Abstract of "Approximation Schemes for Euclidean Vehicle Routing Problems." by Aparna Das, Ph.D., Brown University, May, 2011.

Vehicle routing is a class of optimization problems where the objective is to find low cost delivery routes from depots to customers using vehicles of limited capacity. Vehicle routing problems generalize the traveling salesman problem and have many real world applications to businesses with high transportation costs such as waste removal companies, newspaper deliverers, and food and beverage distributors.

We study two basic vehicle routing problems: *unit demand* routing and *unsplittable demand* routing. In the unit demand problem, items must be delivered from depots to customers using a vehicle of limited capacity and each customer requires delivery of a single item. The unsplittable demand problem is a generalization, where customers can have different demands for the number of items, but each customer's entire demand must be delivered all together by one route.

Both problems are NP-Hard and do not admit better than constant factor approximation algorithms in the metric setting. However in many practical settings the input to the problem has Euclidean structure. We show how to exploit this to design arbitrarily good approximation algorithms. We design a quasi-polynomial time approximation scheme for the Euclidean unit demand problem in constant dimensions, and asymptotic polynomial time approximation schemes for the unsplittable demand problem in one dimension.

Approximation Schemes for Euclidean Vehicle Routing Problems.

by

Aparna Das

B. S., Cornell University, 2001

M. S., University of Wisconsin, 2005

Sc. M., Brown University, 2007

A dissertation submitted in partial fulfillment of the
requirements for the Degree of Doctor of Philosophy
in the Department of Computer Science at Brown University

Providence, Rhode Island

May, 2011

This dissertation by Aparna Das is accepted in its present form by
the Department of Computer Science as satisfying the dissertation requirement
for the degree of Doctor of Philosophy.

Date ________________ ______________________________________

Claire Mathieu, Director

Recommended to the Graduate Council

Date ________________ ______________________________________

Philip Klein, Reader

Date ________________ ______________________________________

Pascal Van Hentenryck, Reader

Approved by the Graduate Council

Date ________________ ______________________________________

Peter Weber
Dean of the Graduate School

Acknowledgements

I owe the deepest gratitude to my advisor Claire Mathieu. Her support and guidance in my research, my career, and life in general, have been invaluable to me. I thank her for always having my best interests in mind. Her generosity and positive outlook continue to impress me and I hope to one day be as good a role model for someone as she is for me.

I thank Philip Klein for serving on my research comps and thesis committees and for his ongoing interest in my work. I am grateful for his feedback and encouragement, and for always pushing me to improve.

I thank Pascal Van Hentenryck for serving on my thesis committee, for his advice on new research topics, and for inspiring me with his wonderful course in optimization.

I am grateful to Lisa Hellerstein for being my very first research advisor and for encouraging me to do research and apply to graduate school.

I also thank the computer science faculty at Brown, especially Amy Greenwald, David Laidlaw, Anna Lysyanskaya, John Savage and Eli Upfal for taking an interest in my work and in me over these past years.

I also thank my collaborators, Matthew Cary, Ioannis Giotis, Anna Karlin, Shay Mozes, and Daniel Ricketts, for allowing me to have supportive and stimulating research experiences.

I feel very lucky to have met my wonderful office mates and friends in the department. I like to thank them for listening to my practice talks, providing me with encouragement and for allowing me to have a life at Brown outside of research.

Lastly, I would like to thank my family for their unconditional support. I am grateful to my husband for his patience, honesty and for believing in me. I am forever indebted to my parents for their dedication and for being our eternal advocates. And finally, to my sister for her short and insightful pieces of advice and for always leading by example.

Contents

1 Introduction **1**

1.1 Problem Definitions 4

1.2 Overview of Results 5

1.3 Techniques 7

1.3.1 Algorithm Techniques 7

1.3.2 Bounds for Analysis 7

2 Traveling Salesman Problem: Review of Arora's Algorithm [2] **10**

2.1 Introduction 10

2.2 Algorithm and Proof of Main Theorem 11

2.2.1 Preprocessing 12

2.2.2 The Structure Theorem 13

2.2.3 Proof of the Main Theorem 13

2.3 The Dynamic Program 14

2.4 Computing a Structured Tour 15

2.4.1 Patching 15

2.4.2 Portal Respecting 18

2.5 Proof of the Structure Theorem 19

2.5.1 A First Approach 19

2.5.2 Properties of New Crossings 21

2.5.3 Correctness of Algorithm 2 23

2.6 Extension to Higher Dimensions 24

3 Vehicle Routing: Unit Demands and Single Depot **25**

3.1 Introduction 25

3.2 Algorithm and Proof of Main Theorem 27

3.2.1 The Structure Theorem 27

3.2.2 Assigning Types 30

3.2.3 A Constant Factor Approximation [29] 30

3.2.4 Proof of Main Theorem 3.1.1 32

3.3 Proof of the Structure Theorem 3.2.6 32
3.3.1 Proof of Lemma 3.3.1 33
3.3.2 Proof of Lemma 3.3.3 36
3.4 The Dynamic Program 37
3.5 Proof of Theorem 3.2.10 40
3.5.1 Proof of Lemma 3.5.1 40
3.5.2 Proof of Lemma 3.5.2 41
3.6 Derandomization 46
3.7 Extension to Higher Dimensions 46

4 Extension to Multiple Depots 47
4.1 Introduction 47
4.2 Preliminaries 48
4.3 Algorithm and Proof of Main Theorem 49
4.3.1 Partitioning into Sub-Instances 50
4.3.2 Preprocessing 51
4.3.3 Extending the Structure Corollary 52
4.3.4 A Constant Factor Approximation for Multiple Depots 52
4.3.5 Proof of Main Theorem 4.1.1 53
4.4 Extending the Dynamic Program 54
4.5 Proof of Theorem 4.3.5 54
4.5.1 Proof of Lemma 4.5.2 54

