Approximation Algorithms for Single-Minded Pricing and Unique Coverage on Graphs and Geometric Objects

by

Seyed Sina Khankhajeh

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

University of Alberta

Abstract

In this thesis we study the Single-Minded Pricing, Unique Coverage, and Uniform-Budget Single-Minded Pricing problems on graphs and on geometric objects.

In Single Minded Pricing, we are given a set U of items and a collection D of subsets of U, called demands. For each demand $d \in D$, we are also given a budget $B(d) \in \mathbb{R}^+$. If each item $u \in U$ is priced at $p(u) \in \mathbb{R}^+$, the revenue from demand $d \in D$ will be $\sum_{u \in d} p(u)$ if $\sum_{u \in d} p(u) \leq B(d)$ and 0 otherwise. We want to assign prices to the items in order to maximize the total revenue. Specifically, we want to assign prices $p : U \mapsto \mathbb{R}^+$ to the items to maximize $\sum_{d \in D : B(d) \geq \sum_{u \in d} p(u)} \sum_{u \in d} p(u)$. Uniform-Budget Single-Minded Pricing is a special case of Single-Minded Pricing where the budgets for all the demands are equal. In Unique Coverage, given a set U of elements and a collection D of subsets of U, we want to find a sub-collection $D' \subseteq D$ to maximize the number of the elements that are uniquely covered, *i.e.,* appear in exactly one set of D'.

In Chapter 2, we consider the Single-Minded Pricing problem on graphs and on geometric objects. In Single-Minded Pricing on graphs, the items are the edges of the given graph and each demand is a path in the graph demanding the edges in that path. In Single-Minded Pricing on geometric objects, each item is a point on a two-dimensional plane and each demand is represented by a geometric object demanding the points within the geometric object. We introduce approximation algorithms for Single-Minded Pricing on geometric objects such as unit squares (axis parallel squares with size of 1), squares (axis parallel), line rectangles (axis parallel rectangles that their bottom sides lie on the same line) and rectangles (axis parallel). Then we prove that Single-Minded Pricing on graphs that have bounded pathwidth is as hard as Unique Coverage in the general setting.

In Chapter 3, we study Unique Coverage on graphs and on geometric objects. In Unique Coverage on graphs, the elements are the edges of the given graph and each subset $d \in D$ of edges, forms a path in the graph. In Unique Coverage on geometric objects, each element is a point on a two-dimensional plane and each subset $d \in D$ is represented by a geometric object and consists of all the points within the geometric object. We present a dynamic programming algorithm for the Unique Coverage problem on corner rectangles (axis parallel rectangles that have the same left-bottom corner). We also introduce a dynamic programming algorithm for the Unique Coverage problem on

trees where the subsets in D form upward paths.

In Chapter 4, we study Uniform-Budget Single-Minded Pricing on graphs and on geometric objects which are similar to Single-Minded Pricing on graphs and on geometric objects, respectively. We take a look at the relation between Uniform-Budget Single-Minded Pricing and Unique Coverage and the results we can obtain, considering this relation. Finally we prove that Uniform-Budget Single-Minded Pricing on trees where demands are upward paths, can be solved in polynomial time.

Acknowledgements

I would like to thank Mohammad R. Salavatipour, my supervisor, who helped me understand the basics of approximation and randomized algorithms which are very rich and great fields of study in computing science. I am very grateful to be under supervision of a person with such knowledge and wisdom.

Furthermore, I would also like to thank Zachary L. Friggstad, Babak Behsaz, Saber Khakpash and Amir Malekzadeh for thinking with me on some problems and giving some ideas that helped me in writing this thesis.

Table of Contents

List of Figures

Chapter 1

Introduction

Imagine that we are a company or store in the business of selling products to consumers. An important aspect of maximizing the revenue obtained is the pricing of our products: a low price will attract more customers, while a high price generates more revenue per sold item. How, then, should we choose the prices optimally? For example, suppose that the customers want to buy bandwidth along the subpaths of a network and are willing to pay up to some amount for the bandwidth, how should one price the bandwidth along the links so as to maximize the revenue? This is the flavor of the problems studied in this thesis.

1.1 Problems Considered

In this thesis we study three problems: Single-Minded Pricing, Unique Coverage, and Uniform-Budget Single-Minded Pricing. Although Uniform-Budget Single-Minded Pricing is a special case of Single-Minded Pricing, because of its close relation to Unique Coverage we study it separately.

In Single Minded Pricing (SMP), we are given a set U of items and a collection D of subsets of U, called demands. For each demand $d \in D$, we are also given a budget $B(d) \in \mathbb{R}^+$. If each item $u \in U$ is priced at $p(u) \in \mathbb{R}^+$, the revenue from demand $d \in D$ will be $\sum_{u \in d} p(u)$ if $\sum_{u \in d} p(u) \leq B(d)$, and 0 otherwise. We want to assign prices to the items in order to maximize the total revenue. Specifically, we want to assign prices $p : U \mapsto \mathbb{R}^+$ to the items to maximize:

$$\sum_{d \in D : B(d) \geq \sum_{u \in d} p(u)} \quad \sum_{u \in d} p(u)$$

Single-Minded Pricing can also be studied in different settings such as on graphs or on geometric objects. In SMP on graphs, the items are the edges of a given graph and each demand is a path in the graph demanding the edges of that path. In SMP on geometric objects, each item is a point

on a two-dimensional plane and each demand is represented by a geometric object demanding the points within that geometric object.

In Unique Coverage (UC), given a set U of elements and a collection D of subsets of U, we want to find a sub-collection $D' \subseteq D$ to maximize the number of the elements that are uniquely covered, *i.e.,* appear in exactly one set of D'.

We also study Unique Coverage on graphs and on geometric objects. In UC on graphs, we are given a set of paths in a graph and we want to choose a subset of the edges of the graph to maximize the number of the paths that are uniquely covered. In UC on geometric objects, we are given a set of points and a set of geometric objects on the plane and we want to choose a subset of the points so as to maximize the number of uniquely covered objects, *i.e.,* objects with only one chosen point within.

Uniform-Budget Single-Minded Pricing (UBSMP) is a special case of Single-Minded Pricing where the budgets for all the demands are equal. Finally, we study UBSMP on graphs and on geometric objects which are similar to SMP on graphs and on geometric objects, respectively.

1.2 Related Works

Guruswami *et al.* [33] introduced Envy-Free Pricing and different variants of it, such as limited or unlimited supply versions of Single-Minded Pricing and Unit-Demand Pricing (See Section 1.3.3 for formal definitions). They managed to formulate item pricing to maximize seller's revenue where later researchers used mainly their formulation. However, before them, there were many studies on item pricing as well (For example, see [1]).

The Envy-Free Pricing problem has been studied in general and also for many special cases after Guruswami et.al's seminal paper [33]. Hartline *et al.* [35] introduced some near-optimal algorithms for some variations of Envy-Free Pricing such as unlimited and limited supply versions of Unit-Demand Pricing and unlimited supply version of Single-Minded Pricing in near-linear time. Briest [6] introduced some hardness results for Envy-Free Pricing in the general setting. Chen *et al.* [16] later presented an interesting polynomial time exact algorithm for the Envy-Free Pricing problem when there is metric substitutability property among the items. There are also many results for Envy-Free Pricing on lesser-known special cases [28, 14].

Unit-Demand Pricing is a variation of the Envy-Free Pricing problem that has been studied extensively after being introduced by Guruswami *et al.* [33]. Briest [7] introduced some hardness results for the Unit-Demand Pricing problem in his paper. There have been a lot of studies recently about this problem [37, 15, 27].

In the definition of Single-Minded Pricing mentioned in this thesis, it is obvious that we have unlimited supply for each item, since we allocate each item $u \in U$ to all satisfied demands that have

u in their demand set. Therefore, note that we have defined the unlimited supply version of Single-Minded Pricing problem as the default definition of Single-Minded Pricing throughout this thesis. For the less studied limited supply version of Single-Minded Pricing, Cheung *et al.* [17] introduced interesting polynomial-time approximation algorithms.

In Envy-Free Pricing, we assume that the cost of each item is zero and prices assigned are non-negative. A more general case, is the case where pricing below cost is possible. This problem is called the "Loss Leaders" problem and was first introduced by Balcan *et al.* [3]. There are a few papers studying this problem, specially its hardness [4, 45, 49].

Studying pricing problems on graphs is not restricted to Envy-Free Pricing on graphs. There are a lot of independent studies that do not follow the envy-freeness rules [33]. For example, Grigoriev *et al.* [32] considered a pricing problem on graphs which was more general than Envy-Free Pricing on graphs. Another interesting graph pricing problem that has been studied thoroughly is the Stackelberg Network Pricing problem [8, 40, 9].

1.3 Notations and Preliminaries

In this section we introduce some notations and preliminaries to be used throughout the thesis. First and foremost, in the Single-Minded Pricing problem, we will denote the number of items by n and the number of demands by m, independent from the setting of the problem. Also in the Unique Coverage problem, we denote the number of elements by n and the number of subsets in the collection (not necessarily unique) by m (the size of the collection). We continue with some notations and preliminaries related to graph theory, approximation and randomized algorithms, envy-free pricing, linear programming and optimization problems on geometric objects.

1.3.1 Graph Theory

In this thesis we represent a graph G as a pair $(V(G), E(G))$ in which $V(G)$ is the set of vertices (nodes) and $E(G)$ is the set of undirected edges. We use V to denote $V(G)$ and E to denote $E(G)$ when G is clear from the context. We denote each edge $e \in E$ by $\{u, v\}$ to specify that u and v are the endpoints of e. We will assume the given graph is undirected and simple in general which means it does not have parallel edges or loops, unless otherwise specified.

In Singled-Minded Pricing on graphs, we want to assign a price $p(e)$ to each edge $e \in E$. In SMP on graphs, the price of a path is defined as the sum of the prices of its edges, *i.e.,* for a path d, $p(d) = \sum_{e \in d} p(e)$.

Assume we are given a rooted tree T. A vertex v is called an ancestor for vertex u, if v is on the path between u and the root; u is called a descendant of v. A subtree of T rooted at node v, is the tree consisting of v and all of its descendants. A path is called upward if one endpoint is an

ancestor of the other endpoint. In an upward path, the top most node of the path is the endpoint closest to the root and the bottom most node of the path is the endpoint furthest from the root. In SMP on Upward Trees and UBSMP on Upward Trees, the given graph is a rooted tree and the demands are upward paths. Also, in UC on Upward Trees, the given graph is a rooted tree and the subsets of edges in D form upward paths.

1.3.2 Approximation and Randomized Algorithms

An optimization problem is the problem of finding a feasible solution with optimum objective value from a set of feasible solutions. An optimization problem can be defined as a pair (S, f), where S is the set of feasible solutions and $f : S \mapsto \mathbb{R}^+$ is the objective function. If our goal is to minimize the objective value, we say the problem is a minimization problem and if our goal is to maximize the objective value, we say the problem is a maximization problem.

Many optimization problems are NP-hard and it is not possible to find a polynomial time algorithm for them unless P=NP. So we try to approximate the solution. We say an algorithm is an α-approximation algorithm for a minimization problem, if it finds a feasible solution s in polynomial time such that $f(s) \leq \alpha.OPT$, where OPT is the optimum objective value for the instance. Similarly, an α-approximation for a maximization problem is an algorithm that finds a solution s in polynomial time such that $f(s) \geq \frac{OPT}{\alpha}$. Sometimes α is called the approximation ratio, approximation factor, or performance ratio.

