University of Alberta

Approximation Algorithms for some Min-max Vehicle Routing Problems

by

Amin Jorati

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

Abstract

Vehicle routing problems are optimization problems that deal with the location and routing of vehicles. A set of clients based in different locations need to be served by a fleet of vehicles. The clients and vehicle depots are modeled as being placed on the vertices of a graph and the distances between them as a metric. Thus, a solution to a vehicle routing problem corresponds to covering the graph using a number of subgraphs, each denoting the route of a vehicle. In this thesis, we consider min-max vehicle routing problems, in which the maximum cost incurred by the subgraph corresponding to each vehicle is to be minimized. We study two types of covering problems and present new or improved approximation algorithms for them.

In Chapter 2, we study the rooted and unrooted variants of min-max k-tour cover problem. Given a metric (V, c) and a number k, a set of tours $\tau_1, \ldots, \tau_k$ in G is called a k-tour cover, if they cover all the vertices of G, i.e. $V = \cup_{i=1}^{k} V(\tau_i)$. The unrooted min-max k-tour cover problem is that of giving a k-tour cover of G where the maximum cost of a tour under the metric c is minimized. Analogously, in the rooted version, we are given a subset of vertices R of size k and each tour of the k-tour cover is required to be rooted at a distinct vertex in R. We improve on the approximation ratios of these problems by giving a $(16/3 + \epsilon)$-approximation algorithm for the unrooted min-max k-tour cover problem and a $(7 + \epsilon)$-approximation algorithm for rooted version.

In Chapter 3, we study the unrooted min-max k-star cover problem. Given a metric (V, c) and a number k, a set of stars $S_1, \ldots, S_k$ in G is called a k-star cover, if they cover all the vertices of G, i.e. $V = \cup_{i=1}^{k} V(S_i)$. The rooted min-max k-star cover problem is that of giving a k-star cover of G where the maximum cost of a star under the metric c is minimized. We improve on the approximation ratio when the number of stars k is slightly violated, i.e. bi-criteria approximations and present an $(O(1/\epsilon), 1 + \epsilon)$ bi-criteria approximation. We also study the problem on more restricted metrics, namely the line metric and Euclidean metric. For the problem on the line metric, we present a QPTAS, and for the problem on the line metric in the special case that the stars are non-crossing, we present a PTAS for the unrooted min-max k-star cover problem. Then, we explore the possibility of Polynomial time approximation schemes on the Euclidean metric. We rule out this possibility by giving an APX-hardness reduction.

Table of Contents

1 Introduction **1**
1.1 Motivation . . . 1
1.1.1 Vehicle Routing Problems . . . 2
1.1.2 Covering problems in Vehicle Routing . . . 3
1.2 Preliminaries and Notation . . . 4
1.2.1 Graph Theory . . . 4
1.2.2 Approximation Algorithms for NP-hard Optimization Problems . . . 5

2 Min-max k-Tour Cover **9**
2.1 Problem Definitions . . . 9
2.2 Background . . . 10
2.3 Rooted k-tour cover algorithm . . . 11
2.4 Unrooted k-tour cover algorithm . . . 15

3 Min-max k-Star Cover **19**
3.1 Problem Definitions . . . 19
3.2 Background . . . 20
3.3 Bi-criteria Approximations . . . 22
3.4 Line Metric . . . 23
3.4.1 QPTAS . . . 24
3.4.2 PTAS for the line metric when Stars are non-crossing . . . 30
3.5 Euclidean Metric . . . 31
3.5.1 APX-hardness . . . 31

4 Conclusion **35**
4.1 Summary . . . 35
4.2 Future Directions . . . 35

Bibliography **37**

List of Tables

2.1 The currently best approximation ratios for different versions of the k-tour cover problem when the depots are unspecified (Unrooted), there is a single depot or there are multiple depots on the network, with respect to two different objective functions, i.e. (i) minimizing the sum of the cost of all the tours, and (ii) minimizing the maximum cost incurred by each vehicle; α_{TSP} is the best approximation ratio for TSP. 11

List of Figures

2.1 Algorithm for the rooted k-Tour cover problem 12
2.2 (a) T_v, the subtree rooted at v, and all its children, $u_1, \ldots, u_{d(v)-1}$. (b) T_e, a degree-1 subtree of v that is formed by its edge to its child u_i, and T_{u_i}. 13
2.3 Algorithm for the unrooted k-tour cover problem 16

3.1 A degree-3 vertex v and its three neighbors w, x, y are shown. Each edge is replaced with a path of length 4. 32
3.2 A star cover of cost at most 3 in H. The dominating set of G includes v and y, but not w, x, or z. The dotted lines show the arms of the star cover. The middle node on the path $p(\{v, w\})$ is a singleton stars. 33

Chapter 1

Introduction

Optimization problems are ubiquitous in real world, where applications call for a solution that satisfies certain hard constraints and budgets dictated by the environment, yet striving to minimize the cost. In this thesis, we look at a few optimization problems that arise in the transportation industry and concerns locating and routing of vehicles. Since these problems fall in a category of intractable problems, either of optimality or running time needs to be compromised. We will propose algorithms for the problems introduced that run in a reasonable amount of time, yet produce solutions that are guaranteed to be close to the optimal value.

1.1 Motivation

A diverse class of optimization problems arise from logistics, transportation, city planning, and routing. Consider the following two examples.

Example 1 *Consider a hospital that owns a number of ambulances which could be located in certain places throughout the city. There is usually an estimated number of requests to transfer patients between different medical facilities during the day. The hospital needs to locate the ambulances in certain stations such that each ambulance delivers its patient as soon as possible to the required medical facility. This decision has to be taken respecting the requirements that all patients need to be transferred and that the hospital can only own a certain number of ambulances and cannot afford to buy any new ambulances.*

We formalize and study some variations of the above problem, called the Tour Cover Problem in Chapter 2. Another example concerning distribution of goods is as follows.

Example 2 *Assume you own a company that produces a certain product and delivers it from its warehouses to its customers. The delivery is performed by the transportation department of the company that has a fleet of vehicles. Each customer location demands a truckload of the product each day, so after serving each customer, a truck needs to get back to a warehouse to restock before serving another customer. The company is expanding to a new area and has secured funding for* *a specific number of warehouses and vehicles. The decision the company faces is to choose the location of the warehouses and assign the customer locations to warehouses such that the last driver checks out as early as possible, to minimize the overtime payment the company has to endure over the course of years.*

We formalize and study some variations of the above problem, called the Star Cover Problem in Chapter 3. The above are examples of a class of problems called Vehicle Routing Problems (VRP).

1.1.1 Vehicle Routing Problems

Vehicle Routing Problems (VRP) is a rich class of problems, that are extensively studied in Operations Research [40, 22, 12]. They arise from practical applications in transportation and deal with the assignment of routes, for purposes such as the delivery of goods from certain depots to the customers in various locations. These routes will then be assigned to a fleet of vehicles. The route assignment has to be done subject to operation constraints imposed by the vehicles, the demands of the customers, and the underlying structure of the transportation network they operate in. The goal is to find such routes subject to the restrictions that satisfy the demands and minimize the cost, such as the total mileage traveled. This research area originated by a problem introduced in [13], for finding the optimal route for a truck delivering gasoline from a depot to a set of customers. The field subsequently flourished by trying to answer different variations of the original question. Some of the variations studied are described in the list below:

- *Objective function*: Different objectives are of practical relevance in different settings. Some of the more common ones include: Total mileage traveled by all the vehicles, the maximum distance (and hence time) traveled by a single vehicle, the average time that each customer needs to wait before it is served, and the minimum number of vehicles to service all the customers subject to the operational constraints.
- *Network Symmetry*: The underlying transportation network may or may not be symmetric, i.e. the cost of visiting customer b right after customer a may or may not be equal to the cost of the reverse action, i.e. visiting customer a right after customer b.
- *Processing times*: The vehicles might need to stay at each customer location for a certain amount of time processing the request of the customer.
- *Depots*: Depending on the underlying facilities, there might be a single depot or multiple depots located in the transportation network, or the location of the depots might be part of the decision procedure.
- *Capacity*: Each vehicle has a capacity restricting the amount of goods that it can deliver to customers, and thus the number of customers it can satisfy in one trip.

- *Heterogeneous Fleet of Vehicles*: The fleet of vehicles might be homogeneous or heterogeneous, i.e. have differing specification, such as non-uniform capacity or speed.
- *Time Windows*: Each customer v may only be serviced in a certain time interval $[R(v), D(v)]$.

The above classification gives a glimpse into the richness and variety of problems studied in the area of Vehicle Routing. In this thesis, we focus on the following specific subclass of problems. Objective functions: We consider the maximum cost each vehicle incurs as the objective function. Symmetry: We only consider transportation networks where the cost is symmetric. Depots: We consider variations where either multiple depots are given or where a given number of depots need to be established in the network. Other constraints such as the heterogeneity of fleet, etc. are not taken into account, and left as future work.

1.1.2 Covering problems in Vehicle Routing

In this section, we present a specific subclass of vehicle routing problems. In covering problems, each location in the network has a demand for some commodities which need to be delivered by a fleet of k vehicles for a given k. The vehicles will start from a depot and have to return to their designated depot after serving the customers. Depending on the logistics and facilities of the underlying transportation network, we might be able to build k depots in different locations on the network, the vehicles may all be stationed in one existing central depot (The single depot case) or in k potentially different designated depots on the network (Multiple depots case). In any case, each vehicle needs to return to the depot it started from.

The goal would then be to assign a depot for each vehicle that does not have one designated. Each vehicle will thus be assigned to a route that it needs to traverse in order to deliver the goods to the customers on the route. All customers need to be covered by some vehicle in their respective routes. In doing so, the objective is to minimize the cost. The cost each vehicle incurs on a certain route will be different depending on the application. Several different objective functions arise in different scenarios that we describe below:

1. *Min-Sum*: Minimizing the total mileage traveled by all the vehicles. This is the most natural objective function and comes up when we want to minimize the total fuel used by the vehicles or the the total wage of the crew operating the vehicles.
2. *Min-Max*: Minimizing the latest time by which a vehicle is returned to its original depot. This objective function arises when we want to close the facilities at the depot as soon as possible to minimize the after-hour costs of overtime and operating of the facilities.
3. *Min-Latency*: The latency of each customer is the time it gets served by some vehicle. This objective function comes up when we want to maximize customer satisfaction, thus minimize the average service time of customers.

In this thesis, we focus on the Min-Max objective function defined above. In order to formalize the above problems, we formulate them in the language of graph theory as required by the rigorous mathematical analysis of algorithms. For the sake of clarity, we review the relevant concepts of graph theory and fix our notation.

1.2 Preliminaries and Notation

In this section, we review the basics of graph theory and approximation algorithms and present the notation used throughout the thesis. The underlying network of depots and customers can be modeled as a graph with an underlying cost function.

1.2.1 Graph Theory

An undirected graph $G = (V, E)$ is an ordered pair of sets, where V (also denoted $V(G)$) is the set of *vertices* (or *nodes*), and E (also denoted as $E(G)$) is the set of *edges* of V. The edge set consists of edges that are two-element subsets of V. In the applications we consider, we take the set V to be the set of all locations, i.e. the depots and the customer locations. For each pair of locations $i, j \in V$, we will have an edge $e = \{i, j\}$ that denotes the connection between the two locations. Let c_{ij} denote the cost between i and j, i.e. the distance between the two locations or the time it takes to get from i to j. In the applications, we consider, the cost function $c : V \times V \to \mathbb{Q}^+$ is a metric, defined below:

Metric: A function c over V, $c : V \times V \to \mathbb{Q}^+$ is a *metric* if it satisfies the following properties:

1. $c_{ii} = 0, \forall i \in V$.
2. $c_{ij} = c_{ji}, \forall i, j \in V$ (Symmetric Property)
3. $c_{ij} + c_{jk} \geq c_{ik}, \forall i, j, k \in V$ (Triangle Inequality).

The triangle inequality can be assumed, without loss of generality, since in every solution, in going from i to k, we take the shortest path between them, thus incurring the minimum cost, which is at most the cost incurred by going through j. Note that the cost function c of edges of a graph G being metric, also implicitly implies that G is a complete graph, i.e. for any $i, j \in V$, we have $\{i, j\} \in E$. Here, we introduce a few special metrics.

- *Line Metric.* (V, l) is a line metric, if there is a mapping $f : V \to \mathbb{Q}$, such that $l(v_i, v_j) = |f(v_i) - f(v_j)|$, for every $i, j \in V$.
- 2-*Dimensional Euclidean Metric.* (V, l) is a 2-Dimensional Euclidean metric (from here on called a Euclidean metric for short), if there is a mapping $f : V \to \mathbb{Q}^2$, such that for $(x_i, y_i) = f(v_i)$, we have $l(v_i, v_j) = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2}$ for every $i, j \in V$, i.e. the Euclidean distance of the corresponding points on the plane.

- *Tree Metric.* (V, l) is a tree metric if there exists a tree T and a mapping $f : V \to V(T)$, such that for every $u, v \in V, l(u, v)$ is equal to the distance between $f(u)$ and $f(v)$ on the tree T.
- *Doubling Metrics.* The *doubling dimension* of a metric space (V, l) is the smallest $\delta > 0$, such that every ball of radius r (for any $r \in \mathbb{Q}^+ \cup \{0\}$) can be covered by 2^δ balls of radius $r/2$. A metric is called a doubling metric, if its doubling dimension is a constant. Doubling metrics were first introduced in [24].

The underlying transportation network is thus modeled as a graph $G = (V, E)$ with a metric cost function over the edges $c : V \times V \to \mathbb{Q}^+$. In the formal statement of the problems, we stick with the graph theoretic definition for the sake of technical clarity. Here are a few related concepts. More in-depth coverage of these concepts can be found in [7].