5 Vehicle Routing: Unsplittable Demands, Single Depot in One Dimension 56
5.1 Introduction 56
5.2 Preliminaries 59
5.3 Algorithm and Proof of Main Theorem 60
5.3.1 Rounding 60
5.3.2 Partitioning into Regions 61
5.3.3 Solving a Region 61
5.3.4 Proof of Main Theorem 5.1.1 64
5.4 Proof of Lemma 5.3.1 65
5.5 Proof of Lemma 5.3.3 67
5.6 Proof of Lemma 5.3.7 68
5.7 Proof of Lemma 5.3.8 69

6 Extension to Constant Number of Depots 71
6.1 Introduction 71
6.2 Algorithm and Proof of Theorem 6.1.1 72
6.2.1 The Unsplit Problem Between Two Depots. 73

6.2.2 Proof of Main Theorem 6.1.1 74
6.3 Algorithm for Unsplit Between Depots and Proof of Theorem 6.3.1 75
6.3.1 Rounding 75
6.3.2 Partitioning into Regions 77
6.3.3 Solving the Middle Region 77
6.3.4 Proof of Theorem 6.3.1 79
6.4 Proof of Lemma 6.3.2 81
6.5 Proof of Lemma 6.3.4 82
6.6 Proof of Lemma 6.3.6 83
6.7 Proof of Lemma 6.3.7 84

7 Many Customers and Restricted Capacity 86
7.1 Introduction 86
7.2 Algorithm and Proof of Main Theorem 87
7.2.1 Partitioning into $close$ and far Instances 88
7.2.2 Solving the far instance 89
7.2.3 Proof of Main Theorem 7.1.1 91
7.3 Proof of Lemma 7.2.2 92
7.4 Proof of Lemma 7.2.4 93

8 Conclusion and Open Questions 95

Bibliography 98

Chapter 1

Introduction

To highlight the impact of truck distribution and delivery on our daily lives, Joseph Malkevitch, Professor of Mathematics and Computing, writes

> *" Love may make the world go round but what makes our daily lives so much easier are trucks. Trucks deliver the food that we eat to the supermarkets ... they deliver the gasoline to the stations where we refill the gas tanks ... and they are responsible for a huge part of the movement of supplies..."* [39].

And what allows truck delivery to work so well? Joseph Malkevitch reveals that *"what helps make it possible for trucks to carry out all the deliveries they do, is mathematics"*.

The *truck dispatching* problem was first introduced and mathematically analyzed in 1959 by Dantzig and Ramser [19] where the goal was to find low cost delivery routes from depots to customers using a truck of limited capacity. Surprisingly, the authors admitted that *"no practical applications have been made as yet"* for the problem, but went on to describe a linear programming algorithm for it whose *"calculations may be readily performed by hand or automatic digital computing machine"* [19].

With the passing of 50 years and numerous practical applications later, the truck dispatching problem is recognized as the first mathematically formulated *vehicle routing problem*, which now describes a large class of optimization problems with a diverse set of constraints. All vehicle routing problems (VRPs) involve finding routes using a vehicle of limited capacity and typically the objective is to minimize the total cost of the routes. VRPs have been widely studied by researchers in computer science and in operations research. Several popular approximation algorithms exist for VRPs as well as general-purpose heuristics such as local search, tabu search, genetic algorithms, neural networks and ant colony optimization schemes. See [37, 50] for details. Several books are also devoted to VRPs *e.g.* [50] and [28], among others.

The real world applications of vehicle routing problems occur in businesses involved in transportation, distribution and delivery that have high transportation costs. This includes companies such as package and newspaper delivery companies, and food and beverage distributors e.g. companies like UPS, FedEx and Peapod. Towns and cities also solve VPRs for designing sanitation pickup routes and school bus routes. Toth and Vigo report on several cases where business saved between 5 and 20% of total costs utilizing computerized models to solve their vehicle routing problems [50]. The February 2010 issue of OR/MS Today magazine surveyed sixteen commercially available software for VRPs from vendors such as IBM, Jeppesen (a Boeing Company), UPS Logistics Technologies and Route Solutions, and their accompanying article stated that *"Routing software is being used in an increasingly diverse set of industries."* [44].

Problems studied. In this work we focus on two basic vehicle routing problems: *unit demand routing* (UnitDem), and *unsplittable demand routing* (Unsplit). In both problems items are located at some depot (or depots) and must be delivered to customers using vehicles that can carry a limited number of items, for example k items, at any time. The goal is to find the minimum cost set of distribution routes for the vehicle to make all the deliveries, such that each route starts and ends at some depot. In the UnitDem problem , all items are identical and each customer requires delivery of a single item. In the Unsplit problem customers can have different demands for the number of items, but each customer must receive their entire demand at once, in other words splitting a customer's delivery among multiple delivery routes is not allowed.

The UnitDem problem applies directly in settings where all customers have identical demands, for example delivery of the daily newspaper. It also models the setting where customers have different demands but their demands are allowed to be split among multiple delivery routes, for example gas stations receiving their daily shipment of gasoline in multiple deliveries throughout the day. If demands are splittable, a customer with demand w is modeled by w customers with unit demand[1].

In contrast, the Unsplit problem models scenarios where each customer's entire demand must arrive in one delivery. For example residential customers probably prefer to receive their entire grocery order, or their entire take out order in one delivery. The unsplittable constraint introduces *bin-packing* features into the routing problem and thus requires more sophisticated solution techniques than the UnitDem problem.