According to Garey and Johnson [30], NP-complete problems are divided into weak and strong NP-complete problems. An NP-complete (or NP-hard) problem is weakly NP-complete (or weakly NP-hard), if there is an algorithm for the problem whose running time is polynomial in the dimension of the problem and the magnitudes of the data involved (provided these are given as integers), rather than the base-two logarithms of their magnitudes. Such algorithms are technically exponential functions of their input size and are therefore not considered polynomial. In other words, a weak NP-complete problem is NP-complete if numbers are given in binary, but polynomial time solvable when numbers are given in unary. A problem is said to be strongly NP-complete (NP-complete in the strong sense), if it remains so even when all of its numerical parameters are bounded by a polynomial in the length of the input. A problem is said to be strongly NP-hard if a strongly NP-complete problem has a polynomial reduction to it. In other words, a strong NP-complete problem is still NP-complete even if numbers are given in unary.

A Polynomial Time Approximation Scheme (PTAS) for an optimization problem is a scheme such that for any fixed constant ϵ gives a $(1 + \epsilon)$-approximation algorithm whose running time is polynomial. A Fully Polynomial Time Approximation Scheme (FPTAS) for an optimization problem, is a scheme that for any ϵ gives a $(1 + \epsilon)$-approximation algorithm whose running time is

polynomial in the size of the instance and $1/\epsilon$. If an algorithm runs in time polynomial in the size of the input and $1/\epsilon$, but guarantees a $(1+\epsilon)$-approximation only if the input size is at least c_ϵ, where c_ϵ is a constant only dependent on ϵ, it is called an asymptotic FPTAS or FPTAS^w [25]. Another variant of PTAS is the quasi-polynomial-time approximation scheme or QPTAS. A QPTAS has time complexity $n^{polylog(n)}$ for each fixed $\epsilon > 0$. In this thesis, we say a problem is APX-hard [2] to imply that there is no PTAS for it unless P = NP; in other words, there is a c-hardness factor for it for a fixed constant $c > 1$.

A randomized algorithm is an algorithm that uses a sequence of random bits. Such an algorithm typically uses uniformly random bits as an auxiliary input to guide its behavior, in the hope of achieving a good performance in the "average case" over all possible choices of random bits. Formally, the algorithm's performance will be a random variable determined by the random bits; thus either the running time, or the output (or both) are random variables.

In probability theory, the Chernoff bound gives exponentially decreasing bounds on tail distributions of sums of independent random variables. It is a sharper bound than the known first or second moment based tail bounds such as Markov's inequality or Chebyshev inequality, which only yield power-law bounds on tail decay. Formally, assume $X_1, X_2, \ldots, X_n$ are independent Poisson trials with $Pr[X_i = 1] = p_i$, for $0 < p_i < 1$. Then assuming $X = \sum_{i=1}^{n} X_i$ and $\mu = E[X]$:

- for any $0 < \delta : \ Pr[X \geq (1+\delta)\mu] \leq \left(\dfrac{e^\delta}{(1+\delta)^{(1+\delta)}} \right)^\mu$.
- for any $0 < \delta \leq 1 : \ Pr[X \geq (1+\delta)\mu] \leq e^{-\mu\delta^2/3}$.
- for any $R \geq 6\mu : \ Pr[X \geq R] \geq 2^{-R}$.

Note that these bounds are independent of μ.

The method of conditional expectations (also called the method of conditional probabilities) was first introduced by Erdos and Selfridge [24] and later by Raghavan [46]. This technique is a powerful tool that sometimes can help in derandomizing algorithms. In this technique, rather than making all of our random decisions simultaneously, we make them sequentially. Then, instead of making the decisions by choosing randomly between two alternatives, we evaluate both alternatives according to the conditional expectation of the objective function if we fix the decision (and all preceding ones) but make the remaining ones at random. Then we choose the alternative that optimizes this conditional expectation (For formal definition and to see some examples, refer to [44]).

1.3.3 Envy-Free Pricing

Guruswami *et al.* [33] introduced Envy-Free Pricing and different variants of it. We define them in this section (For more information, see [33]):

- **Envy-Free Pricing**: In Envy-Free Pricing, we are given a set U of distinct items and a number of customers. We have c_u copies of each item $u \in U$. The case in which for all items $u \in U$, $c_u = \infty$, is called the unlimited version of Envy-Free Pricing. The case where we do not have unlimited copy of all items is called the limited version of Envy-Free Pricing. Each customer i has budget $B_i(S)$ for each subset $S \subseteq U$, which is the maximum amount that customer i is willing to pay for subset S. Assume we have a price of $p(u)$ for each item $u \in U$. The utility of subset $S \subseteq U$ for customer i, is defined as $R_i(S) = B_i(S) - \sum_{u \in S} p(u)$. It measures the "joy" at having bought the subset S at the given price for customer i. We want to assign a price of $p(u)$ for each item $u \in U$ and also allocate a subset of items to each customer so that the total revenue is maximized, *i.e.,* the sum of the prices of all sold items is maximized, while each customer is maximally happy with the subset of the items she has received. Specifically, for each customer, the utility of the subset that is allocated to her is maximum over all subsets (See [33] for formal definition of Envy-Free Pricing).

- **Unit-Demand Pricing**: Unit-Demand Pricing is a special case of Envy-Free Pricing where each customer is willing to buy at most one item and for each item, the customer has a budget which is the maximum amount the customer is willing to pay for that item.

- **Single-Minded Pricing**: Single-Minded Pricing is another special case of Envy-Free Pricing where each customer is interested in only one specific subset of items and has a budget for that subset which is the maximum amount the customer is willing to pay to receive that subset of items.

- **The Tollbooth Problem**: We denote the Single-Minded Pricing problem on Trees as the Tollbooth problem. In the Tollbooth problem, the items are the edges on a given tree and each demand (requested by a unique customer) is represented as a path on the tree and the customer demands all the edges on that path [33].

- **The Highway Problem**: The Highway Problem is a special case of the Tollbooth problem where the underlying tree given in the input is a path [33]. A special case of this problem is the Nested Highway problem where for each pair of demands d, d', at least one of the following is true: $d \subseteq d'$ or $d' \subseteq d$ or $d \cap d' = \emptyset$ [10, 3].

1.3.4 Linear Programming

Linear programming (LP) is a problem that can be expressed in the following form:

$$\begin{aligned} \text{Minimize } & \sum_{i=1}^{n} c_i x_i & \\ \text{Subject to } & \sum_{i=1}^{n} A_{ji} x_i \geq b_j & \forall 1 \leq j \leq m, \\ & x_i \geq 0 & \forall 1 \leq i \leq n, \end{aligned}$$

where $x \in \mathbb{R}^n$ represents the vector of variables and $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, and $c \in \mathbb{R}^n$ are fixed known coefficients. The goal is to find a vector x which satisfies the inequalities and minimizes the objective function.

A unimodular matrix M is a square integer matrix having determinant $+1$ or -1. A totally unimodular matrix (TU matrix) is a matrix for which every square non-singular submatrix is unimodular. A totally unimodular matrix need not be square itself. From the definition it follows that any totally unimodular matrix has only 0, $+1$ or -1 entries. Totally unimodular matrices are extremely important in combinatorial optimization since they give a quick way to verify that a linear program is integral (has an integral optimum, when any optimum exists). Specifically, if A is TU and b is integral, then linear programs of forms like $\{\min cx \mid Ax \geq b, x \geq 0\}$ or $\{\max cx \mid Ax \leq b\}$ have integral optimum, for any c [36].

1.3.5 Optimization Problems on Geometric Objects

We study two different optimization problems in different settings such as on different geometric objects. When studied on geometric objects, the problem is defined on a two-dimensional plane. The input to the problem is a set of points on the plane and also a set of geometric objects on the plane. We will consider these objects in the thesis:

- **Rectangle**: Axis-parallel rectangles
- **Line Rectangle**: Axis-parallel rectangles which their lower sides fall on the same line.
- **Corner Rectangle**: Axis-parallel rectangles which have the same left-bottom corner.
- **Squares**: Axis-parallel squares of arbitrary sizes
- **Disk**: Disks of arbitrary sizes
- **Unit Square**: Axis-parallel squares of size 1
- **Unit Disk**: Disks of diameter 1

- **Disks with bounded ply**: A set of disks with arbitrary sizes where the maximum number of disks containing a point, over all the points on the plane, is bounded.

1.4 Outline of Thesis

In this thesis we study Single-Minded Pricing, Unique Coverage, and Uniform-Budget Single-Minded Pricing on graphs and geometric objects.

In Chapter 2, we study Single-Minded Pricing. We introduce approximation algorithms for SMP on geometric objects such as unit squares, squares, line rectangles, and rectangles. Then we prove that SMP on Graphs with Bounded Pathwidth is as hard as Unique Coverage in the general setting.

In Chapter 3, we discuss Unique Coverage. We present a dynamic programming algorithm for Unique Coverage on Corner Rectangles. We also introduce a dynamic programming algorithm for Unique Coverage on Upward Trees.

In Chapter 4, we study Uniform-Budget Single-Minded Pricing. We take a look at the relation between UBSMP and UC and the results that can be obtained, considering this relation. Finally, we prove that UBSMP on Upward Trees can be solved in polynomial time.

In Chapter 5, we present a summary and mention possible future works on the problems studied in this thesis.

Chapter 2

Single-Minded Pricing

Recall that in Single Minded Pricing (SMP), we are given a set U of items and a collection D of subsets of U, called demands. For each demand $d \in D$, we are also given a budget $B(d) \in \mathbb{R}^+$ which is the maximum revenue obtainable from d. If each item $u \in U$ is priced at $p(u) \in \mathbb{R}^+$, the revenue from demand $d \in D$ will be $\sum_{u \in d} p(u)$ if $\sum_{u \in d} p(u) \leq B(d)$ and 0 otherwise. We want to assign prices to the items in order to maximize the total revenue. Specifically, we want to assign prices $p : U \mapsto \mathbb{R}^+$ to the items to maximize:

$$\sum_{d \in D : B(d) \geq \sum_{u \in d} p(u)} \sum_{u \in d} p(u)$$

In this chapter we study Single-Minded Pricing on graphs and on geometric objects. We start with the previous works done in this area in Section 2.1. In Section 2.2 we introduce some basic results for SMP on several geometric objects. Finally, in Section 2.3 we discuss why SMP on general graphs can be very hard.

2.1 Related Works

Single-Minded Pricing was introduced by Guruswami *et al.* [33]. They proved that Single-Minded Pricing is APX-hard even when $|d_i| = 2$ for all $d_i \in D$ ($|d_i|$ is the number of items inside d_i) and the budgets are either 1 or 2. They also provided an $O(\log n + \log m)$-approximation for Single-Minded Pricing. Later, Briest *et al.* [10] improved the upper bound for Single-Minded Pricing on sparse instances to $O(\log l + \log \beta)$ where l is the maximum number of items in a demand and β is the maximum number of demands per item. They also gave an $O(l^2)$-approximation algorithm for Single-Minded Pricing which was the first result independent from the number of demands. This result was later improved by Balcan *et al.* [3] to an $O(l)$-approximation algorithm using a randomized approach.

Single-Minded Pricing is proven to be a very hard problem as well. Briest [7] proved that for every $\delta > 0$ there exists an $\varepsilon > 0$, such that it is R3SAT*($2^{O(n^\delta)}$)-hard to approximate Single-Minded Pricing within $O(n^\varepsilon)$ (See [26] for the definition of the R3SAT-hardness). DTIME(T) denotes the class of problems that can be solved in the deterministic time $O(T)$. Similarly ZPTIME(T) denotes the class of problems that can be solved using a randomized algorithm whose expected running time is $O(T)$. Chalermsook *et al.* [11] proved that Single-Minded Pricing is $\log^{1-\epsilon}(m+n)$-hard to approximate for any constant ϵ, unless NP $\subseteq$ DTIME($n^{\log^\delta n}$), where δ is a constant depending on ϵ. Restricting their attention to hardness depending only on the number of items, they showed that Single-Minded Pricing is $2^{\log^{1-\delta} n}$-hard to approximate for any $\delta > 0$ unless NP $\subseteq$ ZPTIME($n^{log^{\delta'} n}$), where δ' is some constant depending on δ. They also proved that Single-Minded Pricing is $l^{1/2-\epsilon}$-hard to approximate for any constant ϵ (where l is the maximum number of items in a demand).