- **Subgraph**. A graph H is a *subgraph* of a graph G, if $V(H) \subseteq V(G)$ and $E(H) \subseteq V(G)$.
- **Matching**. A *matching* M of a graph G is a subset of its edges $M \subseteq E(G)$ such that no two edges in M are adjacent. A *maximum* matching is one with the maximum cardinality. A *perfect* matching is one where each vertex is covered once.
- **Walk**. A *walk* W in a graph G is an alternating sequence of vertices and edges of G, i.e. $W = v_0 e_1 v_1 \dots v_{l-1} e_l v_l$, such that $e_i = \{v_{i-1}, v_i\}$, for $1 \leq i \leq l$.
- **Tour**. A *tour* τ in a graph G is a spanning closed walk that traverses each vertex at least once.
- **Star**. A graph $S = (r, L)$ is called a *star*, if $V(S) = \{r\} \cup L$, $L = \{l_1, \dots, l_n\}$, $E(S) = \{\{r, l_i\} | \forall 1 \leq i \leq n\}$. The vertex r is called the *root*, and the vertices L are the *leaves* of the star.
- **Tour Cover**. A *tour cover* of a graph G is a set of tours $\tau_1, \dots, \tau_k$ such that each vertex of $V(G)$ is visited by at least one tour. It is also called a *k-Tour Cover* where k is the number of tours.
- **Star Cover**. A *star cover* of a graph G is a set of stars $S_1, \dots, S_k$ such that $V(S_1), \dots, V(S_k)$ is a partition of of $V(G)$.

1.2.2 Approximation Algorithms for NP-hard Optimization Problems

Most combinatorial optimization problems in general and vehicle routing problems in particular are NP-hard, thus widely believed to be intractable [20]. The Operations Research community has long studied these problems developing exact algorithms by means of heuristics, or putting major restrictions on the size of the problems that can be solved in a reasonable amount of time. The field of approximation algorithms on the other hand deals with the intractability of NP-hard optimization problems in a different way and provides efficient algorithms that produce suboptimal solutions with a guaranteed bound on the quality of the solution. The guarantee is on the worst case error by which the produced solution deviates from the value of the optimal solution. The error bound is usually multiplicative with respect to the optimal value. In a few cases, additive approximation have been given. This bound is referred to as *the approximation ratio*, as defined below.

Approximation Ratio. For a minimization problem Π, an algorithm A has an approximation ratio of α, if for any given instance I of Π, the algorithm A produces a feasible solution for I that has cost $A(I) \leq \alpha \cdot \mathrm{OPT}_{\Pi}(\mathrm{I})$, where $\mathrm{OPT}_{\Pi}(\mathrm{I})$ is the optimal value of Π on instance I.

The approximation ratio is a function of the size of the instance $n = size(I)$. An approximation ratio of $\alpha = 1$ is equivalent to solving the problem exactly. Thus, for any NP-hard problem, there is no 1-approximation, assuming P $\neq$ NP. The closer α is to 1, the better the approximation. One may wonder if it is possible to approximate NP-hard problems to any degree close to 1. It turns out in much the same way that decision version of an optimization problems can be NP-hard, α-approximating a problem can also be NP-hard for a certain α. Results of the second type rely on a different type of machinery from classical NP-hardness reductions, including such notions as approximation-preserving reductions and the celebrated probabilistic characterization of the class NP known as PCP (see [41, 42] and the references therein for more information on this). Each NP-hard optimization problem may only be approximable up to a certain threshold function. The *approximability threshold* of a problem Π is a function α_{Π}, for which we have an α_{Π}-approximation for Π and the problem of $(\alpha_{\Pi} - \epsilon)$-approximating Π is NP-hard, for any constant $\epsilon > 0$. Interestingly, different NP-hard optimization problems behave very differently in terms of their approximability and their approximability thresholds could be anywhere from $(1 + \epsilon)$ for all $\epsilon > 0$ (called a PTAS, defined below) to some polynomial function in n.

Polynomial Time Approximation Scheme (PTAS). For a minimization problem Π, a PTAS is a class of algorithms $A(\epsilon)$, such that for each given $\epsilon > 0$, produces a solution of cost at most $(1+\epsilon) \cdot \mathrm{OPT}_{\Pi}(\mathrm{I})$, for any instance I. The running time of the algorithm must polynomial in $size(I)$, but may be exponential in $\frac{1}{\epsilon}$.

Approximation algorithms are a trade-off between optimality and running time. Thus, a major requirement for an approximation algorithms is to run in a reasonable amount of time. This usually means in time polynomial in the size of the input. This requirement is slightly relaxed by considering algorithms that run in *quasi-polynomial time*. An algorithm running in quasi-polynomial time requires $O(2^{O(\mathrm{Polylog}(n))})$ time. Note that a Quasi-polynomial running time is slower than polynomial time, but faster than exponential time. In other words, quasi-polynomial time is super-polynomial and sub-exponential. The significance of quasi-polynomial algorithms comes from a hypothesis in complexity theory called the *Exponential Time Hypothesis* (ETH). The ETH, first formalized by [28], says that 3SAT cannot be solved in sub-exponential time, i.e. time $2^{o(n)}$. Note that this hypothesis is strong, and implies P $\neq$ NP.

Thus, if there exists an algorithm that runs in quasi-polynomial for a problem, it gives a strong evidence that the problem might be solvable in polynomial time. In particular, if there exists an algorithm that achieves an approximation ratio α in quasi-polynomial time, it means that the problem is approximable to a factor of α, unless NP $\subseteq O(2^{O(\text{Polylog}(n))})$. The ETH states that the latter is a sound assumption. This motivates the definition of a QPTAS, similar to a PTAS as follows.

Quasi-Polynomial Time Approximation Scheme (QPTAS). For a minimization problem Π, a QPTAS is a class of algorithms $A(\epsilon)$, such that for each given $\epsilon > 0$, produces a solution of cost at most $(1+\epsilon) \cdot \text{OPT}_\Pi(\text{I})$, for any instance I, in time *quasi-polynomial* in $size(I)$.

The goal of the field of approximation algorithms is pinpointing the approximability threshold of NP-hard optimization problems. This is achieved by two types of attacks on each problem.

1. *Upper bounds on the approximation ratio.* First, to provide an algorithm that produces a feasible solution that attains a certain approximation ratio.

2. *Lower bounds on the approximation ratio.* To prove that approximating a problem to a certain ratio is intractable, i.e. unlikely under widely believed complexity-theoretic assumptions, such as P $\neq$ NP.

Such results, give in turn upper and lower bounds on the approximability ratio of the problem. We mainly focus on the first type of results and most of the results presented in this thesis fall in that category, except for the hardness result in Section 3.5, which is of the second type. For some NP-hard optimization problems, tight approximations have been established and for others, the upper and lower bounds on the approximability remain wide open. The field of approximation algorithms is replete with open problems and has matured over the years. An overview of the field as we know it today has been given in recent books [42, 41, 25]. We will illustrate the notion of approximation ratio on the Traveling Salesman Problem below, which is one of the most famous problems in combinatorial optimization and serves as the most fundamental problem underlying the vehicle routing problems.

The Traveling Salesman Problem (TSP). Given a complete graph $G = (V, E)$ with a cost function $c : E \to \mathbb{Q}^+$ on the edges, find an ordering of the vertices, such that the resulting tour that traverses V in this order is of minimum cost.

TSP, in its full generality remains NP-hard to approximate to any polynomial time computable function as can be seen by a reduction from the Hamiltonian Cycle problem. Thus, attention has been restricted to instances, where the cost function satisfies the triangle inequality, i.e. is a metric. The instance of TSP, where the cost function is metric is referred to as metric TSP. In most applications in VRP, the best way to go from a vertex i to j is to take the shortest path between them, in case that is different from the direct route between them. Thus, TSP reduces to the case where we consider the shortest path metric graph. Since we exclusively deal with the metric TSP problem, we refer to the metric TSP simply as TSP. Approximation algorithms for TSP have been based on two lower bounds, namely the minimum spanning tree (MST) and the minimum perfect matching. This results in a $3/2$-approximation algorithm for TSP, that we mention below. Both lower bounds and the approximation algorithm are common building blocks in many vehicle routing problems.

Theorem 1 (Christofides [10]) *There is a $3/2$-approximation for TSP.*

Proof: Consider a Minimum spanning tree T of the graph G. The cost of T is a lower bound to the cost of optimal TSP. This can be seen by removing an edge from TSP; the remaining path is a spanning tree of G. Thus, the cost of a minimum spanning tree is less than the cost of the optimal TSP tour, denoited by OPT. Duplicating all the edges of the MST gives a graph of cost at most $2 \cdot \mathrm{OPT}$, where the degree of all vertices is even. Thus, by traversing the edges of this graph and shortcutting over already-visited vertices, we can find a spanning tour, without increasing the cost. The MST lower bound gives an approximation ratio of 2. We need another lower bound to improve on this. Consider any even subset of the vertices of G, the optimal TSP incurs a tour of cost no more on these vertices by shortcutting over all the other vertices. The edges of this tour are the union of two perfect matchings, each consisting of alternating edges. Hence, the cost of the minimum perfect matching over the nodes is at most half the cost of the optimal TSP. Now, let M be the minimum cost perfect matching on the set of odd vertices in T, where T is an MST. The union of these two subgraphs $T \cup M$ induces a graph with even degrees and costs at most $3/2 \cdot \mathrm{OPT}$. We can traverse the edges of this graph and find a tour of cost no more by shortcutting over vertices we have already visited. ■

A closely related problem is the TSP-Path problem, where instead of a tour, we seek the minimum cost path. Hoogeveen [27] in a parallel result to Christofides, gave a $5/3$-approximation for the TSP-Path problem. This ratio has been improved to $\frac{1+\sqrt{5}}{2}$ [1] very recently. Despite the fact that TSP has been extensively studied, little progress has been made in terms of its approximability in the past 37 years and the $3/2$ ratio has been only slightly shattered for the special case of graphic TSP very recently [35, 21]. On the other hand, the best lower bound on the approximability threshold of this problem is $\frac{129}{128}$ [37]. Closing this gap is a major open question in its own right.

Chapter 2

Min-max k-Tour Cover

In this chapter, we consider the tour cover problems, that is the problem of covering the vertices of a graph using a number of tours. The tour cover problems are some of the most natural variants of vehicle routing problems studied. In all real world applications, the vehicles need to get back to their designated depots for freight and fuel recharge, maintenance and administration. Thus, the route traversed by each vehicle will be a tour. On the other hand, these problems generalize the celebrated Traveling Salesman Problem. Thus, tour cover problems are also of theoretical interest. Consider the case, when there is a single vehicle that needs to cover all the customers. The path traversed by this vehicle gives a single spanning tour, minimizing which is the TSP mentioned above. TSP is one of the most important problems in combinatorial optimization that has attracted a lot of attention in the Operations Research community, as reflected in a number of books devoted to the subject, e.g. [11, 2, 32]. Subsequently, it has also been subject to much research from the standpoint of approximation algorithms, starting with [10], and culminating in a recent stream of improvements, including [21, 35, 36]. Thus, tour cover problems are generalizations of TSP, and thus of great practical and theoretical significance.

2.1 Problem Definitions

Here, we formally define the two variants of tour cover problem that we consider in this thesis. Consider Example 1 in Chapter 2. The case where the hospital has already made the decision on the locations to build the medical facilities to have the ambulances stationed at. We can model the location of the patients and the medical facilities as the underlying network by the graph G and represent the distances between locations by the cost function c. The decision problem will be, given the facility locations as roots, to decide which clients to assign to each ambulance stationed at a facility. The goal is to minimize the latest time an ambulance gets back to its assigned facility and delivers all the patients. The problem can be formalized as the *rooted* tour cover problem as follows.

Problem 1 **(Rooted k-Tour Cover)** *Given a graph (V, E) with a metric cost function on the edges $c : E \to \mathbb{Q}^+$, and a set of roots $R \subseteq V$, find a set of $k = |R|$ tours $\{\tau_i\}$, each rooted at a distinct* *root in R that cover the graph, i.e. $\cup_i V(\tau_i) = V$. The objective function is to minimize the cost of the heaviest tour, i.e.* $\min \max_i c(\tau_i)$.

Analogously, we can define the variant, in which the location of the facilities are not specified and is part of the decision problem. We call this variant of the problem, the *unrooted* k-tour cover problem.

Problem 2 (Unrooted k-Tour Cover) *Given a graph $G = (V, E)$ with a metric cost function on the edges $c : E \to \mathbb{Q}^+$, and an integer k, find a collection of k tours $\tau_1, \ldots, \tau_k$ that cover V, i.e. $\cup_{i=1}^k V(\tau_i) = V$. The objective function is to minimize the cost of the largest tour, i.e.* $\min \max_i c(\tau_i)$.

2.2 Background

The state-of-the-art in terms of approximation ratio of tour cover problems for the min-sum and min-max objective functions is summarized in Table 2.1, for different root structures of the network. In each of the covering problems, determining the assignment of vertices to their respective roots will reduce the problem into disjoint instances, which then can be solved independently of others. Each such independent instance will be the problem of assigning a route to all the vertices that have been assigned to a certain root. This latter problem is an an instance of TSP, for which $3/2$-approximation exists. Thus, if the assignment of vertices to the roots can be done optimally, the $3/2$-approximation of TSP would carry over to the min-max k-tour cover problem. To understand the hardness of this assignment problem, we consider the bin packing problem, defined below.

The Bin Packing Problem. Given n items with sizes $s_1, \ldots, s_n$ and a bin capacity B, pack the items into a minimum number of bins of capacity B.

Another version of the bin packing problem that is of relevance to the problems that we address in this thesis is the Min-max version: Given n items with sizes $s_1, \ldots, s_n$ and m identical bins, find a minimum size B for bins and a packing of all the items into the m identical bins of size B. Note the similarity between bins and vehicles. The bin packing problem is one of the basic problems in combinatorial optimization. Both variations above are NP-hard, the second one is similar to the tour cover problems in this section. In the bin packing problem, the cost incurred by the items in a bin is a linear function of the costs of the items, whereas in the tour cover problem there is an underlying graph defining the costs.

For the rooted k-tour cover problem, Even et al. [16] give a $(4 + \epsilon)$-approximation for the corresponding tree cover problem, which by the doubling and shortcutting technique, gives an $(8 + \epsilon)$-approximation for the tour cover problem, which they call the *nurse location problem.* They suggest the direct approach to the tour cover problem as an open problem, which might lead to a better approximation ratio. We improve the ratio down to $(7 + \epsilon)$ in Algorithm 2.1 in section 2.3. This result was also independently obtained by [44].