Having multiple depots models delivery companies with several branches for example a grocery chain with multiple locations, all containing products for delivery. The multiple depot setting also has applications to the design of telecommunication networks where user nodes must be connected to one of many possible hubs using links with limited capacities [38].

Hardness of VRPs. While VRPs have abundant applications, unfortunately almost all of them are computationally hard i.e. NP-Hard [30], thus it is unlikely that we can find algorithms to solve these problems optimally in reasonable time. The hardness of VRPs stem from the hardness of the *traveling salesman problem* (TSP) where the goal is to find the minimum cost delivery route with a vehicle of infinite capacity. Almost all VRPs are at least as hard as the TSP. Even stronger hardness results exist for VRPs with the unsplittable constraint stemming from the hardness of the *bin packing* problem.

[1]This reduction works in polynomial time only if the maximum customer demand is $poly(n)$.

One way researchers have tried to tackle VRPs is via approximation algorithms, which do not generally find the optimal solution but are guaranteed to find a solution which is *close* to optimal. Hardness results from general (non-metric) TSP show that it is NP-Hard to approximate the general UnitDem and Unsplit problems to any constant factor [49]. In the *metric* setting (i.e assuming triangle inequality) all three problems admit constant factor approximations. The constant factor approximations for UnitDem and Unsplit are based on an algorithm by Haimovich and Rinnooy Kan from 1985 [29], which partitions a tour of *all* the customers into smaller parts such the total number of items demanded in each part is at most the vehicle capacity.

For NP-Hard problems, the best approximation guarantees are provided by *approximation schemes*. These allow the optimal solution to be approximated with arbitary precision by spending more computation time. Unfortunately hardness results from TSP imply that metric UnitDem and Unsplit, as well as many other VRPs, do not admit *polynomial time approximation schemes* unless P=NP, i.e they are APX-Hard. Thus constant factor approximations are most likely the best possible in the metric setting of these problems.

One setting where it may be possible to do better for the UnitDem problem is when the input is modelled as having additional geometric structure than just metric structure. The hardness results above do not necessarily apply in geometric settings and we may be able to exploit the additional structure to design polynomial time approximation schemes. The TSP is a prominent example where better approximation algorithms are possible under geometic settings. The metric setting of TSP admits no approximation scheme (assuming P≠NP), but approximation schemes have been designed for the *Euclidean* setting [2, 40] and the *planar graph* setting [34].

Hardness results from bin packing imply that the Unsplit problem does not admit polynomial time approximation scheme unless P=NP, even in one dimension [27]. However this does not rule out an *asymptotic* approximation scheme, which performs like a polynomial time approximation scheme when the cost of optimal solution is large. Asymptotic approximation schemes are known for the bin packing problem.

Thesis statement. In this thesis, we study the Euclidean setting of the UnitDem problem in small dimension and the 1-dimensional setting of the Unsplit problem. We design approximation algorithms with improved guarantees over the metric case for both of these settings leading to the thesis statement:

> *Arbitrarily good approximation guarantees are possible by exploiting Euclidean properties in vehicle routing problems.*

In the Euclidean setting all the customers and the depots lie in $\mathcal{R}^d$ for small dimension d and distances are given by the ℓ_2 norm. In the 1-dimensional setting they lie on the line and distances are given by the ℓ_1 norm. The Euclidean setting is often a good model for practical problems. In fact many of the common test beds for VRPs are 2-dimensional Euclidean instances. For example all the instances listed on *researchandpractice.com* and more than half of the instances on *webvrp.com* are 2-dimensional and Euclidean [53, 54]. While the 1-dimensional setting of the Unsplit problem has a limited number of applications in routing and distribution it has important applications in job scheduling. See Chapter 5 for more details.

The typical criticism about approximation schemes is that they are often elaborate algorithms with big running times and thus they are often regarded only as theoretical tools that are useful to pinning down the complexity of a problem. This is true in the case of our approximation scheme for Euclidean UnitDem (Chapters 3, and 4) which runs in *quasipolynomial* time. While our algorithm provides strong evidence that a polynomial time approximation scheme should also be possible, it is still an open question, and remains an active line of research. If a polynomial time approximation scheme can be designed it will show that the UnitDem problem is computationally equivalent to TSP, i.e. the capacity of the vehicle is irrelevant in terms of computational hardness. Our approximation schemes for the Unsplit problem are reasonably efficient and simple. Thus we believe these can be used in practice.

Designing theoretically justified algorithms are also useful for designing new algorithmic techniques and validating existing heuristics. In the process of designing approximation schemes we are forced to identify parts of the problem where we can afford to a compute a coarse solution versus parts where the solution must be computed more precisely. These types of insights are often useful to designing heuristics that can be used in real software. For example our algorithm for UnitDem suggest that a coarse solution can be computed in regions containing many customers which require service from large number of routes. Our algorithm for Unsplit suggests that a coarse solution may be computed in regions containing many large demands.

Designing approximation schemes for VRP problems also requires developing techniques to handle the *global* vehicle capacity constraint. In contrast TSP has no such global containt. Thus in the case of TSP we can afford to spend most of the computational effort on finding very good *local* solutions which can be pasted together, without the risk of violating any global constraint. Things are not as simple for the VRP where it is sometimes better to choose a suboptimal local solution due to the capacity constraint. Developing techniques for handling global constraints such as the vehicle capacity may be useful to solving other graph problems.

1.1 Problem Definitions

We formally define the problems studied in this thesis and state definitions that will be used throughtout.

Euclidean traveling salesman problem (TSP). Given a set of customers C represented as n points in $\mathcal{R}^d$, where d is a small constant independent of n, find the shortest length tour that visits all customers in C.