SMP on geometric objects is studied more recently. The only major previous work in this area is a QPTAS for SMP on Corner Rectangles by Chalermsook *et al.* [12].

On the other hand, SMP on graphs is a well-studied field. Guruswami *et al.* [33] introduced SMP on Trees as "the Tollbooth problem" and SMP on Paths as "the Highway problem". For the Tollbooth problem, Guruswami *et al.* proved that it is still APX-hard but for the special case where the tree is rooted and all the demands start from the root, the problem is solvable in polynomial time [33]. The first significant hardness result for the Highway problem was presented by Briest *et al.* [10] where they proved that the special case of the Highway problem where the paths are nested (the Nested Highway problem) is weakly NP-hard and provided an FPTAS for it. They also proved that the Tollbooth problem stays APX-hard even under very restricted assumptions. Balcan *et al.* [3] independently came up with an FPTAS for the Nested Highway problem. They also managed to improve the approximation ratio of the Highway problem from $O(\log m + \log n)$ to $O(\log n)$. Elbassioni *et al.* [23] gave the first QPTAS for the Highway problem. The approximation ratio of the Tollbooth problem was not improved until Elbassioni *et al.* 's later paper [21] in which they gave an $O(\log n)$-approximation algorithm for the problem and also introduced the Upward Tollbooth problem (The special case of the Tollbooth problem where the demands are upward paths) and came up with a QPTAS for it. They also proved that the Highway problem is strongly NP-hard (and again at [22]). Elbassioni [20] later defined the notion of ϵ-Envy-Free Pricing and introduced a QPTAS for ϵ-Envy-Free Pricing on Paths. Kortsarz *et al.* [43] showed that the gap in revenue between a naturally defined upward instance of the Tollbooth problem and the original instance can be at least $\Omega(\log \log n)$. The best result for the Tollbooth problem so far is presented by Gamzu *et al.* [29] which is an $O(\frac{\log n}{\log \log n})$-approximation algorithm. The best result for the Highway problem finally was presented by Grandoni *et al.* [31] which was a PTAS and closed the complexity status of the problem because of the previous strong NP-hardness result. Their PTAS also extends to the

special case of the Tollbooth problem where the number of leaves is constant.

SMP on general graphs is a very hard problem. Chalermsook *et al.* [11] showed that SMP on graphs is as hard as Unique Coverage in the general setting (Unique Coverage and its hardness will be discussed in the next chapter).

In SMP on graphs, we use the edges of the graph as a representation for the items. Now, assume in Single-Minded Pricing in the general setting, we know that each customer is interested in at most two items. The best way to represent this case with graphs is to consider each item as a vertex in a graph and each customer as an edge (interested in its endpoints) or a vertex in that graph. Balcan *et al.* [3] defined this problem as "Graph Vertex Pricing" and gave a 4-approximation algorithm for it. The APX-hardness that Guruswami *et al.* presented in their paper [33] for the Tollbooth problem, also extends to Graph Vertex Pricing. Khandekar *et al.* [41] improved the hardness result by proving that Graph Vertex Pricing is still APX-hard even when restricted to bipartite graphs. A bipartite graph is a graph whose vertices can be divided into two disjoint sets such that every edge connects a vertex in one set to a vertex in the other set. They also proved that Graph Vertex Pricing is NP-hard to approximate within a factor of 2 assuming the Unique Games Conjecture (See [42] for the definition of the game and the conjecture), and it is NP-hard to approximate within a factor of 17/16 unless P=NP.

A tree decomposition of a graph $G = (V, E)$ is a tree, $T = (I, F)$, with nodes $X_1, \ldots, X_{|I|}$, where each X_i is a subset of V, satisfying the following properties:

- The union of all sets X_i equals V. That is, $\bigcup_{i=1}^{|I|} X_i = V$.
- If X_i and X_j both contain a vertex v, then all nodes X_k of the tree in the (unique) path between X_i and X_j contain v as well. Equivalently, the tree nodes containing vertex v form a connected subtree of T.
- For every edge $\{v, w\}$ in the graph, there is a subset X_i that contains both v and w.

The width of a tree decomposition is the size of its largest set X_i minus one. The treewidth of a graph G is the minimum width among all possible tree decompositions of G.

The genus of a graph is the minimum integer k such that the graph can be drawn without crossing itself on a sphere with k handles (*i.e.,* an oriented surface of genus k) (See [47] for the formal definition and the complexity of finding the genus of a graph).

Chalermsook *et al.* [13] showed that there exists an FPTAS for Graph Vertex Pricing on graphs with bounded treewidth. For Graph Vertex Pricing on bounded genus graphs they also presented a PTAS and showed that Graph Vertex Pricing is NP-hard even on planar graphs. A planar graph is a graph that can be embedded in the plane, *i.e.,* it can be drawn on the plane in such a way that its edges intersect only at their endpoints. So a planar graph has genus 0.

2.2 SMP on Geometric Objects

In this section, we present some approximation results for SMP on geometric objects. All of our results are based on the QPTAS result that was introduced for SMP on Corner Rectangles by Chalermsook *et al.* [12]. We present basic approximation results for SMP on Unit Squares, Squares, Line Rectangles, and Rectangles. Recall that in SMP on Corner Rectangles, the demands are given as rectangles that have the same bottom-left corner, each demanding the points within.

Recall that in SMP on Unit Squares, we have a set of items represented as points on a plane and a set of demands represented as unit squares, each demanding the points within. Similar to SMP in the general setting, we want to assign prices to the items to maximize the revenue which is the sum of the prices of all sold items.

Corollary 2.2.1. *There is a* $(16+\epsilon)$*-approximation algorithm that runs in quasi-polynomial time for SMP on Unit Squares, for any fixed constant* $\epsilon > 0$.

Proof. First, we partition the plane into unit squares of size 1 by horizontal and vertical lines of distance 1 apart (See Figure 2.1 to see an example of the partition) so that each point (item) is not on one of the horizontal or vertical lines (this is easy to do so by shifting the horizontal and vertical lines until there are no points on any of these lines). We denote these unit squares as the generated unit squares to avoid confusion with our demands which are also represented by unit squares.

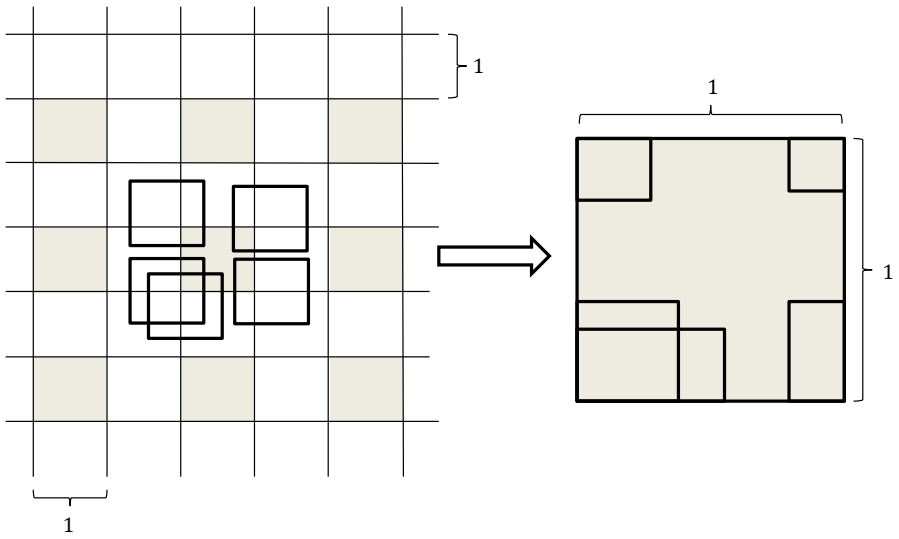

Figure 2.1: Single-Minded Pricing on Unit Squares

Now, we look at odd/even rows/columns and divide the generated unit squares to four groups. Note that, this induces a partition of the points (items) as well, since every point is in exactly one generated unit square. The points in one of these four groups give us at least a quarter of the optimal revenue (since the relation between the prices of the points and the objective function is linear). We denote generated unit squares in that group as selected unit squares (In Figure 2.1, the selected unit squares are shown in gray color).

We assign a price of zero to all items except for ones that fall within the selected unit squares. Note that each demand can have points in at most one selected unit square. Therefore, this will partition the current instance into independent sub-instances where for each independent sub-instance all the items (points) are within a selected unit square (In Figure 2.1, on the right, you can see an example of an independent sub-instance, where the unit square shown is a selected unit square and rectangles shown within the selected unit square are parts of the demands that fall within the selected unit square).

Consider any of these independent sub-instances. Since the items are within the selected unit square and the demands that have some of these items in their set, are also represented by unit squares, therefore, each unit square representing a demand contains at least one of the corners of the selected unit square. We partition the demands into four groups each corresponding to a corner of the selected unit square (For a demand such that its unit square contains more than one corner of the selected unit square, we put it into one of those corners' groups arbitrarily). Again, the demands in one of these groups will give us at least a quarter of the optimal revenue for this sub-instance. Without loss of generality, let us assume it is the demands that their unit squares contain the bottom-left corner of the selected unit square.

We can consider only the part of each demand's unit square that falls within the selected unit square since the other part has no points with non-zero price (See Figure 2.1, on the right for an example). Therefore, this case is basically the Single-Minded Pricing problem on Corner Rectangles for which there is a QPTAS [12].

We lost a factor of 4 in the approximation at first, by assigning a price of zero to all points except for the points within the selected unit squares (We consider all 4 cases and choose the one with the most generated revenue). We also lost another factor of 4 in each independent sub-instance by only considering the demands that contain a specific corner of the selected unit square (Again, we consider all 4 corner cases and choose the one with the most generated revenue). Using the QPTAS for SMP on Corner Rectangles, we get a $(16+\epsilon)$-approximation algorithm running in quasi-polynomial time for any fixed constant $\epsilon > 0$ for SMP on Unit Squares. □

We continue by considering the Single-Minded Pricing problem on squares of arbitrary size (By size of a square, we mean the length of its side). In SMP on Squares, we have a set of items

represented as points on the plane and a set of demands represented as squares each demanding the points within. The objective function for an instance of SMP on Squares is similar to the objective function for an instance of Single-Minded Pricing in the general setting. By size of a demand $d \in D$, we mean the size of the square it is represented by, and denote it by $|d|$. Note that in Single-Minded Pricing on geometric objects, by scaling the plane we can increase or decrease the size of our objects without loss of generality. We will use this in the proof of the following theorem:

Theorem 2.2.2. *There is an $O(\log L)$-approximation algorithm in quasi-polynomial time for SMP on Squares where L is the ratio between the maximum and the minimum over all sizes of the squares.*

Proof. We scale the plane so that the smallest square has size 1. This means that the sizes of squares fall between 1 and L (including both). For $0 \leq i \leq \lfloor \log_2 L \rfloor$, we put the demands $d \in D$ with size $2^i \leq |d| < 2^{i+1}$ in group i. Since, the demands in one of these groups can give us at least $1/(\log_2 L + 1)$ of the optimal revenue, therefore, by losing a factor of $O(\log L)$ in the approximation ratio, we can assume that square sizes are at least 2^j and less than 2^{j+1} for some $0 \leq j \leq \lfloor \log_2 L \rfloor$. We scale down the plane by factor of 2^j, so that square sizes are at least 1 and less than 2, without loss of generality.

Consider horizontal and vertical lines on the plane of distance 4 apart, where the locations of the lines are chosen totally at random. Since the distance between the lines is 4, each square can intersect at most one horizontal and one vertical line since its size is less than 2 and the probability of a square to intersect both a horizontal and a vertical line is at least 1/16 (1/4 for horizontal and 1/4 for vertical line) since its size is at least 1 (See Figure 2.2, for an illustration).