For the unrooted k-tour cover problem, Xu et al. [44] gave a 6-approximation . Independently, Khani et al. [31] give a 3-approximation for the closely related tree cover version of the problem, which implies a 6-approximation by duplicating and shortcutting edges. In section 2.4, we introduce algorithm 2.3 that improves the approximation ratio to $16/3 + \epsilon$.

2.3 Rooted k-tour cover algorithm

In this section, we give an algorithm for the rooted k-tour cover problem, depicted in Figure 2.1. The algorithm takes an upper bound on the value of the optimal solution $B \geq \mathrm{OPT}$ and will produce a solution whose cost, we can bound with respect to B. We can find such a value $B \leq (1 + \epsilon) \cdot \mathrm{OPT}$ by doing a binary search in the range of possible values for the value of optimal in polynomial time and by running the algorithm on each such value and returning the best answer. This will give a similar guarantee on the value of an optimal solution. It remains to give an algorithm that produces a solution with a guaranteed bound on B, when it is given such a value that is an upper bound on the optimal value, i.e. $B \geq \mathrm{OPT}$. The algorithm has been described in Figure 2.1 in detail and works as follows. The algorithm cuts edges of cost more than $B/2$. If this results in more than one connected component, the different components will be considered as separate instances of the tour cover problem and solved independently. Consider a vertex $v \in V$. In the optimal solution, v will be covered by a tour rooted at some root r. The tour consists of two paths from r to v; the lighter of these two paths has cost at most $\mathrm{OPT}/2$. So, we will have $c(v, r) \leq \mathrm{OPT}/2$. If edges of cost greater than $B/2 \geq \mathrm{OPT}/2$ are deleted from the graph, each vertex will remain in the same connected component as its corresponding root in the optimal solution. So, without loss of generality we can solve the problem in each connected component separately.

In each connected component, the algorithm finds a minimum spanning tree. Then this tree is split away into smaller trees of the appropriate size as explained in Lemma 3 with $\lambda = 3B/2$. Then, a matching is found between the set of split trees and potential roots that could cover them. A root can cover a tree it the cost of the smallest edge between the root and the tree is at most $B/2$. This has been formalized by the bipartite graph G' in algorithm 2.1. The existence of this matching has been

Table 2.1: The currently best approximation ratios for different versions of the k-tour cover problem when the depots are unspecified (Unrooted), there is a single depot or there are multiple depots on the network, with respect to two different objective functions, i.e. (i) minimizing the sum of the cost of all the tours, and (ii) minimizing the maximum cost incurred by each vehicle; α_{TSP} is the best approximation ratio for TSP.

k-Tour cover	Unrooted	Single depot r	Multiple depots R=$\{r_i\}_{i=1}^{k}$
Min sum	2	$3/2$ [19]	$3/2$ [43] for k const
Min max	6 [31]	$\alpha_{TSP} + 1 - 1/k$ [18]	$8 + \epsilon$ [16]; $7 + \epsilon$ (Theorem 5)

shown in Lemma 4. This, gives a k-tree cover, which can be turned into a tour cover by duplicating and shortcutting edges.

Figure 2.1: Algorithm for the rooted k-Tour cover problem

Input: $G = (V, E), c : E \to \mathbb{Q}^+, R = \{r_1, \ldots, r_k\}, B \in \mathbb{Q}^+$
Output: A set of tours $\tau_1, \ldots, \tau_k$ of cost at most $7B$ each, rooted at $r_1, \ldots, r_k$ respectively, if $B \geq$ OPT.

1. Find the forest F by discarding edges of length greater than $\frac{B}{2}$ (let $E_{\leq \frac{B}{2}}$ be the set of remaining edges e with $c(e) \leq B/2$). Apply an MST algorithm, *e.g.* Kruskal on each connected component of $G_{\leq B/2} = (V \setminus R, E_{\leq \frac{B}{2}})$.
2. For each connected component T of F, split T into trees that have size in the interval $[\frac{3}{2}B, 3 \cdot B)$ and potentially one left-over tree of size in $(0, \frac{3}{2}B)$
3. Consider the bipartite graph $G' = (\mathcal{T}, R, E')$, where $\mathcal{T}$ is the set of trees split from T in the previous step, and the set of edges are
$E' = \{(T, r) | T \in \mathcal{T}, r \in R, \exists v \in T \text{ such that } c(r, v) \leq \frac{B}{2}\}$.
4. Find a maximum matching M in G'. For each root $r \in R$, let $e_M(r)$ be the edge matched to r. Consider the tree resulting from the union of this edge and the tree $T_M(r)$ that the root has been matched to. Form a tour rooted at r by doubling and shortcutting the edges of $T_M(r) \cup e_M(r)$.

As mentioned above, we assume that F is connected and is a tree. First, we prove an upper bound on the cost of this tree.

Lemma 2 $c(F) \leq (3k/2 - 1)B$

Proof: Consider an optimal solution, where every vertex $v \in V$ is covered by a tour. The sum of the cost of all the tours is at most $k \cdot B$. Now, shrink the components induced by these optimal tours down to their respective roots and consider the MST on the roots in the shrunk graph. As per our assumption, the graph is connected and has no edge of cost greater than $B/2$. So the cost of the MST is at most $(k - 1)B/2$. Expanding the tours back, together with this MST gives a connected subgraph of cost $(3k/2 - 1)B$. Hence, the same upper bound applies to the minimum spanning tree. ■

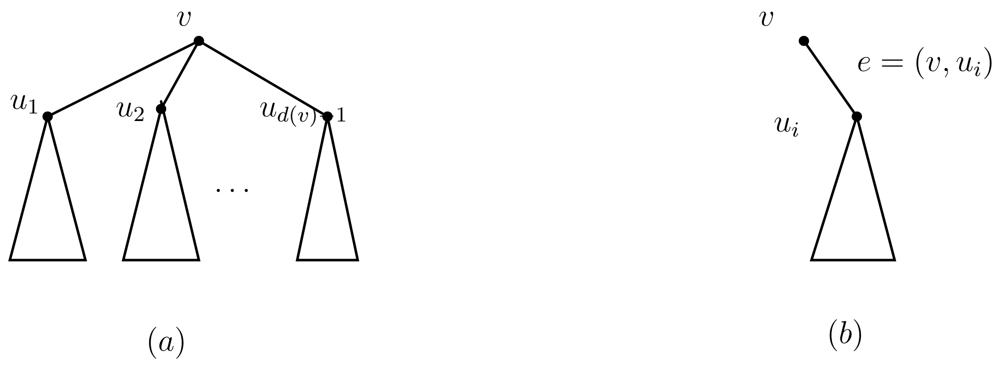
The following lemma has been proved in [16]. We reproduce the proof in here for the sake of completeness.

Lemma 3 *[16] For any tree T with a cost function c on the edges and some λ, where $c(e) \leq \lambda$ for every $e \in E$, T can be split away to edge-disjoint trees of size in $[\lambda, 2\lambda)$ and at most one left-over tree of size in $(0, \lambda)$.*

Proof: For every vertex v, let T_v be the subtree rooted at v. If $e = (u, v)$ is an edge in the tree and u is a parent of v, let T_e denote the tree rooted at u with the edge e and the subtree T_v. Looking at each vertex v, we consider all the subtrees rooted at v where the degree of v is 1. If v

is connected to its children by means of the edges $e_1 = (v, u_1), \ldots, e_{d(v)-1} = (v, u_{d(v)-1})$, such degree-1 subtrees are $T_{e_1}, \ldots, T_{e_{d(v)-1}}$ (see Figure 2.2). Now, consider the deepest vertex v, where $c(T_v) > 2\lambda$, but for all its degree-1 subtrees $c(T_{e_i}) < 2\lambda$. We can take away each degree-1 subtree that has cost in $[\lambda, 2\lambda)$. All the remaining degree-1 subtrees will have cost in $(0, \lambda)$. Start packing these subtrees together one at a time in some arbitrary order until the cost of the tree is greater than λ for the first time. Take this new subtree T' as a new one and set it aside; $c(T') < 2\lambda$. Now iterate on finding the new vertex v. In the last iteration, there may be only one tree left of cost less than λ.

Figure 2.2: (a) T_v, the subtree rooted at v, and all its children, $u_1, \ldots, u_{d(v)-1}$. (b) T_e, a degree-1 subtree of v that is formed by its edge to its child u_i, and T_{u_i}.



By adding such subtrees one at a time we can form a tree T of the desired size, add it to $\mathcal{T}$, split it away from F and iterate. ■

Now apply the above lemma to F with $\lambda = 3B/2$. Furthermore note that these trees are edge disjoint and their costs can add up to at most $c(F)$, thus there will be at most $k-1$ trees of cost in $[3B/2, 3B)$ and one left-over tree of cost in $(0, 3/2B)$.

Lemma 4 *G' has a a perfect matching.*

Proof: G' is a bipartite graph with bipartition $(\mathcal{T}, R)$ of the trees that we split away in the algorithm and the roots available. Our goal is to show that every tree that we split away can be matched to a different root close to it. Note that the edges are $E(G') = \{(T, r) | T \in \mathcal{T}, r \in R, \exists v \in T \text{ such that } c(r, v) \leq \frac{B}{2}\}$. In order to prove this, we appeal to Hall's theorem which gives a sufficient and necessary condition for the existence of a perfect matching. In this case, Hall's theorem assures the existence of a matching of size $|\mathcal{T}|$, when the following condition holds. For every $\mathcal{T}' \subset \mathcal{T}$, it has at least as many neighbors in G', i.e. $|N_{G'}(\mathcal{T}')| \geq |\mathcal{T}'|$.

Now, consider a subset $\mathcal{T}' \subset \mathcal{T}$, which consists of $t = |\mathcal{T}'|$ trees. For the sake of convenience, name the trees as $\mathcal{T}' = \{T_1, \ldots, T_t\}$. We can characterize the set of its neighbors as $R' = N_{G'}(\mathcal{T}') = \{r \in R | c(r, v) \leq B/2, \text{ such that } v \in T_i, \text{ for some } i \leq t\}$.

Note that no root in $R \setminus R'$ can cover any vertex in $V(\mathcal{T})$ in the optimal solution, since any such root is too far from all the vertices. So, there are $|R'|$ tours $C_1, \ldots, C_{|R'|}$ rooted at R' that cover

$\cup_i V(T_i)$ in the optimal solution. We stress the fact that we don't know what these tours are, and use them to prove existence.

The Hall's condition that we are left to prove is that $|R'| \geq t$. Assume for the sake of contradiction that $|R'| < t$. Suppose we shrink the tours $C_1, \ldots, C_{|R'|}$ in the original graph G. Let E' be the set of the edges of F that lie in the shrunk graph $G/\{V(C_1), \ldots, V(C_{|R'|})\}$. In other words, E' is the set of edges that run between the different components spanned by the tours C_i.

Now, consider the tree F, we remove all the edges in the trees $\{T_i\}_{i=1}^{t}$ and add all the edges in $\bigcup_{i=1}^{|R'|} E(C_i) \cup E'$ to make a graph F'. We show that: (a) F' has cost strictly less than F, and (b) F' maintains all the connectivities provided by F. The points (a) and (b) contradict with the fact that F is an MST.

Proof of (a): Consider all the edges removed, $E_T = \bigcup_{i=1}^{t} E(T_i)$. We know that all the trees in $\mathcal{T}$, except potentially one have cost in $[3/2B, 3B)$, so $c(E_T) \geq \frac{3}{2}(t-1)B$. On the other hand, consider all the edges added to the graph, $E_C = \bigcup_{i=1}^{|R'|} E(C_i) \cup E'$. Each cycle C_i has a cost of at most OPT and the cost of E' is dominated by a minimum spanning forest in the shrunk graph $G/\{V(C_1), \ldots, V(C_{|R'|})\}$ when the edges are of cost at most $B/2$. Thus, $c(E') \leq (|R'|-1)\frac{B}{2}$. So, $c(E_C) \leq |R'| \cdot \mathrm{OPT} + (|\mathrm{R}'|-1)\frac{\mathrm{B}}{2} \leq \frac{3}{2}|\mathrm{R}'| \cdot \mathrm{B} - \frac{\mathrm{B}}{2}$. Together with the assumption that $|R'| < t$, this gives $c(E_C) \leq \frac{3}{2}(t-1)B - \frac{B}{2}$. This shows that the edges removed have greater cost than the edges added $c(E_T) > c(E_C)$.

Proof of (b): Now, we claim that if we replace $E(F)$ with $(E(F) \backslash (\bigcup_{i=1}^{t} E(T_i))) \cup (\bigcup_{i=1}^{|R'|-1} E(C_i) \cup E')$, the connectivity of vertices remains unchanged. The only vertices that can lose their connectivity are the ones in a tree that has its edges removed. Let $s, t \in T$ be two such vertices in some $T \in \mathcal{T}'$. Let s and t be connected in T by a path $s = v_0, v_1, \ldots, v_r = t$. For each $0 \leq i < r$, v_i and v_{i+1} are covered by two tours C_1^i and C_2^i. If the two tours happen to be the same, $E(C_1^i)$ maintains the connectivity of v_i and v_{i+1}. Otherwise, $(v_i, v_{i+1}) \in E(T)$ will be an inter-cycle edge in T. So, it will be kept in E'. Thus, the connectivity will be preserved.

(a) and (b) together with F being the minimum spanning tree, results in a contradiction, hence we must have $|R'| \geq t$. This, in turn will prove the existence of a matching in G' that covers all trees $\mathcal{T}'$ by Hall's theorem as stated above. ■

Theorem 5 *Algorithm of Figure 2.1 is a $(7+\epsilon)$-approximation for the rooted k-tour cover problem.*

Proof: The set of trees split away from F in Lemma 3 covers all the vertices of G and the perfect matching M ensured by Lemma 4 gives a way to map the trees $\mathcal{T}$ to the roots R. Each tree formed by connecting a root $r \in R$ to a corresponding tree costs at most $\mathrm{OPT}/2 + 3\mathrm{B} \leq 3.5\mathrm{B}$. Each tree can be turned into a tour by duplicating the edges and shortcutting over the vertices of each graph results in a tour of cost $7B \leq (7+\epsilon) \cdot \mathrm{OPT}$. ■

As shown above, this improves upon the factor $(8+\epsilon)$-approximation of [16], namely to $(7+\epsilon)$. Note that if we can partition the vertices and assign each partition to the respective root in R as in the optimal solution, the problem breaks down to solving k independent instances of the traveling salesman problem and the $\alpha_{TSP} = 3/2$ approximation of TSP will follow. In our approach above, we manage to partially disentangle the k instances of TSP by removing edges of cost $B/2$, where no partition might be broken up. This suggests that further disentangling the instances might lead into better approximations.