Euclidean unit demand vehicle routing problem (UnitDem). Given a positive integer k denoting the vehicle capacity, a set C of n customers and a set D of depots, such that C and D are points in $\mathcal{R}^d$ where d is a small constant independent of n, find a collection of tours of minimum total length covering all customers in C, such that each tour in the collection starts and ends at a depot and covers at most k customers.

D contains only one depot in the *single depot* setting and multiple depots in the *multiple depot* setting.

One dimensional unsplittable demand vehicle routing problem (Unsplit). Given a positive integer k denoting the vehicle capacity, a set $C = \{(p_i, w_i)\}_{i \leq n}$ of n customers each with a position p_i on the line and a demand $w_i \leq k$, and a set of m depots D each represented also by a position on the line, find a collection of tours of minimum total length covering the demands of all customers in C, such that each tour starts and ends at some depot, delivers at most k demand and such that no customer's demand is split up among multiple tours.

D contains only one depot in the *single depot* setting and multiple depots in the *multiple depot* setting. In the setting with *restricted capacity*, $k \leq n$.

Geometric definitions. In the *metric* setting the distance between all nodes (i.e customers and depots) satisfy the triangle inequality. In the *Euclidean* setting all nodes lie in $\mathcal{R}^d$ for some constant d and distances are given by the ℓ_2 norm. In the 1-dimensional setting all nodes lie on a line and distances are given by the ℓ_1 norm.

1.2 Overview of Results

We define approximation algorithms and approximation schemes and give a high level overview of our results.

Approximation schemes. Consider any minimization problem and let OPT denote its optimal cost. An algorithm $\mathcal{A}$ for the problem is a c-approximation if it outputs a solution with cost A such that $A \leq c \cdot \text{OPT}$. Algorithm $\mathcal{A}$ is an *approximation scheme* if it is a c-approximation *for all* error parameters $c > 0$. Approximation schemes are particularly designed for the setting where $c \in (0, 1)$, so c is typically denoted by ϵ to emphasize that it is small. Thus for all error parameters $\epsilon \in (0, 1)$ an approximation scheme's output has cost at most $(1 + \epsilon)\text{OPT}$. In fact, $\mathcal{A}$ is considered an approximation scheme as long as its output has cost $(1 + O(\epsilon))\text{OPT}$, as in that case, running $\mathcal{A}$ with error parameter $\epsilon/O(1)$ outputs a solution of cost $(1 + \epsilon)\text{OPT}$ as desired.

A *polynomial time approximation scheme* (PTAS) is an approximation scheme whose running time is polynomial in the size of the instance but can depend arbitrarily on ϵ, and usually grows exponentially in $1/\epsilon$. A *fully polynomial time approximation scheme* (FPTAS) has a running time that is polynomial in the size of the input and in $1/\epsilon$. A *quasi polynomial time approximation scheme* (QPTAS) has running time that is quasipolynomial in the size of the problem instance; if the instance is of size n, it runs in time $n^{\log^{O(1)} n}$.

An *asymptotic* approximation scheme outputs, for every $\epsilon > 0$, a solution of cost at most $(1 + \epsilon)\text{OPT} + \alpha$ where α is a constant independent of the size of the instance. It is "asymptotic" as when

	UnitDem VRP		Unsplit VRP	
	Single depot	Multiple depots	Single depot	Constant depots
Approx. ratios	Metric: 2.5 [29] Euclidean: $2+\epsilon$ [29, 2]	Metric: 5.5 [38] Euclidean: $4+2\epsilon$ [38, 2]	Metric: 3.5 [30] 1-dim: 1.75 [55]	Metric: 7 [38, 30] 1-dim: 6 [38, 30]
Approx. schemes	Metric: APX [7] Euclidean: PTAS for $k = \Omega(n)$ [2, 7], $k \leq 2^{\log^{f(\epsilon)} n}$ [1] QPTAS all k Chap. 3	Metric: APX [7] Euclidean: PTAS for $k = \Omega(n)$ [2], $k, m \leq \frac{\log n}{\log \log n}$ [14] QPTAS all k Chap. 4	Metric: APX [7] 1-dim: APTAS Chap. 5 1-dim: PTAS for $k = poly(n)$ Chap. 7	Metric: APX [7] 1-dim: APTAS Chap. 6

Table 1.1: Status of UnitDem and Unsplit VRP problems.

OPT is large i.e OPT $> \alpha/\epsilon$, the cost of the output is at most $(1+\epsilon)\text{OPT} + \epsilon\text{OPT} = (1+2\epsilon)\text{OPT}$. Thus using the asymptotic approximation scheme with error parameter $\epsilon/2$ would return a solution of cost at most $(1+\epsilon)$OPT as desired. Asymptotic approximation schemes are also described as *fully polynomial time* or *polynomial time* based on their running time's dependence on ϵ as described above. These are abbreviated as AFPTAS and APTAS respectively.

Approximation schemes are desirable as they can be tuned to approximate the problem within any required degree. They allow a tradeoff between running time and precision; by spending more computation time we get solutions which are closer to optimal. For example setting $\epsilon = .05$ an approximation scheme outputs solutions that are within 5% of the optimal.

Results and Organization. In this work, we design approximation schemes for the *Euclidean* UnitDem problem in small dimension and the 1-dimensional Unsplit problem. We study both the single depot and multiple depot settings. In summary our results include:

- A quasi polynomial time approximation scheme for Euclidean UnitDem VRP with single depot (Chapter 3). Our algorithm extends to the setting with multiple depots (Chapter 4).
- An asymptotic polynomial time approximation scheme (for a suitable definition of asymptotic) for 1-dimensional Unsplit VRP with a single depot (Chapter 5). Our algorithm extends when there are constant number of depots (Chapter 6).
- A polynomial time approximation scheme for 1-dimensional Unsplit VRP with restricted capacity and a single depot. (Chapter 7).