From this point forward, we only consider demands that intersect both a vertical and a horizontal line. We will lose another factor of 16 in the approximation ratio. Since lines are of distance 4 apart and the squares have size less than 2, by considering only these kinds of demands (intersecting horizontal and vertical lines), the instance in fact can be partitioned into independent sub-instances where for each sub-instance, all the demands contain the intersection of a horizontal and a vertical line (See Figure 2.2, on the right, for a sub-instance). Since sub-instances are independent, we will try to solve each independently and the total revenue will be the sum of the revenues of these sub-instances.

Consider any of the mentioned sub-instances. We can divide the points of this instance into four groups, by considering their locations with respect to the horizontal and the vertical lines (For example, the points in the gray area in Figure 2.2, are on the top-right of the lines' intersection point). Since the points in one of these four groups can give us 1/4 of the optimal revenue for this sub-instance, w.l.o.g we assume that it is the points on the top-right side of the lines' intersection (The gray area in Figure 2.2) and assign a price of zero to all the other points in the sub-instance. This will cost us another factor of 4 in the approximation ratio.

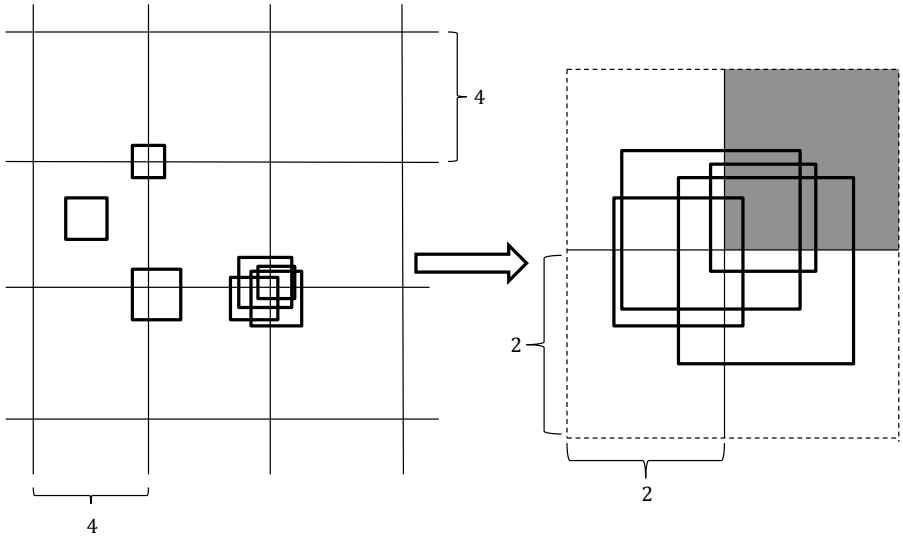

Figure 2.2: Single-Minded Pricing on Squares

We can consider only the part of each square that falls within the gray area that is shown in Figure 2.2, since the other parts have no points with non-zero prices. Therefore, this case is basically the Single-Minded Pricing problem on Corner Rectangles for which there is a QPTAS [12].

We lost a factor of $O(\log L)$ in the approximation ratio at first to be able to assume that the square sizes are between 1 and 2. We also only considered demands that intersect a vertical and a horizontal line which costs us another factor of 16. Finally we only considered the points in one area out of four areas which again costs us a factor of 4 in the approximation ratio. What was left, was an instance of SMP on Corner Rectangles for which there is a QPTAS [12]. Putting all of these together, we get a randomized $O(\log L)$-approximation algorithm running in quasi-polynomial time for SMP on Squares.

We can easily derandomize this algorithm by exhaustively checking all possibilities for vertical and horizontal lines. Although the location of lines can be any real number, the number of important locations are polynomial in the number of items and demands (by looking at the coordinates of the points and the corners of the squares). So we can derandomize the solution in polynomial time by exhaustively searching all the possible cases and get a deterministic $O(\log L)$-approximation algorithm running in quasi-polynomial time for SMP on Squares.

□

Recall that line rectangles are a set of rectangles on the plane such that their lower sides fall on the same line. In SMP on Line Rectangles, we have a set of items represented as points on the plane and a set of demands represented as line rectangles each demanding the points within. Again, the objective function here is similar to the objective function for an instance of Single-Minded Pricing in the general setting.

Theorem 2.2.3. *There is an $O(\log n)$-approximation algorithm in quasi-polynomial time for SMP on Line Rectangles.*

Proof. First we sort our items according to their x-value. So, we can assume that our items are given as $u_1, u_2, \ldots, u_n$, where if (x_i, y_i) denotes the coordinates of item u_i, then $x_1 \leq x_2 \leq \cdots \leq x_n$. For convenience, we assume n is a power of 2 (which we can always achieve by adding extra points outside of objects and in a way that the order of x-coordinates is still non-decreasing).

We begin by partitioning the demands (line rectangles) into $\log_2 n$ groups. Specifically, let D_1 be the set of all demands that intersect line $x = x_{\frac{n}{2}}$ (vertical line at point $x_{\frac{n}{2}}$). Let D_2 be the set of all demands not in D_1 which intersect either $x = x_{\frac{n}{4}}$ or $x = x_{\frac{3n}{4}}$. More generally, for $1 \leq i \leq \log_2 n$ let D_i be the set of all demands not in $D_1 \cup D_2 \cup \cdots \cup D_{i-1}$ which intersect line $x = x_{\frac{(2k+1)n}{2^i}}$ for some $0 \leq k < 2^{i-1}$. The demands in one of these groups can achieve at least $1/\log_2 n$ of the optimal revenue. Let us assume it is the demands in D_j for some $1 \leq j \leq \log_2 n$ (by computing the revenue for all groups and choosing the maximum overall, we can find the group D_j). So, by losing a factor of $O(\log n)$ in the approximation ratio, we can assume that the set of demands in the instance is D_j.

Note that, the demands in D_j are the demands that intersect line $x = x_{\frac{(2k+1)n}{2^j}}$ for some $0 \leq k < 2^{j-1}$ and do not intersect lines $x = x_{\frac{(2k)n}{2^j}}$ for any $0 \leq k \leq 2^{j-1}$. This means that, the instance with the demands in D_j can be partitioned into 2^{j-1} independent sub-instances, where for $0 \leq k < 2^{j-1}$, the k^{th} sub-instance consists of demands that intersect line $x = x_{\frac{(2k+1)n}{2^j}}$. Since the sub-instances are independent, the total revenue for the instance will be the sum of the revenues for the sub-instances. For this reason, we consider each sub-instance independently.

In each independent sub-instance, we are given a set of demands represented by line rectangles, where all the line rectangles intersect a given vertical line. Either the points on the right side of the vertical line or the points on the left side of the vertical line can output at least half the optimal revenue for this instance. Without loss of generality, we assume the points on the right side of the vertical line can provide us with at least half the optimal revenue and assign a price of zero to all the points on the left side of the vertical line (we check for both cases and choose the case with more revenue). This will cost us another factor of 2 in the approximation ratio. Since all the points on the left side of the vertical line are assigned a price of zero, we can actually consider only the parts of the line rectangles that are on the right side of the vertical line. This will be in fact another representation of SMP on Corner Rectangles for which there is a QPTAS [12].

We lost a factor of $O(\log n)$ in the approximation at first by only considering the demands in group D_j. We lost another factor of 2 in the next step in each independent sub-instance. The final instance was in fact an instance of SMP on Corner Rectangles, for which Chalermsook *et al.* gave a QPTAS [12]. Putting all of these together, we get an $O(\log n)$-approximation algorithm running in quasi-polynomial time for SMP on Line Rectangles.

□

In SMP on Rectangles, we have a set of items represented as points on the plane and a set of demands represented as rectangles each demanding the points within. We want to assign prices to the points in order to maximize the total revenue which is the sum of all sold items (points).

Theorem 2.2.4. *There is an $O(\log^2 n)$-approximation algorithm in quasi-polynomial time for SMP on Rectangles.*

Proof. First we sort our items according to their y-value. So, we can assume that our items are given as $u_1, u_2, \ldots, u_n$, where if (x_i, y_i) denotes the coordinates of item u_i, then $y_1 \leq y_2 \leq \cdots \leq y_n$. For convenience, we assume n is a power of 2 (which we can always achieve by adding extra points outside of objects and in a way that the order of y-coordinates is still non-decreasing).

We begin by partitioning the demands (rectangles) into $\log_2 n$ groups. Specifically, let D_1 be the set of all demands that intersect line $y = y_{\frac{n}{2}}$ (horizontal line at point $u_{\frac{n}{2}}$). Let D_2 be the set of all demands not in D_1 which intersect either $y = y_{\frac{n}{4}}$ or $y = y_{\frac{3n}{4}}$. More generally, for $1 \leq i \leq \log_2 n$ let D_i be the set of all demands not in $D_1 \cup D_2 \cup \cdots \cup D_{i-1}$ which intersect line $y = y_{\frac{(2k+1)n}{2^i}}$ for some $0 \leq k < 2^{i-1}$. The demands in one of these groups can achieve at least $1/\log_2 n$ of the optimal revenue. Let us assume it is the demands in D_j for some $1 \leq j \leq \log_2 n$ (by computing the revenue for all groups and choosing the maximum overall, we can find the group D_j). So, by losing a factor of $O(\log n)$ in the approximation ratio, we can assume that the set of demands in the instance is D_j.

Note that, the demands in D_j are the demands that intersect line $y = y_{\frac{(2k+1)n}{2^j}}$ for some $0 \leq k < 2^{j-1}$ and do not intersect lines $y = y_{\frac{(2k)n}{2^j}}$ for any $0 \leq k \leq 2^{j-1}$. This means that, the instance with the demands in D_j can be partitioned into 2^{j-1} independent sub-instances, where for $0 \leq k < 2^{j-1}$, the k^{th} sub-instance consists of demands that intersect line $y = y_{\frac{(2k+1)n}{2^j}}$. Since the sub-instances are independent, the total revenue for the instance will be the sum of the revenues for the sub-instances. For this reason, we consider each sub-instance independently.

In each independent sub-instance, we are given a set of demands represented by rectangles, where all the rectangles intersect a given horizontal line. Either the points in this sub-instance higher than the horizontal line or the points lower than the horizontal line can output at least half the optimal revenue. Without loss of generality, we assume the points higher than the horizontal line can provide us with at least half the optimal revenue and assign a price of zero to all the points lower than the

horizontal line (we check for both cases and choose the case with more revenue). This will cost us another factor of 2 in the approximation ratio. Since, all the points lower than the horizontal line are assigned a price of zero, we can actually consider only the parts of the rectangles that are higher than the horizontal line. This will be in fact another representation of SMP on Line Rectangles for which in Theorem 2.2.3, we introduced an $O(\log n)$-approximation algorithm running in quasi-polynomial time.

We lost a factor of $O(\log n)$ in the approximation at first by only considering the demands in group D_j. We lost another factor of 2 in the next step in each independent sub-instance. The final instance was in fact an instance of SMP on Line Rectangles, for which we presented an $O(\log n)$-approximation algorithm in quasi-polynomial time in Theorem 2.2.3. Putting all of these together, we get an $O(\log^2 n)$-approximation algorithm running in quasi-polynomial time for SMP on Rectangles.

□

2.3 SMP on Graphs with Bounded Pathwidth

In this section we discuss the hardness of SMP on general graphs. As mentioned in Section 2.1, SMP on Trees is known to be APX-hard and the best known approximation for it has ratio $O(\frac{\log n}{\log \log n})$. For general graphs, the problem becomes very hard. Chalermsook *et al.* [11] proved that SMP on graphs is as hard as Unique Coverage.