2.4 Unrooted k-tour cover algorithm

In this section, we give an algorithm for the unrooted k-tour cover problem. The algorithm is depicted in Figure 2.3. Similar to the algorithm for the rooted version, algorithm 2.3 guess an upper bound B on the optimum such that $\mathrm{OPT} \leq \mathrm{B} \leq (1+\epsilon)\cdot\mathrm{OPT}$. This guess is carried out by means of a binary search over the range of possible values. Algorithm 2.3 takes such a value $B \geq \mathrm{OPT}$ as an argument, and produces a solution with a guaranteed bound with respect to B.

The algorithm works as follows. Edges with a cost of more than $B/3$ are discarded. This might result in more than one connected component. Note that a tour in an optimal solution can only have at most two edges discarded, and thus will either be entirely in one connected component or two. The algorithm then tries to restore some of the lost connectivity by adding edges. Consider the *small* components in $G_{\leq B/3}$ (defined in step 2 in Algorithm 2.3). We only restore the connectivity for such small components. The algorithm guesses the values d and f by iterating over all the possible values. f is the number of small components that have a broken part of a tour from an optimal solution and the other part being part of a large component. And d is the number of small components that cannot be matched to any other component. This can be due to the fact that the small component does not include a broken part of a tour in an optimal solution. To represent all the possible connections to the small components, the graph G' is built in Step 4 of the algorithm. $V(G') = X \cup Y \cup Z$, where X corresponds to all the small components, Y is a set of dummy nodes that will define which components won't be matched, and Z is the set of small components that get matched to a large component. This correspondence of Y and Z to the small components and the possible matchings of different small components together is found by a matching M in G' in Step 5 of the algorithm. The connectivities between the components in G' are restored according to the matching M in Step 6. Then in each component we split away trees from the minimum spanning tree by the procedure of lemma 3 with $\lambda = 4B/3$. The resulting tree cover is then duplicated and shortcut to form a tour cover as explained in Algorithm 2.3.

Lemma 6 *The cost of each tour τ output by Algorithm 2.3 is at most* $16B/3$.

Proof: We first bound the cost of a tree $T \in \mathcal{T}$: (i)If $T \in \mathcal{T}_1$, then T is the MST on the union of two small trees, which can be connected by an edge of cost at most $B/2$. As the cost of each small tree is at most B. There is a graph spanning vertices of T of cost at most $5B/2$. (ii) If $T \in \mathcal{T}_2$, then T is a small tree and has cost $c(T) < B$. (iii) If $T \in \mathcal{T}_3$, then T is output by the tree splitting

Figure 2.3: Algorithm for the unrooted k-tour cover problem

Input: $G = (V, E), c : E \to \mathbb{Q}^+, k \in \mathbb{N}, B \in \mathbb{Q}^+$
Output: A set of tours $\tau_1, \ldots, \tau_k$, covering V, where the cost of each tour is at most $16B/3$, if $B \geq \text{OPT}$.

1. Remove edges of cost $c_e > B/3$. Let $G_{\leq B/3}$ be the resulting graph, and $C_1, \ldots, C_s$ be its connected components.
2. Compute the MST in each connected component of $G_{\leq B/3}$, and partition them into two groups $\mathcal{S}$ and $\mathcal{L}$, called *small* and *large* respectively as follows.
   - $\mathcal{S} = \{C_i | MST(C_i) \leq B\}$.
   - $\mathcal{L} = \{C_i | MST(C_i) > B\}$.
3. Guess the values d, f by iterating over all the possible values in $\{0, \ldots, |\mathcal{S}|\}$, running the algorithm with each such guessed pair and returning the best output.
4. Build the graph $G' = (V', E')$ as follows.
   - $V' = X \cup Y \cup Z$, where
     - $X = \{x_1, \ldots, x_{|\mathcal{S}|}\}$, where each x_i corresponds to the i-th small component in $\mathcal{S}$.
     - $Y = \{y_1, \ldots, y_d\}$ are dummy vertices.
     - $Z = \{z_1, \ldots, z_f\}$.
   - $E' = E_1 \cup E_2 \cup E_3$, where
     - $E_1 = \{\{x_i, x_j\} | x_i, x_j \in X, c(C_i, C_j) < B/2\}$, and set cost $c(e) = 0$, for all $e \in E_1$.
     - $E_2 = \{\{x_i, y_j\} | x_i \in X, y_j \in Y\}$, and set cost $c(e) = 0$, for all $e \in E_2$.
     - $E_3 = \{\{x_i, z_j\} | x_i \in X, z_j \in Z, c(\{x_i, z_j\}) = c_l(C_i) < B\}$, where the costs on the edges are defined by $c(\{x_i, z_j\}) = c_l(C_i) = MST(C_i) + c(C_i, \mathcal{L})$. And $c(C_i, \mathcal{L}) = \min_{\substack{u \in C_i, \\ v \in C, C \in \mathcal{L}}} c(u, v)$ is the minimum cost of an edge between the component C_i and large components.
5. Find a minimum cost matching M in $G' = (V', E')$. Let $M = M_1 \cup M_2 \cup M_3$, where $M_i = M \cap E_i$, for $i = 1, 2, 3$.
6. Let $\mathcal{T} = \mathcal{T}_1 \cup \mathcal{T}_2 \cup \mathcal{T}_3$, where
   - $\mathcal{T}_1 = \{MST(C_i \cup C_j) | \{x_i, x_j\} \in M_1\}$.
   - $\mathcal{T}_2 = \{MST(C_i) | \{x_i, y_j\} \in M_2, \text{ for some } j\}$.
   - For each $\{x_i, z\} \in M_3$, merge the small component C_i with the large component C_j that is closest to it. Let $\mathcal{L}'$ be the new large components after merging. For each $\mathcal{T}_3'$ be the union of the MSTs of components in $\mathcal{L}'$. Apply the tree splitting Lemma 3 to each tree in $\mathcal{T}_3'$ with $\lambda = 4B/3$, with the modification that the second last tree and the left-over tree are taken as one tree. Let $\mathcal{T}_3$ be the set of resulting trees.
7. Output the tours resulted by duplicating and shortcutting the edges in each of the trees in $\mathcal{T}$.

Lemma 3 which produces trees of cost at most $2\lambda = 8B/3$. Each tour τ is produced by duplicating and shortcutting a tree $T \in \mathcal{T}$. Thus, $c(\tau) \leq 2c(T) \leq 16B/3$. ■

Lemma 7 *The number of tours output by algorithm 2.3 is at most k.*

Proof: For the proof of this theorem, we need to look at the structure of the optimum solution. Let $T_{\mathrm{OPT}} = \{\tau_1, \ldots, \tau_k\}$ be the k tours of an optimal solution. Each such tour τ_i has cost at most B. Thus, τ_i either falls in one connected component in $G_{\leq B/3}$, or in at most two different connected components, as it can have at most two edges greater than $B/3$. In the latter case, τ_i has been broken by removal of edges of cost greater than $B/3$. The optimum tours of T_{OPT} in $G_{\leq B/3}$ fall in one of the following categories:

1. *Small tours.* The whole tour or its two parts fall in $\mathcal{S}$, i.e. small components of $G_{\leq B/3}$. Let k_s be the number of such tours in T_{OPT}.

2. *Mixed tours.* The tour has been broken in $G_{\leq B/3}$, and one part is in $\mathcal{S}$ and the other part is in $\mathcal{L}$. Let k_m be the number of such tours in T_{OPT}.

3. *Large Tours.* The whole tour or its two parts fall in $\mathcal{L}$, i.e. large components of $G_{\leq B/3}$. Let k_l be the number of such tours in T_{OPT}. These tours consist of the two following groups:

   - Tours that are wholly contained in one large component in $\mathcal{L}$. Let $k_l^{(h)}$ be the number of such tours in T_{OPT}.
   - Tours that are broken, but both parts are in large components in $\mathcal{L}$. Let $k_l^{(b)}$ be the number of such tours in T_{OPT}.

By definition, we have $k_l = k_l^{(h)} + k_l^{(b)}$. Now, in order to consider the structure of T_{OPT} over the components corresponding to V', we define the following graph. Let $G'' = (X, E'')$, where there is an edge $E'' = \{(x_i, x_j) | \exists \tau \in T_{\mathrm{OPT}} \text{ s.t. } V(\tau) \cap C_i \neq \emptyset, V(\tau) \cap C_j \neq \emptyset\}$. Note that E'' also includes loops, as i and j are not required to be distinct. Let $I \subset X$ be the set of isolated nodes in G''. This means that the corresponding components to I are contain incident to mixed tours. Let $M_{G''}$ be a maximum matching in G'', and $U_{G''} = X \setminus (I \cup V(M_{G''}))$, the isolated nodes of G'' that are not matched. Similarly, let $U = X \setminus (I \cup V(M))$.

Consider an iteration of algorithm 2.3, where $f = |I|$, and $d = |U_{G''}|$. We have $|M| + |U| \leq |M_{G''}| + |U_{G''}|$. Furthermore, $|M_{G''}| + |U_{G''}| \leq k_s$.

In Algorithm 2.3, for the number of trees we have $|\mathcal{T}_1| = |M|$ and $|\mathcal{T}_2| = |U|$, thus $|\mathcal{T}_1| + |\mathcal{T}_2| \leq k_s$. It remains to bound $|\mathcal{T}_3|$. In order to bound this value, we consider the total cost of $\mathcal{T}_3$. Note that $\sum_{T \in \mathcal{T}_3} c(T) \leq \sum_{T \in \mathcal{T}'_3} c(T)$ and that $\mathcal{T}'_3$ is a forest with $|\mathcal{L}|$ connected components. We bound the total cost $\sum_{T \in \mathcal{T}'_3}$ by considering the large and mixed tours of T_{OPT} and the edges connecting the mentioned tours. The total cost of large and mixed tours is at most $(k_l + k_m) \cdot B$. Any two such tours can be connected by an edge of cost at most $B/3$ in $G_{B/3}$. Thus, a spanning forest with $|\mathcal{L}|$ connected components can be made with an additional cost of $(k_l + k_m - |\mathcal{L}|) \cdot B/3$. So, $\sum_{T \in \mathcal{T}'_3} c(T) \leq 4B/3 \cdot (k_l + k_m) - B/3 \cdot |\mathcal{L}|$. Note that $\mathcal{T}_3$ is made from $\mathcal{T}_3$ by the application of Lemma 3 with $\lambda = 4B/3$. Also, note that Lemma 3 has been applied with the modification that the second last tree is kept together with the last, i.e. left-over tree. Thus the only trees that have cost less than $4B/3$, is a tree that is a single component in $\mathcal{T}'_3$. As each component in $\mathcal{T}_3$ is a large component, the cost of the tree is at least B, and there are at most $|\mathcal{L}|$ such trees. Thus, $|\mathcal{T}_3| \leq k_l + k_m$.

■

The above two lemmas imply the main theorem of this section

Theorem 8 *Algorithm 2.3 is a $(16/3 + \epsilon)$-approximation algorithm for the unrooted k-tour cover problem.*

Chapter 3

Min-max k-Star Cover

In this chapter, we consider the star cover problems, that is the problem of covering the vertices of a graph using a number of stars. Star cover problems are at the intersection of vehicle routing problems and clustering problems. In VRP, star cover problems model cases when each vehicle is stationed at a depot and it needs to go back to its designated depot after serving each customer before it can go and serve another customer. On the other hand, star cover problems can also be viewed as a class of clustering problems. Each star denotes one cluster with the root being the cluster center and leaves being cluster members. Clustering problems are of great practical and theoretical importance in computer science and thus, they have been studied through different lenses and by separate communities, such as machine learning and algorithms. The area of clustering data is rich with an extensive literature. We will only review the most relevant literature in section 3.2.

3.1 Problem Definitions

Before we turn to the rigorous study of star cover problems, we need to define them formally. Consider Example 2 in Chapter 1. We can model the customer locations and potential warehouse location as the underlying network by a graph G and the distances in between locations by a metric cost function c. Similar to the tour cover problem discussed in Chapter 2, this gives rise to two variants of the problem, depending on whether or not the warehouse locations have already been established. We refer to the variant in which warehouse locations have been fixed as the *rooted* star cover problem and define it formally as follows.

Problem 3 (Rooted k-Star Cover) *Given a graph (V, E) with a metric cost function on the edges $c : E \to \mathbb{Q}^+$, and a set of roots $R \subseteq V$, find a partitioning of $V \setminus R$ into k sets $V_1, \ldots, V_k$, where $R = \{r_1, \ldots, r_k\}$, such that the cost of the heaviest star $S_i = (r_i, V_i)$ is minimized. Thus, the objective function is to minimize $\max_i \sum_{v \in V_i} c(r_i, v)$.*

Similarly, we can define the variant in which the locations of the roots of the stars is part of the decision problem as the *unrooted* the star cover problem as follows.

Problem 4 **(Unrooted k-Star Cover)** *Given a graph (V, E) with a metric cost function on the edges $c : E \to \mathbb{Q}^+$, find a partitioning of V into k sets $V_1, \ldots, V_k$, together with a root in each partition $r_i \in V_i$, such that the cost of the heaviest star $S_i = (r_i, V_i)$ is minimized. Thus, the objective function is to minimize* $\max_i \sum_{v \in V_i} c(r_i, v)$.

