $X := \{e \in S : x_e = 1\}$, $Y := \{e \in S : y_e = 1\}$. We will refer to the solution as (X, Y).*

Under these definitions, we see that (6.4.2) implies that we will always have

$$Y = \{S \setminus X\}^+ \tag{6.4.3}$$

in any optimal solution (X,Y) to $(LR(\lambda)')$. Thus, we can replace y_e by $1-x_e$ if $e \in S^+$, and by 0 otherwise. Making this substitution into $(LR(\lambda)')$, we get:

$$\begin{aligned}
\max \quad & w^T x + \sum_{e\in S^+} (w-\lambda c)^T (1-x_e) && (LR(\lambda)'') \\
\text{s.t.} \quad & x(T) \le r_1(T) \quad \forall T \subseteq S \\
& x(T) \le r_2(T) \quad \forall T \subseteq S \\
& x \in \mathbb{Z}_+^S && (6.4.4)
\end{aligned}$$

Before proceeding, we will simplify the objective function of $(LR(\lambda)'')$.

$$\begin{aligned}
w^T x + \sum_{e\in S^+} (w-\lambda c)^T (1-x_e) &= \sum_{e\in S} w_e x_e + \sum_{e\in S^+} (w_e - \lambda c_e) - \sum_{e\in S^+} (w_e - \lambda c_e) x_e \\
&= \sum_{e\in S^+} (\lambda c_e x_e) + \sum_{e\in S\setminus S^+} (w_e x_e) + R \\
&= \sum_{e\in S} \min\{w_e, \lambda c_e\} x_e + R && (6.4.5)
\end{aligned}$$

Above, $R = \sum_{e\in S^+}(w_e - \lambda c_e)$ is a constant (and can thus be removed from the objective function), and (6.4.5) follows since $\lambda c_e \ge w_e \ \forall e \in S \setminus S^+$ and $w_e \ge \lambda c_e \ \forall e \in S^+$.

What remains is an instance of matroid intersection, with weights

$$w'_e = \min\{w_e, \lambda c_e\},$$

which can be solved efficiently. Note that any solution X to this problem implies a corresponding Y as in 6.4.3.

Remark 6.4.5. *For any solution X to $(LR(\lambda)'')$, $w'(X) = w_\lambda(X, \{S \setminus X\}^+) - R$.*

As in [1], we can use Megiddo's parametric search technique to obtain the optimal Lagrangian multiplier $\lambda \ge 0$ and two solutions (X_1, Y_1) and (X_2, Y_2) with $X_1, X_2 \in \mathscr{I}_1 \cap \mathscr{I}_2$ such that $c(Y_1) \le C \le c(Y_2)$ and (X_1, Y_1) and (X_2, Y_2) are optimal with respect to Lagrangian weights.

As in Section 2.1, we add dummy elements if necessary and truncate the matroids, resulting in X_1 and X_2 being maximum w'-weight common bases of each of the two truncated matroids.

Lemmas 2.1.3 - 2.1.7 from Section 2.1 all hold exactly as in the budgeted matroid intersection algorithm. Each time we execute the algorithm in Lemma 2.1.7 and it returns a third maximum w'-weight common basis A, we will calculate $Y_A = \{S \setminus A\}^+$. We replace X_1 by A if $c(Y_A) \leq C$ and X_2 by A if $c(Y_A) \geq C$. As in section 2.1, this process will end in polynomial time with X_1, X_2 adjacent in the common basis polytope of $\mathscr{M}_1$ and $\mathscr{M}_2$.

If either $c(Y_1) = C$ or $c(Y_2) = C$, we have a feasible solution that is optimal with respect to the original weights (by a proof similar to that of Theorem 1.5.12), so we are done. Assume $c(Y_1) < C < c(Y_2)$. Without loss of generality, assume that $X_1 \setminus X_2 = \{s_1, \ldots, s_r\}$ and $X_2 \setminus X_1 = \{t_1, \ldots, t_r\}$.