An understanding of our algorithm for UnitDem VRP requires an understanding of Arora's Euclidean TSP algorithm which we review in Chapter 2. Table 1.1 summarizes the existing algorithms for the UnitDem and Unsplit problems.

1.3 Techniques

1.3.1 Algorithm Techniques

We highlight some of the techniques used in our algorithms:

- *Simplifying the solution:* A common theme in all our algorithm is to first define some simplifying structure for solutions and then restrict ourselves to only search for these *structured* solutions. The simplicity of the structure implies that there cannot be too many different structured solutions and thus enumeration or dynamic programming can be used to find the structured solution with minimum cost. For example in the UnitDem problem a structured solution is one that uses portals, has few box crossings, and covers a predefined threshold number of customers. In the Unsplit problem it is a solution that covers customers from a small region. The main challenge is to identity simplifying structure that still allows for near optimal solutions.

- *Rounding:* We use rounding to simplify the input and also during computation to keep track of fewer possibilities. At times we round values to predetermined thresholds that are independent of the input. For example, in the UnitDem problem tour capacities are rounded to powers of $(1+\epsilon)$ (Chapters 3 and 4). We also round values to flexible thresholds extending a technique due to Fernandez de la Vega and Lueker [26]. In the Unsplit problem, we round customer demands to a constant number of thresholds, where the thresholds vary based on the demands that appear in the input (Chapters 5, 6, 7).

- *Arora's Techniques:* We make extensive use of the techniques from Arora's Euclidean TSP algorithm including the randomized dissection, the placement of portals and Arora's Structure Theorem. We use these for the UnitDem algorithms (Chapters 3 and 4).

1.3.2 Bounds for Analysis

To analyze our algorithms we like to compare the cost of its output with the cost of OPT. However as OPT is unknown instead we compare to a lower bound of OPT.

VRP Lower bounds. Some version of the following two lower bounds for UnitDem and Unsplit was proved by Haimovich and Rinnooy Kan [29] and by Haimovich, Rinnooy Kan and Stougie [30].

Definition 1.3.1. *(Rad) Let I be an instance of the UnitDem or Unsplit problem with customers C and depots D and vehicle capacity k. For each customer $i \in C$ let w_i denote the demand of i, where for UnitDem $w_i = 1$, $\forall i$. Let $r_i = \min_{d\in D} dist(i,d)$ denote the* radius *of i, where $dist(i,d)$ is the distance of customer i to depot d. The Rad of I is*

$$Rad(I) = \frac{2}{k}\sum_{i\in C} w_i \cdot r_i$$

Lemma 1.3.2. *(Rad Lower Bound.) Let I be an instance of the UnitDem or Unsplit problem, and let OPT denote the cost of the optimal solution of I. We have that,*

$$\mathit{Rad}(I) \leq OPT.$$

Proof. Let S be a solution of I, s be a tour of S and $i \in s$ denote that customer i is covered by tour s. Let d be the originating depot of tour s. The cost of s is at least the distance to the farthest customer from d covered by s, i.e $\mathrm{cost}(s) \geq 2\max_{i\in s}\mathrm{dist}(i,d)$. As $\mathrm{dist}(i,d)$ is at least r_i, we have:

$$\begin{aligned}\mathrm{cost}(S) &\geq \sum_{s\in S} 2\cdot \max_{i\in s} r_i \\ &= \sum_{s\in S} 2\cdot \max_{i\in s} r_i \cdot \sum_{i\in s}\frac{w_i}{\sum_{j\in s} w_j} \\ &\geq \sum_{s\in S}\sum_{i\in s} 2\cdot \max_{i\in s} r_i \cdot \frac{w_i}{k} \\ &\geq \sum_{s\in S}\sum_{i\in s} 2\cdot r_i \cdot \frac{w_i}{k} \quad = \quad \frac{2}{k}\sum_{i\in I} r_i\cdot w_i\end{aligned}$$

The second line holds as $\frac{\sum_{i\in s} w_i}{(\sum_{j\in s} w_j)} = 1$ and the third line as s covers total demand $\leq k$. □

Lemma 1.3.3. *(TSP Lower bound) Let I be an instance of the single depot UnitDem or Unsplit problem with customers C and depot d, and let OPT denote the optimal solution of I. The TSP of I i.e. the shortest length tour that visits all the customers and depot d, is such that $TSP \leq OPT$.*

Proof. Let S be a solution of I. Thus S visits all customers and the depot and therefore $cost(S) \geq$ TSP. S can be modified to visit each customer and the depot at most once by taking detours. The cost of the modified solution is still at most the cost S by the triangle inequality. □

The TSP is a lower bound in the single depot setting, where the solution is connected. To see why it may not be a lower bound in the multiple depot setting, consider an instance with two depots that are far from each other, but each is surrounded by a cloud of near by customers. A TSP of all the customers must traverse the distance between the depots, where as OPT may cost much less by consisting of tours that stay near one of two depots.

TSP bounds. The minimum spanning tree (MST) is a simple lower bound for the TSP.