A Path-decomposition of a graph G is a sequence of subsets X_i of vertices of G, with two properties:

- For each edge of G, there exists an i such that both endpoints of the edge belong to subset X_i, and
- For every three indices $i \leq j \leq k$, $X_i \cap X_k \subseteq X_j$.

The width of a path decomposition is the size of its largest set X_i minus one. The pathwidth of G is the minimum width of any path-decomposition of G.

We prove that the graph Chalermsook *et al.* [11] provided in their instance to prove the hardness of SMP on graphs has pathwidth equal to 2 and also provide a constant approximation solution when the pathwidth of the given graph is equal to 1.

Theorem 2.3.1. *SMP on graphs is as hard as Unique Coverage even when the pahtwidth of the given graph is* 2.

Proof. The graph Chalermsook *et al.* [11] provided in their solution to prove that SMP on graphs is as hard as UC, is shown in Figure 2.3.

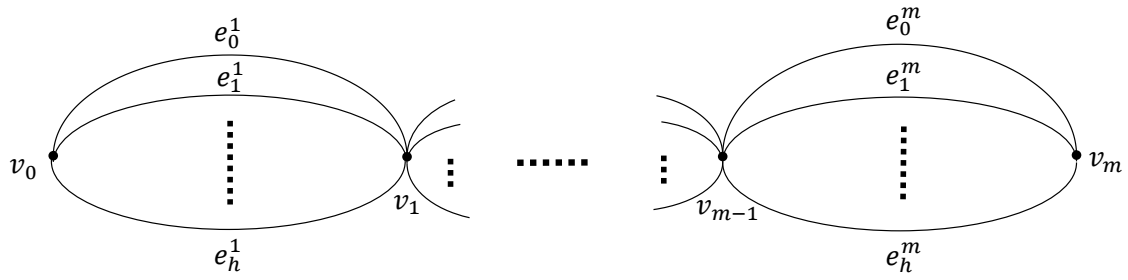

Figure 2.3: Lower Bound for Single-Minded Pricing on Graphs with Bounded Pathwidth

Their construction is as follows: Let (U, D) be an instance of Unique Coverage, where $|U| = n$ and $|D| = m$. Recall that in Unique Coverage, given a set $U = \{u_1, \ldots, u_n\}$ of elements, and given a collection $D = \{D_1, \ldots, D_m\}$ of subsets of U, we want to find a subcollection $D' \subseteq D$ to maximize the number of elements that are uniquely covered, *i.e.,* appear in exactly one set of D'. They construct an instance of SMP on graphs as follows: Graph $G = (V, E)$ consists of $m+1$ vertices $v_0, \ldots, v_m$. Let $h = \lceil \log_2 m \rceil$. For each consecutive pair (v_{i-1}, v_i) of vertices, $0 < i \leq m$, they add $h + 1$ parallel edges $e_0^i, \ldots, e_h^i$. These edges are viewed as representing the set $D_i \in D$. We now define the set of the paths (the demands) in the graph. All the paths start from v_0 and end at v_m. For each element $u \in U$ and for each $j : 1 \leq j \leq h$, there is a set $P(u, j)$ of 2^{h-j} paths. The budget of each path in $P(u, j)$ is 2^j, the source vertex is v_0, and the sink is v_m. Each path in $P(u, j)$ consists of the edges $e_{i_1}^1, e_{i_2}^2, \ldots, e_{i_m}^m$, where for all $1 \leq l \leq m$, if $u \in D_l$ then $i_l = j$, or otherwise $i_l = 0$. This completes the description of the construction.

As you can see the graph is not simple. Since all the demands that are defined in the instance start at v_0 and end at v_m, in order to handle the parallel edges and make the graph simple, we subdivide the edges. A subdivision of a graph G is a graph resulting from the subdivision of edges in G. The subdivision of some edge e with endpoints $\{u, v\}$ yields a graph containing one new vertex w, and with an edge set replacing e by two new edges, $\{u, w\}$ and $\{w, v\}$. Specifically, for each edge e_j^i we subdivide it using a new vertex called v_j^i. It is easy to see that the construction can be converted back to the original construction easily without changing the revenue of the given instance in SMP. The only change is doubling up the number of edges. This adjustment obviously will not change the statements proven in the original paper [11], that leads to proving that SMP on graphs is as hard as Unique Coverage. Therefore, SMP on this simple graph is still as hard as Unique Coverage in the general setting.

We now prove that the converted simple graph has pathwidth equal to 2. We define a path-decomposition of our simple graph as follows: for any $i : 0 \leq i < m$ and any $j : 0 \leq j \leq h$, we set $X_{i(h+1)+j} = \{v_i, v_j^{i+1}, v_{i+1}\}$. Obviously the sequence of subsets defined as sequence X, covers all the edges and also satisfies the second property which is equivalent to requiring that the subsets containing any particular vertex form a contiguous subsequence of the whole sequence. Since all the subsets in the decomposition have size equal to 3, therefore the width of this decomposition is equal to 2. Thus, the pathwidth of this graph is at most 2 and it is easy to see that it cannot be 1 (includes loops). This proves the theorem.

□

On the other hand, it is easy to come up with a constant approximation solution for SMP on graphs where the input graph's pathwidth is equal to 1.

Theorem 2.3.2. *There is a $(5+\epsilon)$-approximation for SMP on graphs with pathwidth $= 1$ for any fixed constant $\epsilon > 0$.*

Proof. The maximal graph with pathwidth equal to 1 is called a Caterpillar tree in which all the vertices are within distance 1 of a central path (See Figure 2.4 for an example of Caterpillar tree).

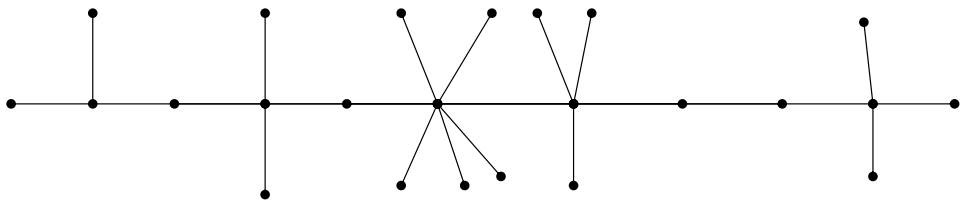

Figure 2.4: Upper Bound for Single-Minded Pricing on Graphs with Bounded Pathwidth

By OPT we denote the maximum revenue obtainable for the instance, given a graph and a set of paths as the demands. We consider the following cases.

- Case 1: At least $\frac{1}{5}$OPT is achieved from edges on the central path.

  In this case we assign a price of zero to all the edges other than the edges on the central path. This way all the demands can be considered to be only on the central path and the problem turns into the Highway problem for which there is a $PTAS$ [31].

- Case 2: At least $\frac{4}{5}$OPT is achieved from the edges other than the ones on the central path.

  In this case we assign a price of zero to all the edges on the central path. This way we can contract the central path down to a vertex, and our graph will turn into a star graph. A star graph is a tree which consists of a single internal vertex (and $|V| - 1$ leaves). Since in this case each demand has at most two items, this case is another representation of Graph Vertex Pricing, for which there is a 4-approximation algorithm [3].

In both cases, we get a constant approximation solution. Overall, we achieve an $(5 + \epsilon)$-approximation algorithm for SMP on graphs with pathwidth equal to 1.

□

Note that, trees can have pathwidth up to $O(\log n)$. So, these results do not settle any complexity status for SMP on trees.

Chapter 3

Unique Coverage

In this chapter, we study the Unique Coverage problem in different settings such as on graphs and geometric objects. Unique Coverage (UC) is defined as follows: Given a set $U = \{u_1, \ldots, u_n\}$ of elements, and a collection $D = \{D_1, \ldots, D_m\}$ of subsets of U, find a subcollection $D' \subseteq D$ to maximize the number of elements that are uniquely covered, *i.e.,* appear in exactly one set of D'.

We start by mentioning some previous works done in this area, then we introduce dynamic programming algorithms for the Unique Coverage problem on corner rectangles and upward trees.

3.1 Related Works

Demaine *et al.* [19] were the first to study the Unique Coverage problem thoroughly. They proved semilogarithmic inapproximability for the Unique Coverage problem. Specifically, they proved that for any arbitrary small constant $\epsilon > 0$, assuming that NP $\not\subseteq$ BPTIME(2^{n^ϵ}) (To see the definition of the BPTIME class, refer to [5]), it is hard to approximate the Unique Coverage problem within a factor of $O(\log^\sigma n)$ for some constant $\sigma = \sigma(\epsilon)$. They also proved that assuming the R3SAT hardness hypothesis (See [26] for the definition of the R3SAT hardness hypothesis), UC is hard to approximate within a factor of $O(\log^{1/3-\sigma} n)$ for an arbitrary small constant $\sigma > 0$. Under a different hypothesis for hardness of Balanced Bipartite Independent Set (Refer to [19] for the formal definition of the problem and the hypothesis), they show that it is hard to approximate UC within a factor of $O(\log n)$, where the constant in the $O(.)$ term depends on ϵ.

On the upper bound side, Demaine *et al.* [19] presented an $O(\log l)$-approximation algorithm for UC where every set has at most l elements. They also came up with an $O(\log \beta)$-approximation for UC where every element is present in at most β sets.

Guruswami *et al.* [34] similarly proved that there is an $O(\log l)$-approximation algorithm for UC in the general setting. They also presented an e-approximation algorithm when every set has exactly l elements and proved $(e - \epsilon)$-inapproximability for this case, for any fixed constant $\epsilon > 0$.

Leeuwen [48] was the first to prove that UC on Unit Disks and Unit Squares are NP-hard. He also presented a 2-approximation algorithm for UC on Unit Squares. This result was improved later by Ito *et al.* [39] to a PTAS. Erlebach *et al.* [25] provided an 18-approximation algorithm for UC on Unit Disks. This was also improved by Ito *et al.* [38] to factor of 4.31. Erlebach *et al.* [25] also presented an asymptotic FPTAS for UC on disks with bounded ply. The only result for UC on graphs is an $O(\log \log n)$-approximation algorithm for UC on Trees by Cygan *et al.* [18].

3.2 UC on Corner Rectangles

In this section we introduce a dynamic programming algorithm for UC on Corner Rectangles. In UC on Corner Rectangles, given a set $U = \{u_1, \ldots, u_n\}$ of points on the plane, and a collection $D = \{D_1, \ldots, D_m\}$ of corner rectangles (axis parallel rectangles with the same bottom-left corner), we want to find a subset $U' \subseteq U$ of points to maximize the number of corner rectangles that are uniquely hit, *i.e.,* only have one point of U' within.

Consider an instance (U, D) of UC on Rectangles and assume that if a point $u \in U$ is on a side of a rectangle $d \in D$, then $u \notin d$. We can move the points and the sides of the rectangles slightly so that afterwards no two points or sides of the rectangles are on the same vertical or horizontal line, without changing the size of the optimal solution. Our approach is simple. For example, if the right side of a rectangle $d \in D$ is on the same vertical line as some points or sides of some other rectangles, we move all the points and the sides of the other rectangles that are on the vertical line, slightly to the right and this way no points or sides of rectangles will be on the vertical line except for the right side of d. Other cases such as points falling on the same vertical or horizontal line can be handled similarly. The same approach can be modified and applied easily on UC on Line Rectangles and UC on Corner Rectangles. Therefore, on any of these problems we can assume that no two points, two sides of rectangles, or a point and a side of a rectangle are on the same horizontal or vertical line.

Theorem 3.2.1. *There is a dynamic programming algorithm for Unique Coverage on Corner Rectangles, which is an exact algorithm and runs in polynomial time.*

Proof. We assume that no two points or the right and the top sides of rectangles are on the same vertical or horizontal line (as mentioned before, we can assume this w.l.o.g). We sort the points according to their x-value from small to large. Assuming that our elements are $u_1, u_2, \ldots, u_n$, where (x_i, y_i) denotes the coordinates of element u_i, for $1 \leq i \leq n$, then $x_1 < x_2 < \cdots < x_n$. We set $x_{n+1} = \infty$ and $x_0 = -\infty$.