3.2 Background

Even et al. [15] considered both rooted and unrooted variants of the min-max k-star cover problem. They notice that the rooted version is a special case of the problem of Minimum makespan scheduling on unrelated parallel machines, for which a 2-approximation exists due to Lenstra et al. [33]. Thus, the 2-approximation carries over to the rooted k-star cover problem as well. Lenstra et al. [33] also give a $3/2$ hardness of approximation for the scheduling problem. This result does not automatically result in a corresponding lower bound on the approximability of rooted k-star cover problem. But looking more closely at their reduction, it can be seen that the instance they construct has a metric cost function on the edges of the graph. Thus, the $3/2$ lower bound also applies to rooted k-star cover problem. It remains to close this gap and establish the approximability threshold of rooted k-star cover problem, which is in the interval $(3/2, 2]$. But, as this is almost resolved, we turn our attention to the unrooted variant in this thesis.

For the unrooted k-star cover problem, Even et al. [15] give a $(4+\epsilon, 4)$ bi-criteria approximation. Their algorithm produces $4k$ stars covering all the vertices, such that the cost of each star is no more than 4 times the cost of optimally covering the vertices by k stars. Formally, an (α, β) bi-criteria approximation algorithm for the star cover problem is a set of $\beta \cdot k$ stars that cover all the vertices, such that the cost of each star is at most $\alpha \cdot \text{OPT}$, where OPT is the optimal cost of covering all the vertices by k stars. Note that the solution provided by a bi-criteria approximation is not feasible as it uses more than k stars. However, the cost of the objective function is compared to the cost of the optimum solution *without* violating the restriction of using only k stars.

Subsequently, Arkin et al, [3] improve their result and give a $(3+\epsilon, 3+\epsilon)$ bi-criteria approximation. They also consider another variant of the star cover problem, in which the number of stars is to be minimized, subject to a hard budget constraint on the cost of each star. For this version, they give a $(2\alpha_{k-med} + 1)$-approximation, where α_{k-med} is the approximation ratio of the related k-median problem. We will review this problem in the following.

Clustering. The star cover problem can also be viewed as a k-clustering problem. In a k-clustering problem, the goal is to partition a given metric space into k partitions, in such a way that a certain objective function over the metric is minimized. Different objective functions have been considered arising from various applications. Clustering problems are a mainstay of computer science. The problem has been considered in various forms in different fields. In the following, we review how the problem has been dealt with from the viewpoint of approximation algorithms. A few of the objective functions that have received much attention in approximation algorithms are as follows.

- k**-median**. In the k-median problem, the goal is to produce a k-clustering of the vertices and designate a vertex as the center in each cluster, to minimize the total sum of distances of a vertex to its center. Note that this is very similar to the k-star cover problem considered in this chapter, with the exception that the objective function is a sum over all the clusters, as opposed to the maximum of each cluster. Various approximation algorithms for k-median have been developed, exhibiting a wide range of techniques in the field from rounding [9] to primal-dual technique [29] and local search [5, 23], just to name the hallmarks. These efforts culminated in a recent $(1 + \sqrt{3} + \epsilon)$-approximation due to [34]. Regarding the importance of the k-median problem, it has also been studied in more restricted metrics. In the fixed-dimensional Euclidean metric, the problem remains NP-hard, and a PTAS is known due to [4]. For doubling metrics, a QPTAS has been given by [38]. Interestingly, the problem is solvable in polynomial time on tree metrics as shown by a dynamic programming given by [39].

- k**-means**. In the k-means problem, the goal is to produce a k-clustering of the vertices and designate a vertex as the center in each cluster, in order to minimize the total sum of squares of distances of each vertex to its center. Note that this is very similar to the k-median problem discussed above, with the exception that the squares of distances are considered. This minor difference makes some of the techniques developed for the k-median problem not applicable. k-means has been heavily used in practice for clustering data and various heuristics have been developed for it. From a theoretical perspective, there is an $O(1)$-approximation by local search due to [30].

- k**-center**. In the k-center problem, the goal is to produce a k-clustering of the vertices and designate a vertex as a center to each cluster, so as to minimize the maximum distance of each vertex to its center. Approximation algorithm with an tight approximation ratio of 2 is known [26]. k-center has also been considered in restricted metrics. In the two-dimensional Euclidean metric, it is hard to approximate to a factor smaller than $\sqrt{3}$ [17]. It remains an open problem to give an approximation algorithm with a ratio better than 2 for the two-dimensional Euclidean metric.

- **Min sum radii**. In this problem, the goal is to produce a k-clustering of the input vertices , so as to minimize the sum of the radii of the clusters. A 3.504-approximation was given by [8] for general metrics.

- **Min Sum** k**-Clustering**. In this problem, the goal is to produce a k-clustering of the input vertices, in order to minimize the total sum of all the pairwise distances that fall in the same cluster. There is a PTAS for the case when the number of clusters k is a constant, due to [14], and a $O(\frac{1}{\epsilon}\log^{1+\epsilon} n)$-approximation algorithm for general metrics. This algorithm runs in time $n^{O(\frac{1}{\epsilon})}$ and is due to [6].

3.3 Bi-criteria Approximations

In this section, we look at the bi-criteria approximation for the unrooted k-star cover problem. We revisit the algorithm of Arkin et al. [3], generalize it, improve it and analyze it for general parameters, and also consider its implications for some restricted metrics.

Theorem 9 *[3] There is a $(3+\epsilon, 3+\epsilon)$ bi-criteria approximation algorithm for the Min-max k-Star cover problem.*

We first review their algorithm, as it is the basis for the generalization we present later. The algorithm works as follows. First, they guess an upper bound B on the optimum value of the star cover problem. The approximation will follow by doing a binary search over the range of possible values for the optimum. In the description of the algorithm below we assume that B is such an upper bound in the interval $[\mathrm{OPT}_{\mathrm{k-SC}}, (1+\epsilon)\mathrm{OPT}_{\mathrm{k-SC}}]$, where $\mathrm{OPT}_{\mathrm{k-SC}}$ is the optimal value for the k-star cover problem. The algorithm takes the input $I = (G, c : E \to \mathbb{Q}^+, k)$ to the k-star cover problem, together with the guessed valued B. Then, it runs the $(3+\epsilon)$-approximation of [5] for the k-median problem given input I. This results in a k-star cover with the total cost of at most $(3+\epsilon)\cdot k \cdot B$. For each star in the this cover, the algorithm starts from the arm of the star with the largest cost, going in non-increasing order, up to the first leaf that would accumulate a cost $\lambda = 3B/2$ in the arms of the star. This substar is then plucked away and a new star is formed, where the leaf that had the shortest arm to the root, is the new root, covering the other leaves. The cost of each star formed this way will be at most $2\lambda \le 3B \le (3+\epsilon)\mathrm{OPT}_{\mathrm{k-SC}}$. Also, the total number of stars will be at most $(3+\epsilon)kB/\lambda + k \le (3+\epsilon)k$, where the second term comes from the left-over star at the end of each star given by the k-median algorithm. Hence, the $(3+\epsilon, 3+\epsilon)$ bi-criteria approximation. We parameterize the value of λ in the above algorithm and generalize their result as follows.

Theorem 10 *Let α be the approximation ratio of the k-median problem, and $\beta > 0$ be a parameter. There is a $(2\beta, 1+\alpha/\beta)$ bi-criteria approximation algorithm for the Min-max k-star cover problem.*

Proof: The proof is very similar to that given by [3] that was presented above. By means of a binary search, we guess an upper bound B of the k-star cover problem, where $B \in [\mathrm{OPT}_{\mathrm{k-SC}}, (1+\epsilon)\mathrm{OPT}_{\mathrm{k-SC}}]$, for a given $\epsilon > 0$. We know that $\mathrm{OPT}_{\mathrm{k-med}} \le \mathrm{k}\cdot\mathrm{OPT}_{\mathrm{k-SC}} \le \mathrm{k}\cdot\mathrm{B}$. Running the α-approximation algorithm of k-median, we get k stars $S_1, \ldots, S_k$, such that $\sum_i c(S_i) \le \alpha \cdot k \cdot B$. For each star S_i do the following: Let r_i be its root and $v_1^i, \ldots, v_{l_i}^i$ be its leaves in order of non-increasing distance from the root r_i. Find the smallest t, such that $\sum_{j=1}^{t} c(r_i, v_j^i) > \beta \cdot B$. Separate the leaves $v_1^i, \ldots, v_t^i$ from S_i and make a new star S_1^i that has v_t^i as its root and $v_1^i, \ldots, v_{t-1}^i$ as its leaves. Repeat the above procedure on the remaining star. S_1^i will have cost at most $2\beta \cdot B$. The number of the stars produced this way is at most

$$\begin{aligned}\sum_{i=1}^{k}(\lfloor c(S_i)/\beta \cdot B\rfloor + 1) &\leq k + \sum_{i=1}^{k} c(S_i)/\beta \cdot B && (3.1)\\ &\leq k + \alpha \cdot k \cdot B/\beta \cdot B && (3.2)\\ &\leq k(1+\alpha/\beta) && (3.3)\end{aligned}$$

■

By taking $\beta = \alpha/\epsilon$, we get the following corollary.

Corollary 1 *There is a $(O(\frac{1}{\epsilon}), 1+\epsilon)$ bi-criteria approximation for the min-max k-star cover problem:*

Note that the $(3+\epsilon, 3+\epsilon)$ bi-criteria approximation of [3] follows by an application of the Theorem 10 with $\beta = \alpha/2$ and $\alpha = 3+\epsilon$ [5]. One immediate improvement is that the approximation ratio of k-median has since been improved to $\alpha = 1 + \sqrt{3} + \epsilon$[34]. Applying this, we get the following corollary.

Corollary 2 *We have the following bi-criteria approximations for the min-max k-star cover problem:*

1. $(\alpha' + \epsilon, \alpha' + \epsilon)$, *where* $\alpha' = \frac{1+\sqrt{1+8\alpha}}{2} = \frac{1+\sqrt{1+8(1+\sqrt{3})}}{2} < 2.8905$.
2. $(\alpha'' + \epsilon, 2)$, *where* $\alpha'' = 2\alpha = 2 + 2\sqrt{3} < 5.4642$.

Considering the approximation ratios of k-median on the line and the Euclidean plane, which are 1 and $1+\epsilon$, respectively, we get the following corollary.

Corollary 3 *The min-max k-star cover problem on the*

1. *line metric has a $(1+\epsilon, \frac{2}{\epsilon})$ bi-criteria approximation.*
2. *2-dimensional Euclidean metric has a $(1+\epsilon, \frac{2}{\sqrt{\epsilon}})$ bi-criteria approximation.*

3.4 Line Metric

In this section, we study the Min-max k-star cover, when the cost function is a line metric. In contrast to other clustering problems, such as k-center and k-median that are polynomial time solvable on the line metric, the k-star cover problem remains NP-hard on the line metric.

In Section 3.4.1, we present a QPTAS for the k-star cover problem on the line metric, and in Section 3.4.2 we give a PTAS for the special case when the stars are non-crossing. As the metric is a line metric, we denote the metric by l rather than c in this section.

3.4.1 QPTAS

In this section, we give a quasi-polynomial time approximation scheme for the min-max k-star cover problem, when the input metric is a line metric. In addition to the metric (V, l) and k, the algorithm is also given a precision parameter ϵ as part of the input, and it produces k clusters with respective centers in them. The cost of the star in each cluster rooted at its center is guaranteed to be at most $(1+\epsilon)$ times the maximum cost of a star in the optimal clustering. The running of the algorithm will be quasi-polynomial in n, i.e. $O(n^{\text{Polylog}(n)})$ for any fixed ϵ.

Preprocessing

In the following, we assume that we have guessed the value of the optimal solution up to a factor of $1+\epsilon'$, for a fixed ϵ' depending on ϵ. The algorithm will work with such a guessed value B and produces a solution that is guaranteed to be at most $(1+\epsilon') \cdot B$ if $B \geq \text{OPT}$. So in order to obtain a good approximation ratio, we require that $B \leq (1+\epsilon') \cdot \text{OPT}$. Since the range of possible values of OPT is $[0, \sum_{i,j} l(i,j)]$, we can do a binary search on B in this range and run the algorithm each time, to find a value $B \in [\text{OPT}, (1+\epsilon') \cdot \text{B}]$. Such a binary search will take time $\log(\sum_{i,j} l(i,j))$, which is polynomial in the size of the input, for each fixed ϵ. The cost of the solution will then be at most $(1+\epsilon')^2 \cdot \text{OPT} \leq (1+\epsilon) \cdot \text{OPT}$, for $\epsilon' < \epsilon/3$. To simplify the notation from now on we use ϵ instead of ϵ' and assume $\text{OPT} \leq \text{B} \leq (1+\epsilon) \cdot \text{OPT}$.

Scaling. Given as input a sequence of points on the line $\{a_i\}_{i=1}^{n}$, we relocate the points as follows. Starting from a_1 and going from left to right, if the distance between two consecutive points $l(i, i+1)$ is less than $\frac{\epsilon \cdot B}{n^2}$, we relocate the point v_{i+1} from a_{i+1} to a_i, the location of its nearest left neighbor. Now, we argue that if we solve the star cover problem on this modified input and then revert the points back to their original locations, the cost will go up by only a factor of at most $1+\epsilon$. Consider one arm of a star in the solution to the modified version. The distance of such an arm will be a pairwise distance of points, $l(i,j) = |a_i - a_j|$. When we locate all the points back to their original locations, this distance can go up by $|j-i| \cdot \frac{\epsilon \cdot B}{n^2} \leq \frac{\epsilon \cdot B}{n}$. As each star has at most n arms, the blow-up cost to each star will be at most $\epsilon \cdot B$. Thus, if we find a solution of cost at most B on the modified instance, this would give us a solution of cost at most $(1+\epsilon) \cdot B$ on the original instance.

Hence, we can assume that a non-zero distance between any two consecutive points, $l(i, i+1) = |a_{i+1} - a_i|$ is at least $\frac{\epsilon B}{n^2}$.

Furthermore, we can assume, without loss of generality, that every consecutive distance $l(i, i+1)$ is at most B, as no arm of a star in the optimal solution can run across the corresponding points v_i and v_{i+1}. If $l(i, i+1) > B$, we can consider two independent instances of the problem with inputs $(\{a_1, \dots, a_i\}, k_1)$ and $(\{a_{i+1}, \dots, a_n\}, k_2)$ for all the $n-1$ possibilities where $k_1 \in \{1, \dots, n-1\}$ and $k_2 = n - k_1$.