Lemma 6.4.6. *Given (X_1, Y_1), (X_2, Y_2) as above, there is a polynomial time algorithm which computes (X', Y') such that $X' \in \mathscr{I}_1 \cap \mathscr{I}_2$, $X' \cap Y' = \emptyset$, $c(Y') \leq C$, and $w(X' + Y') \geq opt - 3w_{max}$.*

Proof. Since X_1 and X_2 are adjacent in the common basis polytope, by Lemma 2.1.5 we have unique perfect matchings $M_1 = \{s_1t_1, \ldots, s_r, t_r\}$ in $ex_{\mathscr{M}_1}(X_1, X_2)$ and $M_2 = \{t_1s_2, t_2, s_3, \ldots, t_r, s_1\}$ in $ex_{\mathscr{M}_2}(X_1, X_2)$, and corresponding cycle $(s_1, t_1, s_2, t_2, \ldots, s_r, t_r)$. Assign edge s_jt_j a weight $\delta_j := w'(t_j) - w'(s_j)$ and all other edges weight 0. Since X_1 and X_2 have the same w'-weight, $\sum_{j=1}^{r} \delta_j = 0$. By the Gasoline Lemma, there exists an edge of the cycle such that all partial sums of the weights around the cycle starting at this edge are non-negative. Without loss of generality, assume s_1t_1 is such an edge.

Find the largest $k \leq r$ such that

$$c(Y_1) + c(\{s_j : 1 \leq j \leq k\}^+) - c(\{t_j : 1 \leq j \leq k, t_j \in Y_1\}) \leq C.$$

In other words, find the largest $k \leq r$ such that the Y we would select if we chose X as $X_1 \setminus \{s_1, \ldots, s_k\} \cup \{t_1, \ldots, t_k\}$ is under budget. Since $c(Y_2) > C$, we have $k < r$, and by construction,

$$c(Y_1) + c(\{s_j : 1 \leq j \leq k+1\}^+) - c(\{t_j : 1 \leq j \leq k+1, t_j \in Y_1\}) > C$$

or equivalently,

$$\begin{aligned} c(Y_1) + c(\{s_j : 1 \leq j \leq k\}^+) - c(\{t_j : 1 \leq j \leq k, t_j \in Y_1\}) > \\ C - c_{s_{k+1}} + c(Y_1 \cap t_{k+1}) \geq C - c_{s_{k+1}}. \end{aligned}$$

Since adding s_{k+1} to Y_1 put us over budget, $s_{k+1} \in S^+$ (since otherwise, we would not include s_{k+1} in Y_1, so the cost would not increase). Since for an element e to be included in S^+, it must satisfy $w_e \geq \lambda c_e$, we must have that $w_{s_{k+1}} \geq \lambda c_{s_{k+1}}$, and thus

$$w_{max} \geq \lambda c_{s_{k+1}} \tag{6.4.6}$$

We now show that the solution $X' = X_1 \setminus \{s_1,...,s_{k+1}\} \cup \{t_1,...,t_k\}, Y' = Y_1 \setminus \{t_1,...,t_k\} \cup \{s_j : 1 \leq j \leq k\}^+$ satisfies the claim.

Clearly $X' \cap Y' = \emptyset$, as $X_1 \cap Y_1 = \emptyset$, and the only elements added to either set are removed from the other. Because of how we chose k,

$$C - c_{s_{k+1}} < c(Y') \leq C. \tag{6.4.7}$$

By the Gasoline Lemma, we have

$$w'(X') \geq w'(X_1) - w'_{s_{k+1}} = w'(X_1) - \min\{w_{s_{k+1}}, \lambda c_{s_{k+1}}\} \geq w'(X_1) - w_{max}.$$

Recall that $w'(X') = w_\lambda(X', \{S \setminus X'\}^+) - R$, where $R = w_\lambda(S^+)$ is a constant. Thus,

$$\begin{aligned} w'(X') \geq w'(X_1) - w_{max} &\Leftrightarrow w_\lambda(X', \{S \setminus X'\}^+) - R \geq w_\lambda(X_1, \{S \setminus X_1\}^+) - R - w_{max} \\ &\Leftrightarrow w_\lambda(X', Y' \cup \{s_{k+1}\}^+) \geq w_\lambda(X_1, Y_1) - w_{max} \\ &\Rightarrow w_\lambda(X', Y') \geq w_\lambda(X_1, Y_1) - 2w_{max} \end{aligned} \tag{6.4.8}$$

As in [1], $X' \cup \{s_{k+1}\} \in \mathscr{I}_1$ and $X' \cup \{s_1\} \in \mathscr{I}_2$, so $X' \in \mathscr{I}_1 \cap \mathscr{I}_2$.