The basic idea that helps us solve this problem is that as we move from left to right on the plane and select the points to be in the output set, we do not have to remember all the selected points. We only need to remember the two lowest ones, since all corner rectangles have the same left-bottom

corner.

We define function ϕ as follows: $\phi_i(j,k)$ stores the maximum number of corner rectangles that their right side is to the right of line $x = x_i$ and can be uniquely hit assuming that among the selected points in $\{u_1, \ldots, u_i\}$, u_j (u_k) has the lowest (second lowest, respectively) height ($1 \leq j, k \leq i$ and $y_j < y_k$). If there is only one selected point (u_j) in $\{u_1, \ldots, u_i\}$, then we denote the function ϕ with $\phi_i(j, \emptyset)$, and if there are no points chosen in $\{u_1, \ldots, u_i\}$, we denote it by $\phi_i(\emptyset, \emptyset)$. See Figure 3.1 for an illustration (The points shown in the figure are chosen to be in the output set. Note that j and k are the two chosen points with the lowest height in $\{u_1, \ldots, u_i\}$).

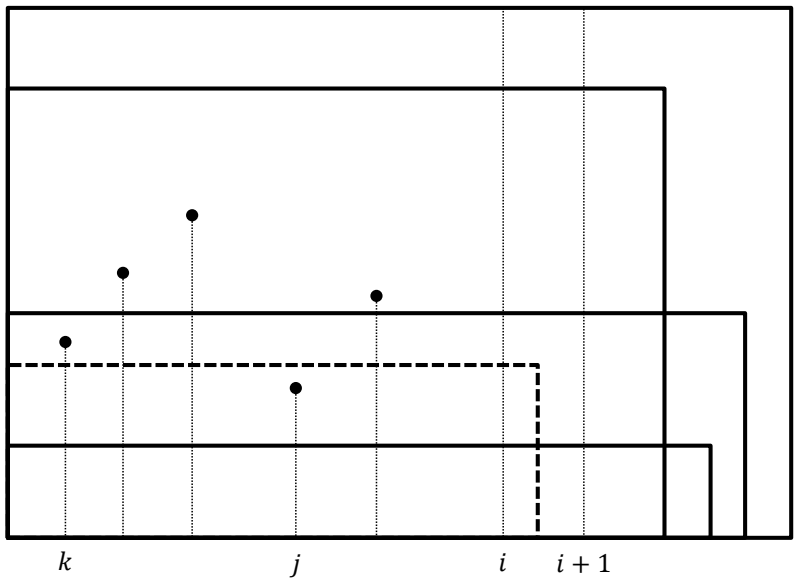

Figure 3.1: Unique Coverage on Corner Rectangles

We also define function f as follows: $f_i(j,k)$ stores the maximum number of corner rectangles that their right sides fall between lines $x = x_i$ and $x = x_{i+1}$ and can be uniquely hit, assuming that among the selected points in $\{u_1, \ldots, u_i\}$, u_j (u_k) has the lowest (second lowest, respectively) height ($1 \leq j, k \leq i$ and $y_j < y_k$). If there is only one selected point (u_j) in $\{u_1, \ldots, u_i\}$, then we denote the function f with $f_i(j, \emptyset)$, and if there are no points chosen in $\{u_1, \ldots, u_i\}$, we denote it by $f_i(\emptyset, \emptyset)$. We can compute $f_i(j,k)$ for any $1 \leq i \leq n$ and $(j,k) \in \{\emptyset, 1, 2, \ldots, n\}^2$ in polynomial time by counting the number of corner rectangles with their right sides between lines $x = x_i$ and $x = x_{i+1}$ and their top sides between $y = y_j$ and $y = y_k$. Computing $f_i(j, \emptyset)$ and $f_i(\emptyset, \emptyset)$ is also

similar and easy. In Figure 3.1, the dashed corner rectangle is a uniquely hit corner rectangle that has contributed in $f_i(j,k)$.

To compute $\phi_i(j,k)$, we can divide the corner rectangles considered in it, into two groups. The first group contains the corner rectangles that their right sides fall between $x = x_i$ and $x = x_{i+1}$. We denote by X_1 the maximum number of uniquely hit corner rectangles that are in the first group. The second group contains the corner rectangles that their right side falls to the right of line $x = x_{i+1}$. We also denote by X_2 the maximum number of uniquely hit corner rectangles that are in the second group ($\phi_i(j,k) = X_1 + X_2$). It is easy to see that $X_1 = f_i(j,k)$. To compute X_2, we should decide if we want to choose u_{i+1} to be in the output set or not and also consider its height. If $y_{i+1} < y_j$, then $X_2 = \max\{\phi_{i+1}(j,k), \phi_{i+1}(i+1,j)\}$ since we have to consider both choosing and not choosing u_{i+1} ($\phi_{i+1}(i+1,j)$ and $\phi_{i+1}(j,k)$, respectively) and choose the one with the larger value. The other cases are also similar. Therefore, we can recursively compute $\phi_i(j,k)$ as follows (The recursive relation is similar for computing $\phi_i(j,\emptyset)$ and $\phi_i(\emptyset,\emptyset)$):

$$\phi_i(j,k) = f_i(j,k) + \begin{cases} \phi_{i+1}(j,k) & y_k < y_{i+1} \\ \max\{\phi_{i+1}(j,k), \phi_{i+1}(j,i+1)\} & y_j < y_{i+1} < y_k \\ \max\{\phi_{i+1}(j,k), \phi_{i+1}(i+1,j)\} & y_{i+1} < y_j \end{cases}$$

Using dynamic programming approach, we can compute and store the values of the recursive function ϕ in polynomial time. The base cases will be when $i = n$ since:

$$\phi_n(j,k) = f_n(j,k)$$

Algorithm 1 Dynamic Program for UC on Corner Rectangles

```
1: Sort the points based on their x-coordinate value from left to right
2: for any (j,k) ∈ {∅,1,2,...,n}^2 do                    ▷ Initializing Base Case
3:     if j ≠ k and y_j < y_k then
4:         φ_n(j,k) ← f_n(j,k)
5: for i ← n − 1 downto 0 do                              ▷ Dynamic Program
6:     for any j,k ∈ {1,2,...,n} ∪ {∅} do
7:         if j ≠ k and y_j < y_k then
8:             sum ← f_i(j,k)
9:             if y_k < y_{i+1} then
10:                sum ← sum + φ_{i+1}(j,k)
11:            else if y_j < y_{i+1} < y_k then
12:                sum ← sum + max(φ_{i+1}(j,k), φ_{i+1}(j,i+1))
13:            else
14:                sum ← sum + max(φ_{i+1}(j,k), φ_{i+1}(i+1,j))
15:        φ_i(j,k) ← sum
```

The dynamic programming algorithm for UC on Corner Rectangles is shown in Algorithm 1. The variable *sum* is a temporary variable that we use to store the total number of uniquely hit

corner rectangles as we compute them. The table size for function f is $O(n^3)$ and computing the value of each element in this table can be done in time $O(m)$ (By iterating once over all the corner rectangles and counting the ones that are uniquely hit). The dynamic programming table size for function ϕ is also $O(n^3)$ and computing the value of each element in this table can be done in $O(1)$ (We should note that we store all the values of the elements in the table for function f prior to computing the values of the elements in the table for function ϕ). Therefore, the total running time of the algorithm is $O(m \times n^3)$ which is polynomial in m and n.

Obviously we are interested in $\phi_0(\emptyset, \emptyset)$ which considers all the corner rectangles since all the corner rectangles have their top-right point to the right of point $x_0 = -\infty$. Therefore, $\phi_0(\emptyset, \emptyset)$ outputs the maximum number of corner rectangles that are uniquely hit. Note that, we did not compute the set of the points maximizing the uniquely hit number of corner rectangles. If the set of the chosen points is also required, we can simply store each decision we make when we are using the function max and then simply by backtracking from $\phi_0(\emptyset, \emptyset)$ we can output the optimal subset of the points leading to the maximum number uniquely hit corner rectangles.

□

3.3 UC on Upward Trees

In this section we present a dynamic programming algorithm that runs in polynomial time for UC on Upward Trees. In UC on Upward Trees, given a set of edges $E = \{e_1, \ldots, e_n\}$ in a tree $T = (V, E)$, and a collection $D = \{D_1, \ldots, D_m\}$ of upward paths in T (A path is called upward if one of its endpoints is an ancestor to the other endpoint), the goal is to find a subset $E' \subseteq E$ of the edges that maximizes the number of upward paths that are uniquely covered, *i.e.,* only have one edge in E'. Note that the best previous result related to this problem is an $O(\log \log n)$-approximation algorithm for UC on trees [18] (where the paths are not necessarily upward).

Theorem 3.3.1. *UC on Upward Trees can be solved in polynomial time.*

Proof. For each node $v \in V$, we denote by T_v the subtree rooted at v (including v) and by P_v the path from v to the root. We also denote by $ch(v)$ the set of children for node $v \in V$. For each node $v \in V$ in the tree and edges e, e' on P_v, $\phi_v(e, e')$ is to store the maximum number of paths in D which have an end-point in T_v and can be uniquely covered if e and e' are the closest and the second closest edges to v, respectively, that are selected. It is possible for e' and even e to be null. Indeed, when the there is only one edge e selected on P_v, we are interested in $\phi_v(e, \emptyset)$ and when there are no selected edges on P_v, we are interested in $\phi_v(\emptyset, \emptyset)$.

For each node $v \in V$ in the tree and edges e, e' on P_v, $f_v(e, e')$ is to store the maximum number of paths in D which their bottom most node is v and can be uniquely covered if e and e' are the

closest and the second closest edges to v, respectively, that are selected. It is possible for e' and even e to be null. Indeed, when the there is only one edge e selected on P_v, we are interested in $f_v(e, \emptyset)$ and when there are no selected edges on P_v, we are interested in $f_v(\emptyset, \emptyset)$. For each node $v \in V$ in the tree and edges e, e' on P_v, $f_v(e, e')$ can be computed easily in polynomial time by counting the number of paths in D that their bottom most node is v and their top most node falls between e and e'. Computing $f_v(e, \emptyset)$ and $f_v(\emptyset, \emptyset)$ are also easy.

To compute $\phi_v(e, e')$, we consider the upward paths in D that their bottom most node is in T_v. We divide these upward paths into two groups. The first group contains upward paths that their bottom most node is exactly v. $f_v(e, e')$ computes the exact number of uniquely covered upward paths among the ones in the first group. The second group of paths are the ones in D that their bottom most node is in the subtree rooted at v excluding v, or in other words, the upward paths in D that their bottom most node is in the subtree rooted at a child of v (including the child). This gives us the idea to find the recursive relation for function ϕ. To compute function ϕ recursively, we should decide if we want to select the edge between v and each child or not. By choosing the maximum value of these two cases we can compute the recursive relation for function ϕ. The recursive function $\phi_v(e, e')$ is computed as follows (the recursive relations for $\phi_v(e, \emptyset)$ and $\phi_v(\emptyset, \emptyset)$ are also similar):

$$\phi_v(e, e') = f_v(e, e') + \sum_{u \in ch(v)} \max(\phi_u(e, e'), \phi_u(\{u, v\}, e))$$

We store the values for this recursive function using a dynamic programming approach. We should store the values in the dynamic programming table as we go up on the tree. Indeed, since leaves of the tree have no children, function ϕ for a leaf $u \in V$ will be:

$$\phi_u(e, e') = f_u(e, e')$$

The dynamic programming algorithm for UC on Upward Trees is given in Algorithm 2. The variable *sum* is a temporary variable that we use to store the total number of uniquely covered upward paths as we compute them. The table size for function f is $O(n^3)$ and computing the value of each element in this table can be done in time $O(m)$ by iterating over all paths in D and counting the ones that are uniquely covered. The dynamic programming table size for function ϕ is $O(n^3)$ and updating the value of each element in the table is $O(n)$ (the number of children for each node). Therefore, the overall running time of the algorithm is $O((m + n) \times n^3)$ which is polynomial in m and n.