So, the maximum pairwise distance $l(i,j)$ will be at most $n \cdot B$. Now, scaling the minimum consecutive distance $l(i, i+1)$ to 1, the maximum pairwise distance $l(i, j)$ will be at most n^3/ϵ, which is a polynomial in n and $1/\epsilon$. Note that as each star consists of at most n arms, each of length at most the maximum distance, B is at most n^4/ϵ. Thus, B is also polynomial in n and $1/\epsilon$.

Dynamic Programming

We present a Dynamic Programming (DP) that produces a $(1+\epsilon)$-approximate solution to an instance of the star cover satisfying the above assumptions. The subproblems for this DP are of the form $(V_{ij} = \{v_i, \dots, v_j\}, k')$, where the goal is to cover the points in the set V_{ij} of consecutive points by k' stars ($0 \leq k' \leq k$), whose centers have to be chosen from the same set V_{ij}, such that each star has cost at most B. Then, we will use dynamic programming to stitch together the solutions. We consider the sets V_{ij} given by the following binary dissection. The dissection will also give a tree structure on the sets that we will use in our dynamic programming.

Dissection. To solve an instance (V_{ij}, k') of the problem, we consider two subsets V_{im} and $V_{m+1,j}$ by breaking V_{ij} into two sets of consecutive points each, by breaking at a midpoint $m \in [i, j]$ such that $|V_{im}|$ and $|V_{m+1,j}|$ differ by at most 1. Note that such a midpoint m always exists. This process gives rise to a dissection tree of height $O(\log n)$ with the set $V = V_{1n}$ at the root, and n singleton intervals V_{ii} as leaves. The proposed DP will operate on this tree structure.

The difficulty of finding a partial solution for a set V_{ij} stems from the fact that in the optimal solution, we might have centers of clusters in V_{ij} covering points outside the set, and at the same time, have cluster centers not in V_{ij} covering points inside the set. Thus, we need to include this information on the interface of V_{ij} in the definition of subproblems. The interface of V_{ij} with both right and left sides of the rest of the line will keep track of all the edges crossing the sides of the interval $I_{ij} = [a_i, a_j]$ in terms of *surplus* and *deficiency* vectors as defined below.

- *Surpluses.* The lengths of the parts of broken arms past the point a_j (resp. a_i) outside the interval I_{ij} of stars originating from within I_{ij} and extending to the right (resp. left). Note that we have points co-located at the same location on the line. So the right and left surplus vectors $S^{(r)}$ and $S^{(l)}$ will be represented as vectors $(s_1, \dots, s_\sigma)$, where s_t will be the number of points located at a certain distance l_t from the right or respectively, left end of the interval I_{ij} that will be covered by stars originating from within I_{ij}

- *Deficiencies.* The lengths of the parts of broken arms entering I_{ij} through the point a_j (resp. a_i) of stars originating from outside I_{ij} and on its right (resp. left) side. Similar to the surplus vectors, the right and left deficiency vectors $D^{(r)}$ and $D^{(l)}$ will be represented as vectors $(d_1, \dots, d_\sigma)$, where d_t will be the number of points inside I_{ij} located at a certain distance l_t from one end of the interval I_{ij} that are to be covered by a center outside I_{ij}.

In the above definition of surplus and deficiency vectors, the numbers s_t and d_t of points at a certain length can be up to n, and the number of different lengths, σ, can be up to n^3/ϵ, leading to large number of possible interfaces. To cut down on the interface of an interval I_{ij} with the rest of the line, we round up the surplus and deficiency lengths on each of the right and left sides to the nearest power of $(1+\epsilon'/\log n)$, for some ϵ' depending on ϵ, at each level of dissection. Thus, we only keep track of lengths $l_t = (1+\epsilon'/\log n)^t$, $t \in \{1,\ldots,\sigma\}$. So there will be $\sigma = O(\log n \cdot \log B/\epsilon') = O(\log^2 n/\epsilon')$ different lengths and as a result at most $n^{O(\log^2 n/\epsilon')}$ different surplus and deficiency vectors. In this way, each arm of a star (in a star cover solution) will be scaled up by a factor of at most $(1+\epsilon'/\log n)$ at each level of DP computation (to account for the rounding), and since the depth of recursion (dissection) is $\lceil \log n \rceil$, this will result in an extra factor of $(1+\epsilon'/\log n)^{\lceil \log n \rceil} \leq (1+\epsilon)$ (for a suitable choice of ϵ') over the entire length of each arm. In other words, if a subproblem for an interval i,j and parameter k' is feasible (with each star costing at most B) without rounding the lengths of deficiency and surplus vectors then the subprbolem with rounded (up to nearest power of $(1+\epsilon'/\log n)$) lengths for deficiency and surplus vectors is feasible if each star is allowed to have cost at most $(1+\epsilon)\cdot B$.

The Dynamic Programming Table. Each entry of the table represents a subproblem $(i, j, k', D^{(r)}, D^{(l)}, S^{(r)}, S^{(l)})$, where:

1. (i, j) represents the set of points $V_{ij} = \{v_i, \ldots, v_j\}$.
2. k' is the number of centers to be opened from among the points in V_{ij}.
3. $D^{(r)} = (d_1^{(r)}, \ldots, d_\sigma^{(r)})$ and $D^{(l)} = (d_1^{(l)}, \ldots, d_\sigma^{(l)})$ are the deficiency vectors on the right and left sides of the interval $I_{ij} = [a_i, a_j]$, respectively.
4. $S^{(r)} = (s_1^{(r)}, \ldots, s_\sigma^{(r)})$ and $S^{(l)} = (s_1^{(l)}, \ldots, s_\sigma^{(l)})$ are the surplus vectors on the right and left sides of I_{ij}, respectively.

Each of $D^{(r)}, D^{(l)}, S^{(r)}$, and $S^{(l)}$ is a vector of size $\sigma = O(\log^2 n/\epsilon')$, where $d_t^{(p)}$ or $s_t^{(p)}$ (for $p \in \{l, r\}$) is the number of broken arm parts of length $(1+\epsilon'/\log n)^t$ (after rounding). Each entry of the table records in boolean values the feasibility of having k' stars rooted in the points in V_{ij}, such that each star has cost at most $(1+\epsilon)\cdot B$. Each of the k' stars would cover some points in V_{ij} and the points located at distances $S^{(r)}$ and $S^{(l)}$ from the endpoints a_i and a_j of the interval. The rest of the points have to be covered with the broken arms of $D^{(r)}$ and $D^{(l)}$, thus connected to the two sides a_i and a_j, respectively. The size of the DP table is $O(n^2 \cdot k \cdot n^{O(\log n \log B/\epsilon')}) = n^{O(\log^2 n/\epsilon')}$, which is quasi-polynomial in n.

Now, we show how to decide the feasibility of each subproblem in the table.

Base Case. The base case is when there is only one point in the interval, say V_{ii}. In this case, each configuration has the form $(i, i, k', D^{(r)}, D^{(l)}, S^{(r)}, S^{(l)})$ and $k' \in \{0, 1\}$. If $k' = 0$ then point i must be covered by centers outside this interval. In this case, the subproblem is feasible if the surplus vector is zero and the deficiency vector is all zero except for an entry of 1 in the corresponding to length zero. If $k' = 1$ then this point will be a center and can support surplus arms whose sum is up to $(1 + \epsilon'/\log n)B$. More specifically, the deficiency vectors should be all zero, and $\sum_{1\leq t\leq\sigma}(s_t^{(r)} + s_t^{(l)}) \cdot (1 + \epsilon'/\log n)^t \leq (1 + \epsilon'/\log n)B$. If these conditions are met then the entry of the table is set to "True".

Recursive Step. We decide the feasibility of a subprolem $P = (i, j, k', D^{(r)}, D^{(l)}, S^{(r)}, S^{(l)})$ by breaking at the midpoint $m \in V_{ij}$, considering the two subsets V_{im} and $V_{m+1,j}$ given by the dissection tree and enumerating over all the remaining values of subproblems P_1 and P_2 on the two respective subsets, and checking if at least one pair of subproblems P_1 and P_2 exist, that are feasible and consistent with P. The two subproblems will be of the form $P_1 = (i, m, k_1', D_1^{(r)}, D_1^{(l)}, S_1^{(r)}, S_1^{(l)})$ and $P_2 = (m + 1, j, k_2', D_2^{(r)}, D_2^{(l)}, S_2^{(r)}, S_2^{(l)})$.

Checking consistency entails checking that all the deficiency/surplus vectors from the subproblems P_1, P_2, and P are consistent at all the breaking points, i.e. i, m, and j. In order to do this, we need to decide how much of the surplus of either of P_1 or P_2 that crosses the midpoint m is to make up for deficiencies in the other subproblem, and how much would go all the way outside the interval I_{ij}. This can be easily done as follows by looking at the surplus lengths and grouping the lengths by comparing them to the length of the other interval. Hence, we can decompose $S_1^{(r)}$ and $S_2^{(l)}$ as follows. In the following, addition and subtraction of vectors are element-wise addition and subtraction, respectively.

- $S_2^{(l)}$ must be decomposed as $S_2^{(l)} = S_2^{'(l)} + S_2^{''(l)}$, where $S_2^{'(l)}, S_2^{''(l)} \in \mathbb{N}^\sigma$ are surplus vectors: $S_2^{''(l)}$ keeps track of the surplus lengths that land outside the interval I_{ij}, to the left of v_i, and $S_2^{'(l)}$ keeps track of all the remaining lengths, i.e. the ones that land and serve deficiencies in I_{im}. Formally, let $S_2^{(l)} = (s_1^{(l)}, \ldots, s_\sigma^{(l)})$, then $S_2^{'(l)} = (s_1^{\prime(l)}, \ldots, s_\sigma^{\prime(l)})$ and $S_2^{''(l)} = (s_1^{\prime\prime(l)}, \ldots, s_\sigma^{\prime\prime(l)})$ are constructed as follows. For all $t \in \{1, \ldots, \sigma\}$, we have $s_t^{\prime(l)} = s_t^{(l)}$, if $l_t \leq l(i, m + 1)$, and it's 0 otherwise. Correspondingly, we have $s_t^{\prime\prime(l)} = s_t^{(l)}$, if $l_t > l(i, m + 1)$, and it's 0 otherwise.

- $S_1^{(r)}$ must be decomposed as $S_1^{(r)} = S_1^{'(r)} + S_1^{''(r)}$, where $S_1^{'(r)}, S_1^{''(r)} \in \mathbb{N}^\sigma$ are surplus vectors. $S_2^{''(r)}$ keeps track of the surplus lengths that land outside the interval I_{ij}, to the right of v_j, and $S_1^{'(l)}$ keeps track of all the remaining lengths, i.e. the ones that land and serve deficiencies in $I_{(m+1)j}$.

  Formally, let $S_1^{(r)} = (s_1^{(r)}, \ldots, s_\sigma^{(r)})$, then $S_1^{\prime(r)} = (s_1^{\prime(r)}, \ldots, s_\sigma^{\prime(r)})$ and $S_1^{''(r)} = (s_1^{\prime\prime(r)}, \ldots, s_\sigma^{\prime\prime(r)})$ are constructed as follows. For all $t \in \{1, \ldots, \sigma\}$, we have $s_t^{\prime(r)} = s_t^{(r)}$, if $l_t \leq l(m, j)$, and it's 0 otherwise. Correspondingly, we have $s_t^{\prime\prime(r)} = s_t^{(r)}$, if $l_t > l(m, j)$, and it's 0 otherwise.

Correspondingly, we need to decompose the deficiency vectors of P_1 and P_2 at the breaking point m, i.e. $D_1^{(r)}$ and $D_2^{(l)}$, depending on whether the deficiency is served by surpluses that originate from the interval of the other subproblem or from outside I_{ij}. Consider such a decomposition for $D_1^{(r)}$. A length of $D_1^{(r)}$ does not reveal by itself whether it will be served by a center in $I_{m+1,j}$ or by a center outside and to the right of I_{ij}. This is in contrast to the surplus vectors, as shown above. However, we can still enumerate over all such decompositions for $D_1^{(r)}$ and $D_2^{(l)}$, and assume that one part gets covered by the surplus inside I_{ij}, and the other, by the surplus from outside I_{ij}.

- $D_1^{(r)} = D_1^{'(r)} + D_1^{''(r)}$, where $D_1^{'(r)}, D_1^{''(r)} \in \mathbb{N}^\sigma$ are deficiency vectors. The deficiency lengths in $D_1^{''(r)}$ will get served by the surplus from outside I_{ij}, i.e. a center located outside the interval and to the right of j, and $D_1^{'(r)}$ will get served by a center in $I_{m+1,j}$.
- $D_2^{(l)} = D_2^{'(l)} + D_2^{''(l)}$, where $D_2^{'(l)}, D_2^{''(l)} \in \mathbb{N}^\sigma$ are deficiency vectors. Similar to above, $D_2^{''(l)}$ keeps track of the deficiency lengths corresponding to vertices that get served by a center outside I_{ij} and to the left of i, and $D_2^{'(l)}$ keeps track of the deficiency lengths originating from points in $I_{m,j}$ that are served by centers in $I_{i,m}$.