We now must bound the weight of (X', Y'):

$$\begin{aligned} w(X', Y') &= w_\lambda(X', Y') + \lambda c(X', Y') \\ &= w_\lambda(X', Y') + \lambda C - \lambda(C - c(Y')) \\ &> w_\lambda(X', Y') + \lambda C - \lambda c_{s_{k+1}} && \text{(by (6.4.7))} \\ &\geq w_\lambda(X', Y') + \lambda C - w_{max} && \text{(by (6.4.6))} \\ &\geq w_\lambda(X_1, Y_1) + \lambda C - 3w_{max} && \text{(by (6.4.8))} \\ &\geq OPT - 3w_{max} \end{aligned}$$

where the last inequality follows from the fact that for any $\lambda \geq 0$, the optimal solution to the Lagrangian relaxation is an upper bound of the original budgeted problem. □

Finally, we present the PTAS for matroid intersection knapsack. The algorithm will follow a similar structure to that of Theorem 2.1.9.

Theorem 6.4.7. *The matroid intersection knapsack problem admits a PTAS.*

Proof. Let $\varepsilon \in (0,1)$. If the optimum solution contains fewer than $p := \lceil 3/\varepsilon \rceil$ elements, we can guess the optimal solution (X^*, Y^*) by brute force in time $O((2m)^p) = O(m^{O(1/\varepsilon)})$, giving a PTAS. (The 2 appears because for each element that we guess to be in the solution, we must also guess if it is in X^* or Y^*.) Otherwise, we guess the p elements (X_H^*, Y_H^*) of largest weight in (X^*, Y^*). We contract the matroids by the elements we put in X_H^* and delete from the matroids all elements we put in Y_H^* or that are of weight larger than any of the elements in (X_H^*, Y_H^*). We also decrease the budget by $c(Y_H^*)$. The maximum weight of any remaining element is

$$w'_{max} \leq w(X_H^*, Y_H^*)/p \leq \varepsilon w(X_H^*, Y_H^*)/2, \tag{6.4.9}$$

since w'_{max} is at most the weight of the least weight element removed, which is at most the average weight of removed elements. Additionally, (X_L^*, Y_L^*) is an optimum solution for the remaining matroid intersection knapsack instance, where $X_L^* := X^* \setminus X_H^*$ and $Y_L^* := Y^* \setminus Y_H^*$. We compute a solution (X', Y') to this instance using the methodology described this section of weight $w(X', Y') \geq w(X_L^*, Y_L^*) - 3w'_{max}$. We return the solution $(X, Y) = (X_H^* \cup X', Y_H^* \cup Y')$.

The algorithm runs in time $O(2m^{p+O(1)}) = O(m^{O(1/\varepsilon)})$. Finally,

$$\begin{aligned}
w(X,Y) &= w(X_H^*, Y_H^*) + w(X', Y') \\
&\geq w(X_H^*, Y_H^*) + w(X_L^*, Y_L^*) - 3w'_{max} \\
&\geq w(X^*, Y^*) - \varepsilon w(X_H^*, Y_H^*) && \text{(by (6.4.9))} \\
&\geq (1-\varepsilon) w(X^*, Y^*).
\end{aligned}$$

□

Chapter 7

Conclusion

In this thesis, we introduced Zip.ca, a DVD rental company, with two main operational problems to solve: the assignment of DVDs to users and the purchasing of new DVDs. In order to grasp these problems, we developed theoretical problems with capability to model Zip's situation. The result was twofold: we obtained algorithms for both of the two problems we set out to solve and theoretical results to many interesting related problems. The assignment algorithm we constructed made improvements in the quality of the assignments generated (especially with respect to percentage of slots filled and percentage of slots filled with DVDs of top rank). Some of our theoretical results are listed below.

- We give a $\frac{1}{2\frac{w_{max}}{w_{min}}}$-competitive algorithm for the online 0-1 budgeted matching problem, and show that this the best possible result for a wide class of algorithms.

- We give a $(1 - \frac{S+1}{S+e})$-competitive algorithm for the budgeted Adwords auction as the size of the bids and cost get small as compared to the budgets.

- We show that Zip's assignment problem can be modeled exactly by an instance of the matching augmentation problem, and give a PTAS.

- We present a polynomial time solvable special case of matching augmentation that models Zip's assignment problem under the uniform cost assumption.

- We develop three related matroid problems, matroid augmentation, matroid knapsack, and matroid intersection knapsack. An FPTAS by dynamic programming is presented for matroid knapsack and PTASes for the other two are provided.

7.1 Future work

Many open problems were encountered throughout this thesis on both the applied and theoretical side. The most obvious and pressing need for work on the applied side is the development of a full DVD purchasing methodology. Although we have developed an algorithm to find approximate solutions maximizing the impact on the metric of one assignment run, it is unclear at this point how this algorithm should be best used to make long-term purchasing decisions. Over the time period of several weeks or months, a DVD will be shipped out many times, so it makes little sense to purchase one with the sole purpose of filling one slot. As demonstrated by the general theme of this thesis, it is likely the case that hidden behind this problem is an interesting theoretical direction, but more work is needed to unveil it.