Definition 1.3.4. *(Minimum spanning tree (MST)) Let $G = (V, E)$ be a connected graph with vertices V and edges E s.t each edge $e = (u, v) \in E$ has cost c_e equal to the distance between vertices u and v. A minimum spanning tree of G is a non-cyclic subgraph $T \subset E$ (i.e a tree) that connects all the vertices such that the sum of the cost of edges in T is minimum.*

Lemma 1.3.5. *(MST lower bound.) For a connected graph G, let TSP denote a tour of the vertices of G and MST denote the minimum spanning tree of G. We have that $MST \leq TSP$.*

Proof. MST is a lower bound for TSP as deleting any edge in the TSP results in a tree that connects all vertices (i.e a spanning tree). □

In our VRP algorithms we will often need to compute a TSP of a subset of the points, for which we will use a simple 2-approximation algorithm of TSP which is based on the MST. When distances satisfy the triangle inequality the output of the 2-approximation has cost at most 2MST, as shown by the following Lemma which appeared in [48].

Lemma 1.3.6. *(MST upper bound.) Let G, TSP and MST be defined as in the above Lemma, and let the cost of edges in G satisfy the triangle inequality. We have that $TSP \leq 2MST$.*

Proof. The following procedure starts from the MST and builds a TSP of cost at most 2MST:

1. Double each edge of the MST to obtain a Eulerian graph G' and find an Eulerian tour of G'.

2. Build a TSP by visiting the vertices in the order they appear in the Eulerian tour of G' shortcutting when necessary.

The Eulerian tour of G' has cost at most 2MST, and due to the triangle inequality the shortcutting steps do not add additional cost Thus TSP $\leq$ 2MST. $\square$

Chapter 2

Traveling Salesman Problem: Review of Arora's Algorithm [2]

2.1 Introduction

This chapter presents a review of Arora's PTAS for Euclidean TSP [2] which we will extend in Chapter 3 to design the QPTAS for the UnitDem problem.

Recall the problem: given a set of customers C represented as n points in $\mathcal{R}^d$, where d is a small constant independent of n, find the shortest length tour that visits all customers in C.

The traveling salesman problem is a fundamental and classical optimization problem. It is believed that some form of the problem was studied by mathematicians in the 1800s and that the familiar version was first studied around the 1930s. TSP is also a computationally difficult problem. Metric TSP is NP-Hard and does not admit a PTAS unless P=NP [5]. The best approximation known in the metric setting is the 3/2-approximation of Christofides which was discovered in 1977 [15]. Euclidean TSP was also shown to be NP-Hard in 1977 by Papadimitriou [42] and many researchers believed that the Euclidean problem also did not admit a PTAS. Thus it was a surprising discovery when Arora and independently Mitchell gave a PTAS for Euclidean TSP. Mitchell's algorithm works on the plane and Arora's algorithm works for constant number of dimensions [2, 40].

The techniques designed by Arora and Mitchell have since been applied to design algorithms for several other NP-Hard geometric problems, including approximation algorithms for TSP with neighborhoods [22], PTASs for Steiner Forest [12], k-Median [35], and k-MST [2, 40] and QPTASs for Minimum Weight Triangulation [46], Minimum Latency problems [4], and UnitDem of Chapter 3, among others. See [3] and [40] for surveys. The techniques of Arora and Mitchell had such great impact that both were awarded the 2010 Gödel Prize for their work on the Euclidean TSP problem. As the Gödel announcement stated *"The discovery of a PTAS for ETSP [Euclidean TSP], with its long trail of consequences, counts as a crowning achievement of geometric optimization"*.

In this chapter we present Arora's algorithm focusing mainly on the setting where the customers are located in $\mathcal{R}^2$. In Section 2.6 we describe his extension to $\mathcal{R}^d$ for constant d. The following theorem is proved,

Theorem 2.1.1. *(Main Theorem) [2] Algorithm 1 is a randomized polynomial time approximation scheme for the two dimensional Euclidean traveling salesman problem. Given $\epsilon > 0$, it outputs a solution with expected length $(1 + O(\epsilon))OPT$, in time $n \log^{O(1/\epsilon)} n$.*

Rao and Smith show how to improve Arora's running time to $O(n \log n + n2^{poly(1/\epsilon)})$. They build a *spanner* graph of the input point where all distances are within $(1 + \epsilon)$ factor of the Euclidean distances. Thus the optimal TSP on the spanner is a near optimal solution for the original input. Arora's techniques are employed to simplify the structure of the spanner which is then exploited to get a dynamic program that computes the TSP in faster time. See [3, 45] for more details.

Overview of Arora's approach. Arora's approximation scheme starts by considering the bounding box containing all the input points. A randomized dissection procedure is used to partition the bounding box recursively into four smaller boxes using one horizontal dissection line and one vertical dissection line, until the smallest boxes are of appropriate size. A small number of predetermined sites called *portals* are placed along the boundaries of all boxes. The algorithm concentrates on finding a TSP solution that is *portal respecting and light*, that is, a tour which enters and exits boxes only through portals and crosses each box at most a constant (i.e $O(1/\epsilon)$) number of times. A dynamic program is used to solve a subproblem in each smaller box and connect subproblems in adjacent boxes together to build the final solution.

Arora's structure theorem shows the existence of a near optimal TSP solution which has the portal respecting and light structure.

There are only a logarithmic number of portals on the boundary of each box and at most $O(n)$ non empty boxes in the dissection. Thus the dynamic program can "guess" the constant number of portals the tour will use in each box leading to the running time of Arora's algorithm.

2.2 Algorithm and Proof of Main Theorem

We review Arora's PTAS for 2-dimensional Euclidean TSP in Algorithm 1 and prove his main Theorem 2.1.1.