Note that the surplus and deficiency vectors, $S_1^{(l)}, S_2^{(r)}, D_1^{(l)}$, and $D_2^{(r)}$, by definition, deal with deficiency and surplus that is outside the interval I_{ij}. From the above decomposition, we see that the surpluses $S_1^{'(r)}$ and $S_2^{'(l)}$ are used to cover deficiencies $D_2^{'(l)}$ and $D_1^{'(r)}$ in the same level of the recursion. Thus, we need the surplus to be at least the deficiency as follows. Note that the $\geq$ binary relation for vectors holds when all the $\geq$ relations hold component-wise for its elements.

$$S_1^{'(r)} \geq D_2^{'(l)} \tag{3.4}$$

$$S_2^{'(l)} \geq D_1^{'(r)} \tag{3.5}$$

On the other hand, surpluses $S_1^{''(r)}$ and $S_2^{''(l)}$, and deficiencies $D_2^{''(l)}$ and $D_1^{''(r)}$ are passed up to a higher level of the recursion, and thus integrated with other surplus/deficiency vectors, specifically with $S_2^{(r)}$, $S_1^{(l)}$, $D_1^{(l)}$, and $D_2^{(r)}$, respectively. They will form, $S^{(r)}$, $S^{(l)}, D^{(l)}$, and $D^{(r)}$. Thus, each of the surplus and deficiency vectors at the two ends of the interval I_{ij} originate from two parts, as mentioned above, e.g. $D^{(r)}$ being an aggregate of $D_2^{(r)}$ and $D_1^{''(r)}$. In aggregating $D_2^{(r)}$ and $D_1^{''(r)}$, we need to add a distance of $l(m,j)$ to the lengths in the vector $D_1^{''(r)}$. After this addition, we need to round the lengths again up to a factor $1+\epsilon'/\log n$ to make them in the appropriate format. Similarly, the lengths represented by $D_2^{''(l)}$ will have a distance of $l(i,m+1)$ added to each one of them. Also each length represented in the vectors $S_2^{''(l)}$ and $S_1^{''(r)}$ will have a distance of $l(i,m+1)$ and $l(m,j)$ subtracted from them, respectively, and rounded up to form surplus vectors in the required format. In order to formalize this idea of moving all the lengths of a surplus/deficiency vector and rounding up the result, we define two rounding function, forward and backward, that will add and subtract a certain distance from the lengths represented in vectors, respectively.

The forward and backward rounding functions $r_f, r_b : \mathbb{N}^\sigma \times \mathbb{Q} \to \mathbb{N}^\sigma$ take a surplus/deficiency vector $(n_1, \ldots, n_\sigma)$, and a value l, which is to be added or subtracted; then add the distance l to all the distances of type l_i (in the case of the forward function) represented in the surplus/deficiency vectors, or subtract (in the case of the backward function), and then round up the values to get new surplus/deficiency vectors. For the forward rounding function r_f, $(n'_1, \ldots, n'_\sigma) = r_f((n_1, \ldots, n_\sigma), l)$ is computed as follows. Starting from the all-zero vector $(n'_1, \ldots, n'_\sigma)$, iterate the following for all $t \in \{1, \ldots, \sigma\}$: round up $l_t + l$ to the nearest power of $1 + \epsilon'/\log n$. Let $l_{t'}$ be the resulting length. Add n_t to $n'_{t'}$. At the end, r_f will output the vector $(n'_1, \ldots, n'_\sigma)$. The backward rounding function r_b is defined similarly, where instead of rounding up $l_i + l$, we do this for $l_i - l$. Having developed the above machinery, we can state formally how surplus/vectors at the ends of I_{ij} are formed from other surplus/deficiency vectors.

$$D^{(r)} = D_2^{(r)} + r_f(D_1^{''(r)}, l(m,j)) \tag{3.6a}$$

$$D^{(l)} = D_1^{(l)} + r_f(D_2^{''(l)}, l(i,m+1)) \tag{3.6b}$$

$$S^{(r)} = S_2^{(r)} + r_b(S_1^{''(r)}, l(m,j)) \tag{3.6c}$$

$$S^{(l)} = S_1^{(l)} + r_b(S_2^{''(l)}, l(i,m+1)) \tag{3.6d}$$

(3.6e)

Finally, the number of open centers in each sub-interval should match with the total number of centers opened.

$$k' = k'_1 + k'_2 \tag{3.7}$$

Now, with the decomposition of surplus/deficiency vectors as above, the consistency of P with P_1 and P_2 follows, if all the above conditions (3.4), (3.5), (3.6), (3.7) are satisfied.

Analysis

Suppose $\text{OPT} \leq \text{B} \leq (1+\epsilon) \cdot \text{OPT}$ and let $\mathcal{O}$ be an optimum solution.

Lemma 11 *For each interval V_{ij}, if $\mathcal{O}$ opens k' centers in that interval then if $D^{(r)}, D^{(l)}, S^{(r)}, S^{(l)}$ are surplus/deficiency vectors that approximately (rounded up to nearest factor of $(1 + \epsilon'/\log n)$) show the number of arms going in/out of the interval to serve the points then the entry of the DP corresponding to $(i, j, k', D^{(r)}, D^{(l)}, S^{(r)}, S^{(l)})$ will be True. Conversely, for each such entry, if the computed value is True then there is way to open k' centers in the interval V_{ij} and assign each point in V_{ij} to one of the open centers or send them outside the interval to be served, and send the surplus of each the k' open center to serve points outside V_{ij} such that the profile of the surplus and deficiencies (again rounded up to nearest factor of $(1 + \epsilon'/\log n)$) are exactly $D^{(r)}, D^{(l)}, S^{(r)}, S^{(l)}$ and the cost of each star is at most $(1 + \epsilon'/\log n)^\ell \cdot B$ where ℓ is the level of V_{ij} in the dissection.*

The proof of this lemma follows from a straight-forward induction. The main observation (as pointed out earlier) is that at each level of recursion, the approximation loss we pay due to the rounding in calculations is a factor of $(1 + \epsilon'/\log n)$, and since the depth of recursion (dissection) is $\lceil \log n \rceil$, this will result in an extra factor of $(1 + \epsilon'/\log n)^{\lceil \log n \rceil} \leq (1 + \epsilon)$ for $\epsilon' \leq \log(1 + \epsilon)$.

Consider a subproblem P, we can enumerate over all the possible subproblems P_1 and P_2 by considering all the possible values to the parameters $k_1', D_1^{(r)}, D_1^{(l)}, S_1^{(r)}, S_1^{(l)}, k_2', D_2^{(r)}, D_2^{(l)}, S_2^{(r)}, S_2^{(l)}$ which can be done in $n^{O(\log^2 n/\epsilon)}$. For fixed values of the above, the consistency equations can be checked in time $\mathrm{Poly}(\sigma) = \mathrm{Poly}(\log \mathrm{n})$. Finally, note that the size of the DP table is at most $n^{O(\log^4 n/\epsilon)}$, thus the total running time is quasi-polynomial.

3.4.2 PTAS **for the line metric when Stars are non-crossing**

In this section, we give a PTAS for the star cover problem on the line for the special case that the stars are non-crossing. We call two stars $S_1 = (r_1, C_1)$ and $S_2 = (r_2, C_2)$ to be *crossing*, if there exists leaves $v_1 \in C_1$ and $v_2 \in C_2$ such that the corresponding intervals to the edges (r_1, v_1) and (r_2, v_2) on the line have overlap, i.e. have non-empty intersection and none of the vertices r_1, r_2, v_1 or v_2 are co-located on the line. In other words, two arms (r_1, v_1) and (r_2, v_2) are crossing if one of the arms has exactly one endpoint falling in the interval spanned by the other arm. A set of stars $\mathcal{S}$ is called *non-crossing*, if there doesn't exist any two stars $S_1, S_2 \in \mathcal{S}$ that are crossing.

In this section, we give a dynamic programming algorithm for the k-star cover problem, when the vertices are on a line and the stars are guaranteed to be non-crossing. First we perform the pre-processing and scaling steps from section 3.4.1 and apply following DP on the modified instance.

Dynamic Programming. The DP algorithm for this problem consists of a few tables that we define in turn.

Table T. The subproblems in T are defined as (r, i, j, b, B', k'), where $r, i, j, b \in \{1, \ldots, n\}$ are indices of vertices, $i \le r \le j$, $i < b \le j$, $k' \le k$ and $B' \le B$. Each table entry of T is a boolean value that is set to "True", if the vertices V_{ib} can be covered either by (i) a star rooted at r. The arms emanating from r that cover a subset of V_{ib} incur a cost of at most $B - B'$ on this star. Or (ii) by any of the k' stars that have spans in V_{ib}.

Base case for T. This corresponds to the case when all the vertices in V_{ib} are covered by the star rooted at r. The corresponding table entry is set to "True", if $k' = 0$ and $\sum_{i \le t \le j} l(r, v_t) \le B - B'$. This means that all the vertices in V_{ib} can be covered by a star of cost at most B that is rooted at r.

Recursive step for T. In order to define recursive steps for computing T, we introduce a new table P, which is built using entries from T. Subproblems of P are of the form (i, j, k'), which corresponds to the case that V_{ij} can be covered by k' stars. By definition of T, $P(i, j, k')$ ="True", if there exist $r \in \{i, \ldots, j\}$ and $B' \ge 0$ such that $T(r, i, j, j, B', k' - 1)$ ="True".

With the help of this new table, we get back to the recursive step for filling T. In the case where $T(r, i, j, b, B', k')$ corresponds to a feasible solution, b is the index of the last vertex covered. Each vertex in V_{ib} is covered. We look at the leftmost stretch of vertices, starting from b and to its left, that are all covered either by (i) the star rooted at r, or (ii) by one of stars that has a smaller span that the one rooted at r. We give separate recursive formulas for each case. $T(r,i,j,b,B',k')$ is set to "True" when either one of the following cases holds:

(i) There exist b', k'' such that $P(b'+1,b,k'')$ ="True", and $T(r,i,j,b',B',k'-k'')$ ="True".

(ii) There exists b' such that $T(r,i,j,b',B'+\sum_{b'<t\leq b} l(r,v_t),k')$ ="True".

Lemma 12 *$T(r,i,j,b,B',k')$ is set to "True", if the vertices V_{ib} can be covered either by (i) a star rooted at r, of cost at most $B-B'$, or (ii) by one of the k' stars that have spans in V_{ib}.*

Proof: The proof follows by an induction following the base case and recursive step of the DP used to fill in the entries of T above.

Base case. If $k'=0$, by the base case of the DP, the corresponding entry of T is set to "True", if all the vertices in V_{ib} are covered by the star rooted at r and its cost is at most $B-B'$.

Inductive step. If $k'\geq 1$, there is at least another star covering vertices in V_{ib}. Look at b, if b is covered by the star rooted at r. Let b' be the last vertex in V_{ib} that is not covered by r. Thus $T(r,i,j,b',B'+X,k')$="True", where $X=\sum_{b'<t\leq b} l(r,v_t)$ is the cost of covering $V_{b'+1,b}$ by r. This corresponds to case (ii) in the recursive step of the DP. If b is not covered by r, it must be covered by a smaller star S'. Let $[v_{b'+1},v_b]$ be the span of S' and k'' be the number of stars covering the vertices $V_{b'+1,b}$. Then, $P(b'+1,b,k'')$ ="True", and $T(r,i,j,b',B',k'-k'')$ ="True". ■

Note that entries of T and P can be computed in non-decreasing order of $j-i$ and subsequently of k'. Having computed P, we need yet another DP to compute the solution to the k-star cover problem. We construct the table D where the subproblems are of the form $D(i,k'), 0\leq i\leq n$, and $k'\leq k$. The entry $D(i,k')$ ="True" means that V_{1i} can be covered by k' stars. The base case is the entry with $i=k'=0$, and is set to "True". Otherwise, $D(i,k')$ is set to "True" when there exists $b\leq i$ and k_1,k_2 such that $D(b,k_1)$ ="True", $P(b+1,i,k_2)$ ="True" and $k_1+k_2\leq k'$. The correctness of computing D follows by an inductive argument similar to the one for T shown above.

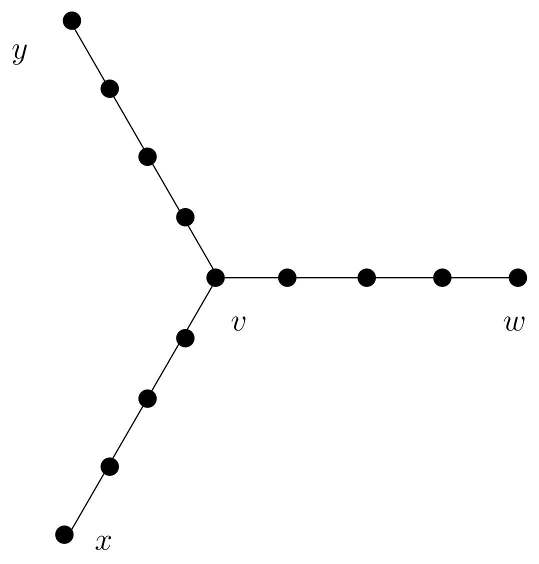
3.5 Euclidean Metric

The approximation schemes for the line metric, presented in section 3.4 provide strong evidence that the k-star cover problem on the line admits a PTAS. Thus, the natural question that arises is exploring the possibility of a PTAS for slightly more general metrics. In the following, we rule out such a possibility for the Euclidean metric.

3.5.1 APX-hardness

In this section, we show that the Min-max k-star cover problem on the Euclidean plane is APX-hard. The reduction is similar to the APX-hardness proof of k-center, given by [17]. We give a gap-introducing reduction from the dominating set problem on planar graphs with maximum degree 3. The latter problem is known to be NP-hard [20]. A planar graph of degree at most 3 can be

Figure 3.1: A degree-3 vertex v and its three neighbors w, x, y are shown. Each edge is replaced with a path of length 4.

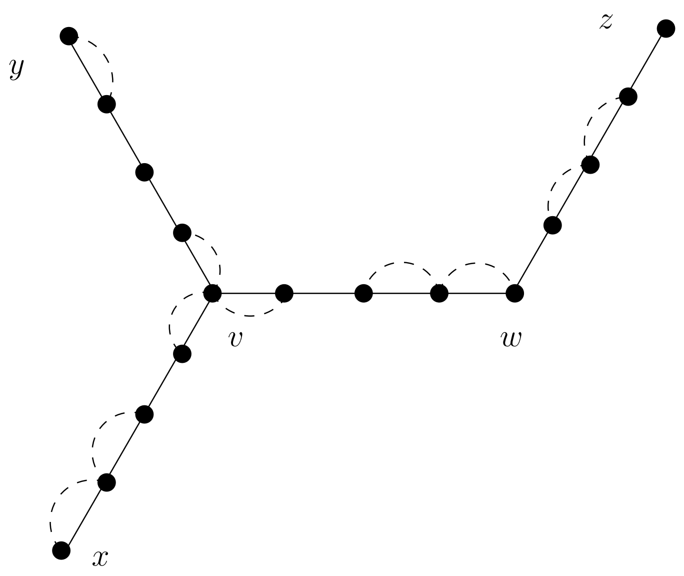


embedded into the plane, where each edge e is replaced with a path $p(e)$ of edges of length 1, such that the length of the path $|p(e)|$ is $3i + 1$, for some i. Each degree 3 vertex will have its three edges at 120 degree angles from each other. Let H be this new graph. The Embedding of one degree-3 vertex and its three neighbors has been shown in Figure 3.1. Note that each of the edges $\{v, w, \}, \{v, x\}, \{v, y\}$ is replaced by a path of length 4.