One main overarching open problem for much of the theoretical work discussed is obtaining hardness of approximation results. Although an FPTAS was obtained for matroid knapsack, this thesis was only able to discover a PTAS for each of matching augmentation, matroid augmentation, and matroid intersection knapsack. These problems are all indeed NP-hard, but it is not known if they are strongly NP-hard. In other words, the question of finding a PTAS for any of these three problems, or proving that none exists, is currently open.

An additional open question is related to the online 0-1 budgeted matching problem. We showed that the competitive ratio we obtained is the best possible for the class of algorithms that always selects an edge to the matching when there exist edges available to select. The design of an algorithm that occasionally (and almost certainly randomly) declines to pick any edge with the intention of obtaining a better competitive ratio is an interesting future direction.

One final direction for future work is to model Zip's assignment problem as a truly online problem. One shortcoming of the algorithm that we implemented is that we relied on empirical testing to ensure its quality. It would be very interesting to present an online algorithm to solve Zip's assignment problem that provided *daily* performance guarantees, rather than guarantees for every thirty-minute period.

Bibliography

[1] A. Berger, V. Bonifaci, F. Grandoni, G. Schfer. Budgeted matching and budgeted matroid intersection via the gasoline puzzle. In *Proceedings, IPCO*, pp. 273-287, 2008.

[2] F. Barahona, W. Pulleyblank. Exact Arborescences, Matchings and Cycles. *Discrete Applied Mathematics*, 16, pp. 91-99, 1987.

[3] B. Birnbaum, C. Mathieu. On-line Matching Made Simple. *ACM SIGACT News*, v.39 n.1, March 2008.

[4] N. Buchbinder. *Designing Competitive Online Algorithms via a Primal-Dual Approach*. PhD thesis, Technion - Israel Institute of Technology, 2008.

[5] W. Cook, W. Cunningham, W. Pulleyblank, A. Schrijver. *Combinatorial Optimization*. Wiley, 1998.

[6] M. Garey, D. Johnson. *Computers and Intractability: a Guide to the Theory of NP-completeness*. Freeman, San Francisco, 1979.

[7] R. Dilworth. A decomposition theorem for partially ordered sets. *Annals of Mathematics* (2)51 pp. 161-166, 1950.

[8] J. Edmonds. Maximum matching and a polyhedron with 0,1-vertices. *Journal of Research of the National Bureau of Standards*, 69, pp. 449-467, 1965.

[9] J. Edmonds. Matroids, submodular functions, and certain polyhedra. In *Combinatorial Structures and Their Applications*, Gordon and Breach, New York, pp. 69-87, 1970.

[10] L. Ford, D. Fulkerson. *Flows in Networks*. Princeton University Press, Princeton, New Jersey, 1962.

[11] M.X. Goemans. Lecture notes on Topics in Combinatorial Optimization, Lecture 13. Massachusetts Institute of Technology Lecture Notes, 2004.

[12] A. Hoffman, J. Kruskal. Integral boundary points of convex polyhedra. In *Linear Inequalities and Related Systems*, pp. 223-246, 1956.

[13] O. Ibarra, C. Kim. Fast approximation algorithms for the knapsack and sum of subset problems. *Journal of the ACM*, 22, pp. 463-468, 1975.

[14] R.M. Karp, U. Vazirani, V. Vazirani. An optimal algorithm for on-line bipartite matching. In *STOC '90: Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pp. 352-358, 1990.

[15] J. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. In *Proceedings of the American Mathematical Society*, pp. 48-50, 1956.

[16] N. Megiddo. Combinatorial optimization with rational objective functions. Mathematics of Operations Research 4(4), pp. 414-424, 1979.

[17] A. Mehta, A. Saberi, U. Vazirani, V. Vazirani. Adwords and generalized online matching. *J. ACM*, 54(5):22, 2007.

[18] K. Mulmuley, U. Vazirani, V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica* 7(1), pp. 105-113, 1987.

[19] C.H. Papadimitriou, M. Yannakakis. On the approximability of trade-offs and optimal access of Web sources. In *Proceedings, 41st Annual Symposium on Foundations of Computer Science*, p.86. November 2000.

[20] R. Rado. A note on independence functions. In *Proceedings of the London Mathematical Society*, 7, pp 300-320, 1957.

[21] R. Ravi, M.X. Goemans. The constrained minimum spanning tree problem. In *Proceedings, Scandinavian Workshop on Algorithmic Theory*, July 1996.

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