Algorithm 1 Arora's PTAS for 2-dimensional Euclidean TSP

```
Input: $n$ points $\in \mathbb{R}^2$
1: Perturb instance, perform random dissection and place portals as described in Section 2.2.1.
2: Find the the minimum length portal respecting light tour as defined in Definition 2.2.1 using
   the DP of Section 2.3.
Output: The tour found by the DP.
```

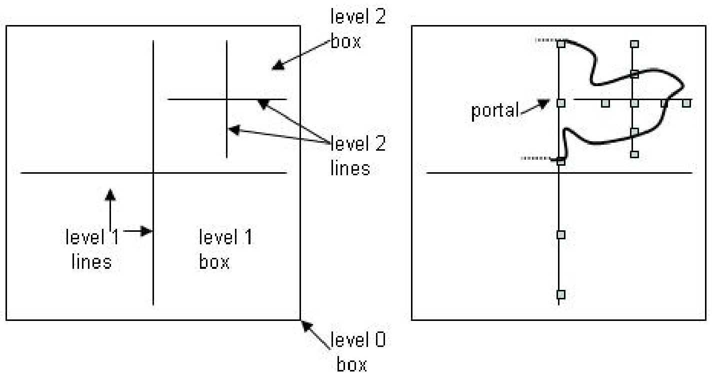

Figure 2.1: On the left a dissection showing the lines and boxes up to level 2. On the right portals on dissection lines and a portal respecting light tour.

2.2.1 Preprocessing

Preprocessing consists of perturbing the input, building a randomized dissection of the input space and placing portals on the dissection lines the tour will use to enter and leave the boxes of the dissection.

Perturbation. Define the *bounding box* as the smallest box whose side length L is a power of 2 that contains all input points and the depot. Let d denote the maximum distance between any two input points. Place a grid of granularity $\frac{d\epsilon}{n}$ inside the bounding box. Move every input point to the center of the grid box it lies in. Several points may map to the same grid box center, treat them as a single point [1]. Without loss of generality we can focus on solving the problem for the perturbed instance. A solution for the perturbed instance can be extended into a solution for the original instance by taking detours from the grid centers to the original locations of the points. The total cost of these detours is at most $n \cdot \frac{\sqrt{2}d\epsilon}{n} = \sqrt{2}d\epsilon$ and OPT $> 2d$, as it must visit the two farthest points. Thus the total cost of detours is at most ϵOPT which is within the required ϵ error parameter.

Finally scale distances by $\frac{4n}{\epsilon d}$ so that all coordinates become integral and the minimum distance between any two grid centers that contain points is least 4. After scaling the maximum distance between points and hence the side length of the bounding box is $L = O(n)$. Scaling does not change the structure of the optimal solution and we can always re-scale to get the cost of the original instance.

Randomized Dissection. Obtain a dissection by recursively partitioning the bounding box into 4 smaller boxes of equal size, using one horizontal and one vertical dissection line, until the smallest boxes are of size 1×1. The dissection can be viewed as a quad-tree with the bounding box as its root and the smallest boxes as the leaves. The bounding box has level 0, the 4 boxes created by the first dissection have level 1, the 16 boxes of the second dissection have level 2 and so on. Since $L = O(n)$ the level of the smallest boxes will be $\ell_{\max} = O(\log L) = O(\log n)$. See Figure 2.1. The

[1]In the UnitDem problem we will treat these as multiple points which are located in the same location.

horizontal and vertical dissection lines are also assigned levels. The boundary of the bounding box has level 0, the 2^{i-1} horizontal and 2^{i-1} vertical lines that form level i boxes by partitioning the level $i-1$ boxes are each assigned level i.

A randomized dissection of the bounding box is obtained by randomly choosing integers $a, b \in [0, L)$, and shifting the x coordinates of all horizontal dissection lines by a and all vertical dissection lines by b and reducing modulo L. For example the level 1 horizontal line moves from $L/2$ to $a + L/2 \mod L$ and the level 1 vertical line moves to $b + L/2 \mod L$. The dissection is "wrapped around" and wrapped around boxes are treated as one region. The crucial property is that the probability that a line l becomes a level ℓ dissection line in the randomized dissection is

$$Pr(\text{level}(l) = \ell) = \frac{2^\ell}{L} \tag{2.1}$$

Portals. Place *portals* along the dissection lines as follows. Let $m = O(\log L/\epsilon)$ and a power of 2. Place $2^\ell m$ portals equidistant apart on each level ℓ dissection line for all $\ell \leq \ell_{max}$. Since a level ℓ line forms the boundary of 2^ℓ level ℓ boxes there will be at most a $4m$ portals along the boundary of any dissection box b. As m and L are powers of 2, portals at lower level boxes will also be portals in higher level boxes. We will compute a tour that always enters and exits boxes at portals.

2.2.2 The Structure Theorem

Arora's structure theorem shows the existence of a near optimal TSP solution that is portal respecting and light as defined below. See Figure 2.1.

Definition 2.2.1. *(Portal respecting and light) Let π be a tour and D a random dissection of the input. A tour* crossing *is an edge of π that crosses a through a dissection line l of D. A tour is* portal respecting *if all crossings occur at portals. A tour is* light *with respect to D if for all dissection boxes b, it crosses each side of b at most $r = O(1/\epsilon)$ times.*

Theorem 2.2.2. *[2](Arora's structure theorem) For any instance I let π denote the optimal TSP solution of length OPT and let D be a randomized dissection of I. Given π, Algorithm 2 outputs a portal respecting light tour with respect to D, of expected length $(1 + O(\epsilon))OPT$.*

The proof of Arora's structure theorem is given in section 2.5.

2.2.3 Proof of the Main Theorem

Arora's Structure Theorem 2.2.2 implies the existence of a portal respecting and light solution with expected length $(1 + O(\epsilon))$OPT. The DP of Section 2.3 computes the portal respecting and light tour of minimum length, thus the DP solution has length at most $(1 + O(\epsilon))$OPT. The DP runs in time $n \cdot \log^{O(1/\epsilon)} n$.