We denote the root of T as r. We are interested in $\phi_r(\emptyset, \emptyset)$ which considers all the paths in D since any path's bottom most node is in $T_r = T$. Therefore, $\phi_r(\emptyset, \emptyset)$ outputs the maximum

Algorithm 2 Dynamic Program for UC on Upward Trees

for any leaf $v \in V$ **do** ▷ Initializing Base Case
  for $e, e' \in E \cup \{\emptyset\}$ **do**
    if e and e' are on P_v and e is closer to v than e' **then**
      $\phi_v(e, e') \leftarrow f_v(e, e')$
Run a BFS on tree starting at root and sort the nodes based on their distance from the root in decreasing order
for any non-leaf $v \in V$ **do** ▷ Dynamic Program
  for $e, e' \in E \cup \{\emptyset\}$ **do**
    if e and e' are on P_v and e is closer to v than e' **then**
      $sum \leftarrow f_v(e, e')$
      for any child $u \in ch(v)$ **do**
        $sum \leftarrow sum + \max(\phi_u(e, e'), \phi_u(\{u, v\}, e))$
      $\phi_v(e, e') \leftarrow sum$

number of paths in D that are uniquely covered. Note that, we did not compute the set of the edges that maximize the number of uniquely covered paths in D. If the set of the selected edges is also required, we simply store each decision we make when we are using the function max and then simply by backtracking from $\phi_r(\emptyset, \emptyset)$ we can output the optimal subset of edges leading to the maximum number uniquely covered paths in D.

□

Chapter 4

Uniform-Budget Single-Minded Pricing

In this chapter we study the Uniform-Budget Single-Minded Pricing (UBSMP) problem which is a special case of Single-Minded Pricing. In Uniform-Budget Single-Minded Pricing, we are given a set U of items and a collection D of subsets of U, called demands. For each demand $d \in D$, we are also given a budget $B(d) = 1$ which is the maximum revenue obtainable from d. If each item $u \in U$ is priced at $p(u) \in \mathbb{R}^+$, the revenue from demand $d \in D$ will be $\sum_{u \in d} p(u)$ if $\sum_{u \in d} p(u) \leq 1$ and 0 otherwise. We want to assign prices to the items in order to maximize the total revenue. Specifically, we want to assign prices $p : U \mapsto \mathbb{R}^+$ to the items to maximize:

$$\sum_{d \in D : 1 \geq \sum_{u \in d} p(u)} \sum_{u \in d} p(u).$$

For any instance of Single-Minded Pricing, it is easy to see that by scaling (up or down) the budgets and the prices we can get to another equivalent instance, because the optimal prices in the second instance are the scaled optimal prices of the first instance. For this reason, if in an instance for Single-Minded Pricing, the budgets are uniform but not equal to one, we can convert the instance to an equivalent Uniform-Budget Single-Minded Pricing instance where all the budgets are equal to 1.

We start with some related works, then study the close relation between UBSMP and the Unique Coverage problem and finally we prove that UBSMP on Upward Trees can be solved in polynomial time.

4.1 Related Works

Demaine *et al.* [19] and Guruswami *et al.* [34] proved that their semilogarithmic inapproximability and also upper bound approximation results for the Unique Coverage problem also extends to UBSMP.

Guruswami *et al.* [33] proved that UBSMP is APX-hard even when the demand sizes are at most 2. This special case is equivalent to UBSMP on star graphs. A star graph $G = (V, E)$ is a tree which consists of a single internal vertex (and $|V| - 1$ leaves). Since any path in a star graph has size at most two, and for any demand with size at most two, there is a path to represent it in the star graph, thus we can conclude that UBSMP is APX-hard on trees. Also since the pathwidth of a star graph is equal to 1, therefore UBSMP is APX-hard even on graphs with pathwidth equal to 1. In another related work, Cygan *et al.* [18] presented an $O(\log \log n)$-approximation algorithm for UBSMP on trees.

4.2 The Relation Between UBSMP and UC

In this section we discuss the close relation between UBSMP and UC and we will mention what results can be achieved considering this relation. UC is in fact a special case of UBSMP, where the prices of items are either 0 or 1. Indeed, consider such an instance of UBSMP. In this instance, we want to choose a subset of items to assign a price of 1, where for each demand, if only one of its items is priced 1, then we get a revenue of 1 from that demand and otherwise we get nothing. In other words, we want our demands to be uniquely hit, and thus, UC is in fact the special case of UBSMP where the prices are either 0 or 1.

Demaine *et al.* [19] and Guruswami *et al.* [34] both independently proved by randomized rounding that in Uniform-Budget Single-Minded Pricing there is a price assignment using only prices 0 and 1 whose revenue is within a constant factor of the optimal. Demaine *et al.* [19] introduced $2e$ as the constant factor and Guruswami *et al.* [34] came up with e as the constant. Consequently, we can use any result from UC in any setting and by including only a factor of e to the approximation ratio, get a result for UBSMP in that setting.

We proved in Theorem 3.2.1 that UC on Corner Rectangles can be solved in polynomial time. This gives us:

Corollary 4.2.1. *There is an e-approximation algorithm for UBSMP on Corner Rectangles.*

Ito *et al.* [39] presented a PTAS for UC on Unit Squares. This results in:

Corollary 4.2.2. *There is an $(e + \epsilon)$-approximation algorithm for UBSMP on Unit Squares.*

Ito *et al.* [38] came up with a 4.31-approximation algorithm for UC on Unit Disks. Therefore:

Corollary 4.2.3. *There is an* $(4.31e)$*-approximation algorithm for UBSMP on Unit Disks.*

4.3 UBSMP on Upward Trees

In this section, we prove that UBSMP on Upward Trees can be solved in polynomial time. In UBSMP on Upward Trees, we are given a rooted tree $T = (V, E)$, and a set D of upward paths, called demands. A path is called *upward* if one of its endpoints is an ancestor to the other endpoint. Like Uniform-Budget Single-Minded Pricing in the general setting, for each demand $d \in D$ we are also given a budget $B(d) = 1$ which is the maximum revenue obtainable from d. If each edge (item) $e \in E$ is priced at $p(e) \in \mathbb{R}^+$, the revenue from demand $d \in D$ will be $\sum_{e \in d} p(e)$ if $\sum_{e \in d} p(e) \leq 1$ and 0 otherwise. We want to assign prices to the edges in order to maximize the total revenue. Specifically, we want to assign prices $p : E \mapsto \mathbb{R}^+$ to the edges to maximize:

$$\sum_{d \in D : 1 \geq \sum_{e \in d} p(e)} \sum_{e \in d} p(e)$$

The idea to solve this problem comes from a similar result on UBSMP on Path Graphs. Cygan *et al.* [18] proved that:

Lemma 4.3.1. *[18] UC on Path Graphs can be solved in polynomial time.*

This gives us an immediate e-approximation algorithm for UBSMP on Path Graphs, because as we mentioned in the previous section, we can use any result from UC in any setting and by including only a factor of e to the approximation ratio, get a result for UBSMP in that setting. But we can do better, because of the following lemma:

Lemma 4.3.2. *[33] In SMP on Path Graphs, if all budgets are integral, then there is an optimal solution in which all prices are integral.*

Since in UBSMP, all the budgets are in $\{0, 1\}$, Lemma 4.3.1 and Lemma 4.3.2 gives us the following:

Corollary 4.3.3. *UBSMP on Path Graphs can be solved in polynomial time.*

We presented an exact algorithm for UC on Upward Trees in Theorem 3.3.1. Again, this gives us an e-approximation algorithm for UBSMP on Upward Trees. But we can do better in this case as well.

Lemma 4.3.4. *In SMP on Upward Trees, if all the budgets are integral, then there is an optimal solution in which all the prices are integral.*

Proof. Assume that our instance is defined on tree $T = (V, E)$ with a set D of demands. Let p be any price function, and $\tilde{D} \subseteq D$ the set of all demands feasible under p. We show that there is an integral assignment p', such that each demand in $\tilde{D}$ is still feasible under p', and the total revenue obtained from $\tilde{D}$ under p' is at least as large as the one under p. Applying this to the optimal price assignment p^* proves the lemma.

Given the set $\tilde{D}$ of demands that is feasible, *i.e.,* $\sum_{e_j \in d_i} p(e_j) \leq B(d_i)$ for each $d_i \in \tilde{D}$, the optimal price assignment that makes all of $\tilde{D}$ feasible is the solution to the following linear program:

$$\begin{aligned} \text{Maximize} \quad & \sum_{d_i \in \tilde{D}} \sum_{e_j \in d_i} p(e_j) & \\ \text{subject to} \quad & \sum_{e_j \in d_i} M_{ij}\ p(e_j) \leq B(d_i) & \text{for each } d_i \in \tilde{D} \\ & p(e_j) \geq 0 & \text{for each } e_j \in E \end{aligned}$$

Each edge $e_j \in E$ is represented by column j, for $1 \leq j \leq n$, and each feasible upward path $d_i \in \tilde{D}$ is represented by row i, for $1 \leq i \leq |\tilde{D}|$, and if $e_j \in d_i$ then $M[i, j] = 1$ and otherwise $M[i, j] = 0$. We claim that M is totally unimodular, *i.e.,* the determinant of each non-singular square sub-matrix is in $\{-1, 0, +1\}$.

Consider any square sub-matrix of M and denote it by M'. M' is actually a matrix that is achieved by deleting some rows and columns from M. Interestingly, if for each column j that is deleted, we contract e_j in T and denote by $T' = (V', E')$ the new tree achieved this way, then each row i in M' still represents an upward path in T'. That is because for any upward path in a tree if we contract some of the edges on that tree, what is left from the path will still be an upward path, in the new tree. We denote by D', the set of upward paths defined by the rows of M'. We can conclude that in M', column j represents the edge $e'_j \in E'$ and row i represents the upward path $d'_i \in D'$ in T'.

To compute the determinant of M', we choose an arbitrary edge $e'_j \in E'$ (with corresponding column j in M') that one of its endpoints is a leaf in T'. Assume $d'_i \in D'$ (with corresponding row i in M') is the shortest upward path in D' that contains e'_j. The important fact is that, d'_i is a subpath of all other upward paths in D' that also contain e'_j. This is easy to see since all of these demands are upward paths that have the same bottom most edge.

We subtract row i from all the rows in M' that their corresponding upward paths in D' that contain e'_j (except for row i itself). Since d'_i is a subpath of all of these upward paths, after subtraction, all entries of M' will still be in $\{0, 1\}$. Subtracting rows from each other doesn't change the determinant of M'. We should also note that, the subtracted rows still represent upward paths, since subtraction of two upward paths which have the same bottom most node, results in

another upward path. After these subtractions, the column j will only have one entry equal to 1 (the entry at row i) and other entries in this column will be 0. Expanding by column j, the determinant does not change except possibly for its sign and thus we can prove unimodularity by induction since we decreased the size of the square sub-matrix by 1.

Therefore the matrix M is totally unimodular and we can apply a theorem of Hoffman and Kruskal [36], which states that for an integral right-hand side vector B, if the matrix for the LP is totally unimodular, then all vertex solutions of the LP are integral. In particular, as there is an optimal solution that is a vertex, we obtain that there is an integer optimum, which completes the proof.

□

Theorem 4.3.5. *UBSMP on Upward Trees can be solved in polynomial time.*

Proof. Since in UBSMP the budges are in $\{0, 1\}$, therefore combining the results from Theorem 3.3.1 and Lemma 4.3.4, proves that UBSMP on Upward Trees can be solved in polynomial time.