Lemma 13 *Given an undirected planar graph G with maximum degree 3, and an integer* $d \in \mathbb{Z}^+$. *G has a dominating set of size at most d, if and only if H has a k-star cover of cost at most* 3, *for* $k = d + \sum_e \frac{|p(e)|-1}{3}$.

Proof: If G has a dominating set D of cardinality d, we build a k-star cover of H as follows. Choose each vertex $v \in D$ as a root and form a star of cost at most 3 covering all its neighboring vertices, e.g. vertex v covering its three neighbors and vertex y covering its one neighbor in Figure 3.2. Now, consider the remaining vertices on each path $p(e)$. Every three consecutive vertices on this path can be covered by a star of cost 2. As $|p(e)| = 3i + 1$, for some i, each such path needs $(|p(e)| - 1)/3$ stars of cost at most 3 to have all the vertices on the $p(e)$ covered, irrespective of whether or not the endpoints are in D. The three different cases have been depicted in Figure 3.2. (i) Both endpoints $v, y \in D$. The middle vertex of $p(\{(v, y)\})$ is not covered by stars rooted at either of v or y and is thus covered by a singleton star of cost 0. (ii) Each of the edges $\{v, x\}$ and $\{v, w\}$ have exactly one endpoint $v \in D$ and the other endpoint $x, w \notin D$. Thus, there is one star of cost 2 covering two of the vertices on each of the paths $p(\{v, x\})$ and $p(\{v, w\})$. This star also covers the endpoint not in the dominating set, i.e. x and w respectively. And (iii) Neither of the endpoints of the edge $\{w, z\}$ are in D. Thus, all the vertices on the corresponding path $p(\{w, z\})$

Figure 3.2: A star cover of cost at most 3 in H. The dominating set of G includes v and y, but not w, x, or z. The dotted lines show the arms of the star cover. The middle node on the path $p(\{v, w\})$ is a singleton stars.



can be covered by stars of cost 3. Note that in all the three cases above, we need exactly one star to cover the vertices on the paths $p(e)$.

Conversely, assume that vertices of H can be covered by k stars of cost at most 3. We show that G has a dominating set of size $k - \sum_e (|p(e)| - 1)/3$. Note that each star of cost at most 3 in H has one of the following structures: (a) a single edge, or a single vertex, (b) a vertex covering its two neighbors, or (c) a vertex covering its three neighbors. Consider the internal vertices of $p(e)$, i.e. all the vertices of $p(e)$, minus the two endpoints. There are at least $(|p(e)| - 1)/3$ stars of type (b) or (a) needed to cover these vertices. This bound is true even if two stars of type (c) cover one point each from the two internal vertices of distance 1 from the two vertices at the endpoint of path $p(e)$. Delete all the stars rooted at the internal vertices of all the paths; these will be stars of type (b) or (a). Let S be the set of roots of remaining stars (which we can assume are all of type (c)). Since there are at least $\sum_e (|p(e)| - 1)/3$ stars removed, $|S| \leq k - \sum_e (|p(e)| - 1)/3$. It remains to show that S forms a dominating set in G.

Consider an edge $e = \{u, v\}$ in G and the corresponding path $p(e)$ in H. If neither of u or v have stars rooted at them (in S), then we show that u, v are adjacent to vertices that have stars rooted at them in S. Suppose neighbor u nor v have stars rooted at them, then they are covered by stars of type (b) or (a) in H. Let $e' = \{u, u'\}$ and $e'' = \{v, v'\}$ be two other edges incident to u and v (in G), respectively. Covering $p(e')$ and $p(e'')$ by $(|p(e')| - 1)/3$ and $(|p(e'')| - 1)/3$ stars of type (b), respectively, each will have two vertices at the end left over that need to be covered by a star of type (c), i.e. $u', v' \in S$; thus S is a dominating set. ■

For a YES instance (G, d) of dominating set, H has a k-star cover of cost 3, as shown above. In the NO case, the cardinality of the dominating set of G is at least $d + 1$, and thus H needs at least $k + 1$ stars of cost at most 3 to cover all its vertices. To cover H with k stars, we either need to include a star rooted at an internal vertex covering three other vertices, instead of two, or a star rooted at an endpoint of $p(e)$ covering four other vertices, instead of three. In either case, the cost of the star will be at least 4. Hence, we get the following theorem.

Theorem 14 *It is NP-hard to α-approximate the min-max k-star cover problem on the Euclidean plane, for any $\alpha < 4/3$.*

Chapter 4

Conclusion

In this thesis, we studied a few covering problems arising from the area of vehicle routing from the standpoint of approximation algorithms. We will review the results in the following and distinguish some branches for future work.

4.1 Summary

The problems studied in this thesis concerned with covering graphs with two specific subgraphs, specifically tours and stars. We considered both rooted and unrooted versions of tour cover and star cover problems. The results we presented in the thesis are summarized in the following.

- We presented a $(7+\epsilon)$-approximation algorithm for the rooted min-max k-tour cover problem in Section 2.3.
- We gave a $(16/3+\epsilon)$-approximation algorithm for the unrooted min-max k-tour cover problem in Section 2.4.
- We generalized the bi-criteria approximation of [3] and showed that it gives an $(O(\frac{1}{\epsilon}), 1+\epsilon)$ bi-criteria approximation for the unrooted min-max k-star cover problem in Section 3.3.
- We presented a QPTAS for the unrooted min-max k-star cover problem on the line metric in Section 3.4.1.
- We designed a PTAS for the unrooted min-max k-star cover problem on the line metric, when the stars are non-crossing, in Section 3.4.2.
- We ruled out the possibility of a PTAS for the unrooted min-max k-star cover problem in the Euclidean metric by presenting an APX-hardness reduction in Section 3.5.

4.2 Future Directions

The whole area of vehicle routing problems is vast and fertile ground for approximation algorithms with relatively little work done to this date. In the following however, we will focus on covering problems and specifically those discussed in this thesis.

For the tour cover problems, the main open problem is to improve on the approximation ratios presented in Chapter 2, and also to improve the lower bounds on their approximability ratios. Another direction for future research is the consideration of more real-world constraints mentioned in Chapter 1, such as capacities on each tour, and time-windows on the coverage time of each vertex.

For the star cover problems, it is an interesting question to close the gap on the approximability of rooted k-star cover problem, which sits somewhere in $(3/2, 2]$. The unrooted k-tour cover problem in turn presents a few open problems. The most important one is to give any non-trivial approximation for the problem on a general metric.

In more restricted metrics, it would be interesting to give a PTAS on the line metric. We hope that the techniques developed in Section 3.4 will lead the way to such approximation schemes. An $O(1)$-approximation for the Euclidean metric has not been ruled out, and we conjecture that an algorithm satisfying such a ratio is possible.

Bibliography

[1] H.C. An, R. Kleinberg, and D.B. Shmoys. Improving christofides' algorithm for the st path tsp. *Arxiv preprint arXiv:1110.4604*, 2011.

[2] David L Applegate, Robert E Bixby, Vasek Chvatal, and William J Cook. *The traveling salesman problem: a computational study*. Princeton University Press, 2011.

[3] E.M. Arkin, R. Hassin, and A. Levin. Approximations for minimum and min-max vehicle routing problems. *Journal of Algorithms*, 59(1):1–18, 2006.

[4] Sanjeev Arora, Prabhakar Raghavan, and Satish Rao. Approximation schemes for euclidean k-medians and related problems. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 106–113. ACM, 1998.

[5] V. Arya, N. Garg, R. Khandekar, A. Meyerson, K. Munagala, and V. Pandit. Local search heuristics for k-median and facility location problems. *SIAM Journal on Computing*, 33(3):544–562, 2004.

[6] Yair Bartal, Moses Charikar, and Danny Raz. Approximating min-sum k-clustering in metric spaces. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 11–20. ACM, 2001.

[7] J Bondy and U Murty. *Graph theory (Graduate texts in mathematics, Vol. 244)*. New York: Springer, 2008.

[8] M. Charikar and R. Panigrahy. Clustering to minimize the sum of cluster diameters. In *STOC*, volume 33, pages 1–10, 2001.

[9] Moses Charikar, Sudipto Guha, Éva Tardos, and David B Shmoys. A constant-factor approximation algorithm for the k-median problem. In *Proceedings of the thirty-first annual ACM symposium on Theory of computing*, pages 1–10. ACM, 1999.

[10] N. Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem., 1976.

[11] William Cook. *In pursuit of the traveling salesman: mathematics at the limits of computation*. Princeton University Press, 2011.

[12] T.G. Crainic and G. Laporte. *Fleet management and logistics*. Kluwer Academic Pub, 1998.

[13] G.B. Dantzig and J.H. Ramser. The truck dispatching problem. *Management science*, pages 80–91, 1959.

[14] W Fernandez de la Vega, Marek Karpinski, Claire Kenyon, and Yuval Rabani. Approximation schemes for clustering problems. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 50–58. ACM, 2003.

[15] G. Even, N. Garg, J. Könemann, R. Ravi, and A. Sinha. Covering graphs using trees and stars. *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 24–35, 2003.

[16] G. Even, N. Garg, J. Könemann, R. Ravi, and A. Sinha. Min-max tree covers of graphs. *Operations Research Letters*, 32(4):309–315, 2004.

[17] Tomás Feder and Daniel Greene. Optimal algorithms for approximate clustering. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 434–444. ACM, 1988.

[18] G.N. Frederickson, M.S. Hecht, and C.E. Kim. Approximation algorithms for some routing problems. *SIAM Journal on Computing*, 7:178, 1978.

[19] AM Frieze. An extension of christofides heuristic to the k-person travelling salesman problem. *Discrete Applied Mathematics*, 6(1):79–83, 1983.

[20] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. WH Freeman & Co., 1979.

[21] S.O. Gharan, A. Saberi, and M. Singh. A randomized rounding approach to the traveling salesman problem. In *FOCS*, 2011.

[22] B.L. Golden, S. Raghavan, and E.A. Wasil. *The vehicle routing problem: latest advances and new challenges*. Springer Verlag, 2008.

[23] A. Gupta and K. Tangwongsan. Simpler analyses of local search algorithms for facility location. *Arxiv preprint arXiv:0809.2554*, 2008.

[24] Anupam Gupta, Robert Krauthgamer, and James R Lee. Bounded geometries, fractals, and low-distortion embeddings. In *Foundations of Computer Science, 2003. Proceedings. 44th Annual IEEE Symposium on*, pages 534–543. IEEE, 2003.

[25] Dorit S Hochbaum. *Approximation algorithms for NP-hard problems*. PWS Publishing Co., 1997.

[26] D.S. Hochbaum and D.B. Shmoys. A unified approach to approximation algorithms for bottleneck problems. *Journal of the ACM (JACM)*, 33(3):533–550, 1986.

[27] JA Hoogeveen. Analysis of christofides' heuristic: Some paths are more difficult than cycles. *Operations Research Letters*, 10(5):291–295, 1991.

[28] Russell Impagliazzo and Ramamohan Paturi. Complexity of k-sat. In *Computational Complexity, 1999. Proceedings. Fourteenth Annual IEEE Conference on*, pages 237–240. IEEE, 1999.

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

[30] Tapas Kanungoa, David M Mountb, Nathan S Netanyahuc, Christine D Piatkoe, Ruth Silvermand, and Angela Y Wuf. A local search approximation algorithm for k-means clustering. *Computational Geometry*, 28:89–112, 2004.

[31] M. R. Khani and M. R. Salavatipour. Improved approximation algorithms for the min-max tree cover and bounded tree cover problems. In *APPROX*, 2011.

[32] Eugene L Lawler, Jan Karel Lenstra, AHG Rinnooy Kan, and David B Shmoys. *The traveling salesman problem: a guided tour of combinatorial optimization*, volume 3. Wiley Chichester, 1985.

[33] J.K. Lenstra, D.B. Shmoys, and É. Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming*, 46(1):259–271, 1990.

[34] Shi Li and Ola Svensson. Approximating k-median via pseudo-approximation. In *Symposium on Theory of Computing (STOC)*, 2013.

[35] T. Mömke and O. Svensson. Approximating graphic TSP by matchings. In *FOCS*, 2011.

[36] Marcin Mucha. $\frac{13}{9}$-approximation for graphic tsp. *Theory of Computing Systems*, pages 1–18, 2012.

[37] C.H. Papadimitriou and S. Vempala. On the approximability of the traveling salesman problem. *Combinatorica*, 26(1):101–120, 2006.

[38] Kunal Talwar. Bypassing the embedding: algorithms for low dimensional metrics. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 281–290. ACM, 2004.

[39] Arie Tamir. An $o(pn^2)$ algorithm for the p-median and related problems on tree graphs. *Operations Research Letters*, 19(2):59–64, 1996.

[40] P. Toth and D. Vigo. *The vehicle routing problem*, volume 9. Society for Industrial Mathematics, 2002.

[41] Vijay V. Vazirani. *Approximation Algorithms*. Springer-Verlag, 2001.

[42] D.P. Williamson and D.B. Shmoys. *The design of approximation algorithms*. Cambridge University Press, 2011.

[43] Z. Xu and B. Rodrigues. A 3/2-approximation algorithm for multiple depot multiple traveling salesman problem. *Algorithm Theory-SWAT 2010*, pages 127–138, 2010.

[44] Zhou Xu, Dongsheng Xu, and Wenbin Zhu. Approximation results for a min–max location-routing problem. *Discrete Applied Mathematics*, 160(3):306–320, 2012.