□

Chapter 5

Conclusion

We conclude by summarizing the results of this thesis and discussing future work directions for the problems considered.

5.1 Summary

We studied three problems in this thesis: Single-Minded Pricing, Unique Coverage, and Uniform-Budget Single-Minded Pricing which is actually a special case of Single-Minded Pricing. For each problem, we considered that problem in different settings such as on graphs and on geometric objects.

In Chapter 2, we studied the Single-Minded Pricing problem. We introduced approximation algorithms for SMP on geometric objects such as convex objects, unit squares, squares, line rectangles, and rectangles. Then we proved that SMP on Graphs with Bounded Pathwidth is as hard as the Unique Coverage problem.

In Chapter 3, we studied the Unique Coverage problem. We presented dynamic programming algorithms for the Unique Coverage problem on corner rectangles and also on upward trees.

In Chapter 4, we studied the Uniform-Budget Single-Minded Pricing problem. We looked at the relation between UBSMP and the Unique Coverage problem and the results that could be achieved considering this relation. Finally we proved that UBSMP on Upward Trees can be solved exactly in polynomial time.

5.2 Directions for Future Work

Many variations of the Single-Minded Pricing and Unique Coverage problems have not been studied yet. Also, for some interesting cases that have been studied, the complexity statuses of these cases are not settled yet.

For the Single-Minded Pricing problem, there have not been a lot of studies on geometric objects. Perhaps, obtaining a PTAS for SMP on Corner Rectangle would be an interesting start which might

lead to better results for SMP on Rectangles. On the other hand, SMP on graphs is well-studied. The most interesting problem in this area is the Tollbooth problem (SMP on Trees) which its complexity status is not settled yet. On the upper bound side, the best result so far, is an $O(\log n/\log\log n)$-approximation [29] and on the lower bound side, the best result is an APX-hardness originally given by Guruswami *et al.* [33]. Also, most of our results on graphs are dedicated to upward trees. For Upward Tollbooth problem, there is a QPTAS and the question for this problem is whether there exists a PTAS.

The Unique Coverage problem on geometric objects has been mostly studied only for objects of unit size, such as unit squares and unit disks. Getting constant approximation results for Unique Coverage on Rectangles or Squares would be a great start on studying the problem on arbitrary sized objects. For Unique Coverage on graphs, only an $O(\log\log n)$-approximation for trees is available. Settling the complexity on trees would be an interesting case, as well.

Like the Unique Coverage problem, the Uniform-Budget Single-Minded Pricing problem is a problem that has not been studied on graphs or geometric objects extensively. Finding a constant approximation algorithm for UBSMP on trees would be an interesting result and would help a lot towards solving the Tollbooth problem.

Bibliography

[1] Gagan Aggarwal, Tomás Feder, Rajeev Motwani, and An Zhu. Algorithms for multi-product pricing. In *Automata, Languages and Programming*, pages 72–83. Springer, 2004.

[2] Giorgio Ausiello. *Complexity and approximation: Combinatorial optimization problems and their approximability properties*. Springer, 1999.

[3] Maria-Florina Balcan and Avrim Blum. Approximation algorithms and online mechanisms for item pricing. In *Proceedings of the 7th ACM Conference on Electronic Commerce*, pages 29–35. ACM, 2006.

[4] Maria-Florina Balcan, Avrim Blum, TH Hubert Chan, and MohammadTaghi Hajiaghayi. A theory of loss-leaders: Making money by pricing below cost. In *Internet and Network Economics*, pages 293–299. Springer, 2007.

[5] Boaz Barak. A probabilistic-time hierarchy theorem for slightly non-uniform algorithms. In *Randomization and approximation techniques in computer science*, pages 194–208. Springer, 2002.

[6] Patrick Briest. Towards hardness of envy-free pricing. *Electronic Colloqium on Computational Complexity, Report*, (150), 2006.

[7] Patrick Briest. Uniform budgets and the envy-free pricing problem. In *Automata, Languages and Programming*, pages 808–819. Springer, 2008.

[8] Patrick Briest, Parinya Chalermsook, Sanjeev Khanna, Bundit Laekhanukit, and Danupon Nanongkai. Improved hardness of approximation for stackelberg shortest-path pricing. In *Internet and Network Economics*, pages 444–454. Springer, 2010.

[9] Patrick Briest, Luciano Guala, Martin Hoefer, and Carmine Ventre. On stackelberg pricing with computationally bounded customers. *Networks*, 60(1):31–44, 2012.

[10] Patrick Briest and Piotr Krysta. Single-minded unlimited supply pricing on sparse instances. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 1093–1102. ACM, 2006.

[11] Parinya Chalermsook, Julia Chuzhoy, Sampath Kannan, and Sanjeev Khanna. Improved hardness results for profit maximization pricing problems with unlimited supply. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 73–84. Springer, 2012.

[12] Parinya Chalermsook, Khaled Elbassioni, Danupon Nanongkai, and He Sun. Geometric pricing: How low dimensionality helps in approximability. *arXiv preprint arXiv:1202.2840*, 2012.

[13] Parinya Chalermsook, Shiva Kintali, Richard Lipton, and Danupon Nanongkai. Graph pricing problem on bounded treewidth, bounded genus and k-partite graphs. *arXiv preprint arXiv:1203.1940*, 2012.

[14] Ning Chen and Xiaotie Deng. Envy-free pricing in multi-item markets. In *Automata, Languages and Programming*, pages 418–429. Springer, 2010.

[15] Ning Chen, Xiaotie Deng, Paul Goldberg, Jinshan Zhang, et al. On revenue maximization with sharp multi-unit demands. *arXiv preprint arXiv:1210.0203*, 2012.

[16] Ning Chen, Arpita Ghosh, and Sergei Vassilvitskii. Optimal envy-free pricing with metric substitutability. *SIAM Journal on Computing*, 40(3):623–645, 2011.

[17] Maurice Cheung and Chaitanya Swamy. Approximation algorithms for single-minded envy-free profit-maximization problems with limited supply. In *Foundations of Computer Science, 2008. FOCS'08. IEEE 49th Annual IEEE Symposium on*, pages 35–44. IEEE, 2008.

[18] Marek Cygan, Fabrizio Grandoni, Stefano Leonardi, Marcin Pilipczuk, and Piotr Sankowski. A path-decomposition theorem with applications to pricing and covering on trees. In *Algorithms–ESA 2012*, pages 349–360. Springer, 2012.

[19] Erik D Demaine, Uriel Feige, MohammadTaghi Hajiaghayi, and Mohammad R Salavatipour. Combination can be hard: Approximability of the unique coverage problem. *SIAM Journal on Computing*, 38(4):1464–1483, 2008.

[20] Khaled Elbassioni. A qptas for-envy-free profit-maximizing pricing on line graphs. In *Automata, Languages, and Programming*, pages 513–524. Springer, 2012.

[21] Khaled Elbassioni, Rajiv Raman, Saurabh Ray, and René Sitters. On profit-maximizing pricing for the highway and tollbooth problems. In *Algorithmic Game Theory*, pages 275–286. Springer, 2009.

[22] Khaled Elbassioni, Rajiv Raman, Saurabh Ray, and René Sitters. On the complexity of the highway problem. *Theoretical Computer Science*, 460:70–77, 2012.

[23] Khaled Elbassioni, René Sitters, and Yan Zhang. A quasi-ptas for profit-maximizing pricing on line graphs. In *Algorithms–ESA 2007*, pages 451–462. Springer, 2007.

[24] Paul Erdös and John L Selfridge. On a combinatorial game. *Journal of Combinatorial Theory, Series A*, 14(3):298–301, 1973.

[25] Thomas Erlebach and Erik Jan van Leeuwen. Approximating geometric coverage problems. In *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1267–1276. Society for Industrial and Applied Mathematics, 2008.

[26] Uriel Feige. Relations between average case complexity and approximation complexity. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 534–543. ACM, 2002.

[27] Cristina G Fernandes, Carlos E Ferreira, Álvaro JP Franco, and Rafael Schouery. The unit-demand envy-free pricing problem. *arXiv preprint arXiv:1310.0038*, 2013.

[28] Amos Fiat and Amiram Wingarten. Envy, multi envy, and revenue maximization. In *Internet and Network Economics*, pages 498–504. Springer, 2009.

[29] Iftah Gamzu and Danny Segev. A sublogarithmic approximation for highway and tollbooth pricing. In *Automata, Languages and Programming*, pages 582–593. Springer, 2010.

[30] Michael R Garey and David S Johnson. *Computers and intractability*. freeman San Francisco, 1979.

[31] Fabrizio Grandoni and Thomas Rothvoß. Pricing on paths: A ptas for the highway problem. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 675–684. SIAM, 2011.

[32] Alexander Grigoriev, Joyce van Loon, René Sitters, and Marc Uetz. How to sell a graph: Guidelines for graph retailers. In *Graph-Theoretic Concepts in Computer Science*, pages 125–136. Springer, 2006.

[33] Venkatesan Guruswami, Jason D Hartline, Anna R Karlin, David Kempe, Claire Kenyon, and Frank McSherry. On profit-maximizing envy-free pricing. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1164–1173. Society for Industrial and Applied Mathematics, 2005.

[34] Venkatesan Guruswami and Luca Trevisan. The complexity of making unique choices: Approximating 1-in-k sat. In *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques*, pages 99–110. Springer, 2005.

[35] Jason D Hartline and Vladlen Koltun. Near-optimal pricing in near-linear time. In *Algorithms and Data Structures*, pages 422–431. Springer, 2005.

[36] Alan J Hoffman and Joseph B Kruskal. Integral boundary points of convex polyhedra. In *50 Years of Integer Programming 1958-2008*, pages 49–76. Springer, 2010.

[37] Sungjin Im, Pin-Yan Lu, and Ya-Jun Wang. Envy-free pricing with general supply constraints for unit demand consumers. *Journal of Computer Science and Technology*, 27(4):702–709, 2012.

[38] Takehiro Ito, Shin-ichi Nakano, Yoshio Okamoto, Yota Otachi, Ryuhei Uehara, Takeaki Uno, and Yushi Uno. A 4.31-approximation for the geometric unique coverage problem on unit disks. In *Algorithms and Computation*, pages 372–381. Springer, 2012.

[39] Takehiro Ito, Shin-Ichi Nakano, Yoshio Okamoto, Yota Otachi, Ryuhei Uehara, Takeaki Uno, and Yushi Uno. A polynomial-time approximation scheme for the geometric unique coverage problem on unit squares. In *Algorithm Theory–SWAT 2012*, pages 24–35. Springer, 2012.

[40] Gwenaë Joret. Stackelberg network pricing is hard to approximate. *Networks*, 57(2):117–120, 2011.

[41] Rohit Khandekar, Tracy Kimbrel, Konstantin Makarychev, and Maxim Sviridenko. On hardness of pricing items for single-minded bidders. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 202–216. Springer, 2009.

[42] Subhash Khot. On the power of unique 2-prover 1-round games. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 767–775. ACM, 2002.

[43] Guy Kortsarz, Harald Räcke, and Rajiv Raman. On the tollbooth problem. 2012.

[44] Michael Mitzenmacher and Eli Upfal. *Probability and computing: Randomized algorithms and probabilistic analysis*. Cambridge University Press, 2005.

[45] Preyas Popat and Yi Wu. On the hardness of pricing loss-leaders. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 735–749. SIAM, 2012.

[46] Prabhakar Raghavan. Probabilistic construction of deterministic algorithms: approximating packing integer programs. *Journal of Computer and System Sciences*, 37(2):130–143, 1988.

[47] Carsten Thomassen. The graph genus problem is np-complete. *Journal of Algorithms*, 10(4):568–576, 1989.

[48] Erik Jan van Leeuwen. *Optimization and approximation on systems of geometric objects*. 2009.

[49] Yi Wu. Pricing loss leaders can be hard. *Journal of Computer Science and Technology*, 27(4):718–726, 2012.