2.3 The Dynamic Program

The DP Table. A *configuration* C of a dissection box b describes the tour segments that cross into b by a list of $O(r)$ portal pairs where each portal pair (p, q) indicates that a segment enters b at portal p and exits at portal q. As the tour is light there are at most r segments that cross into b.

The DP table has an entry for each dissection box b and each configuration C of b. Table entry $L_b[C]$ stores the minimum cost of placing tour segments in b such that they are compatible with C and cover all points inside b (where cost refers to length of the segments). The DP returns the minimum cost table entry of the root level box.

Computing the table entries. The table entries are computed in bottom-up order. The base case is to compute $L_b[C]$ for a leaf box b, As all points at are located in the center of a left box, $L_b(C)$ can be computed by trying the at most r possible ways to place the center into the $O(r)$ segments described by C.

Inductively, let b be a level ℓ box and let b_1, b_2, b_3, b_4 be the children of b at level $\ell + 1$. As the tour is portal respecting and light, the boundaries of b_1, b_2, b_3, b_4 inside b, are crossed at most $4r$ times by the tour and always at portals.

Let Λ be a list of at most $4r$ portal pairs where each pair represents a segment of b_1, b_2, b_3, b_4 and such that for each pair (p, p'), portals p, p' are from the boundaries of b_1, b_2, b_3, b_4 on the inside of b. An *interface vector* $\mathcal{P}$ describes how to partition the portal pairs of list Λ among the segments of configuration C. Fix a list Λ and suppose $\mathcal{P} = i_1 \geq i_2 \ldots \geq i_c$ and configuration $C = (p_1, q_1), (p_2, q_2), \ldots, (p_c, q_c)$. This represents that the first segment of C enters b using portal p_1, uses the segments of b_1, b_2, b_3, b_4 represented by portal pairs 1 through $i_i - 1$ in Λ, and exists b using portal q_1. The second segment of C enters b at p_2 uses the segments of b_1, b_2, b_3, b_4 represented by portal pairs i_1 through $i_2 - 1$, in Λ and exits b through q_2, and so on.

Let C_0 be a configuration for box b. The calculation of $L_b(C_0)$ is done in a brute force manner by iterating through all possible combinations of $\Lambda, \mathcal{P}$ and configurations of b's children, C_1, C_2, C_3, C_4. A combination $C_0, \Lambda, \mathcal{P}, C_1, C_2, C_3, C_4$ is *consistent* if gluing C_1, C_2, C_3, C_4 according to Λ and $\mathcal{P}$ yields configuration C_0. When b is the bounding box, only C_0 equal to the empty configuration is feasible as the solution must be contained inside the bounding box. Thus the segments inside the children of the bounding box must be concatenated together into one tour. In this case the order of the segments in Λ also describes how they should be concatenated together. Thus for a $\Lambda = (p_1, q'_1), \ldots, (p_l, q_l)$ to be feasible in the bounding box it must describe a tour by having $p_1 = q_l$.

The cost of configurations $\{C_i\}_{i \leq 4}$ is stored in the lookup table at $L_{b_i}(C_i)$, so the cost of $(C_1, C_2, C_3, C_4, \Lambda, \mathcal{P})$ is the sum of the costs of $L_{b_i}(C_i)$ for each child box $i \leq 4$. Entry $L_b(C_0)$ is the cost of the tuple $(C_1, C_2, C_3, C_4, \Lambda, \mathcal{P})$ which is consistent with C_0 and has minimum cost.

Running time of dynamic program. A configuration is a list of $O(r) = O(1/\epsilon)$ portal pairs. As each box has $O(m) = O(\log L/\epsilon)$ portals there are $O(m^2) = O((\log L/\epsilon)^2)$ possible portals pairs and each box b has $O((\log L/\epsilon)^{O(1/\epsilon)})$ possible configurations. As there are $O(\log L)$ levels and $O(n)$ non-empty boxes, the DP table has overall size $n \cdot (\log L/\epsilon)^{O(1/\epsilon)}$ which is $n \cdot \log^{O(1/\epsilon)} n$, as $L = O(n)$.

The boundaries of child boxes b_1, b_2, b_3, b_4 have a total of $O(\log L/\epsilon)$ portals. By similar reasoning as above there are $\log^{O(1/\epsilon)} n$ possible choices of Λ. $\mathcal{P}$ is a list of $O(r)$ integers that are increasing and all between 0 and $O(r)$. For a fixed Λ there are $r^{O(r)} = (1/\epsilon)^{O(1/\epsilon)}$ ways to choose $\mathcal{P}$, thus there are $\log^{O(1/\epsilon)} n$ choices of $\Lambda, \mathcal{P}$.

Checking consistency for a particular choice of $\Lambda, \mathcal{P}$ and configurations $\{C_i\}_{i\leq 4}$ can be done in polynomial time in the size of these descriptions. To compute the lookup table entry at $L_b[C_0]$ by running through all combinations, takes time polynomial in $n \log^{O(1/\epsilon)} n$.

2.4 Computing a Structured Tour

Arora's structure theorem uses Algorithm 2 to make a tour portal respecting and light.

Algorithm 2 Make-Structured

```
. Input: Tour $\pi$, Random dissection $D$
1:  Run Bottom-Up-Patching, Alg. 4, on each vertical line $l$ in $D$ in arbitrary order.
2:  Run Bottom-up-Patching, Alg. 4, on each horizontal line $l'$ in $D$ arbitrary order, ignoring new
    crossings created in previous step.
3:  Run Make-Portal-Respecting, Alg. 5, ignoring new crossings created in previous steps.
4:  for each dissection box corner $c$ lying at the intersection of some vertical line $l$ and horizontal
    line $l'$ do
5:      if there are more than two horizontal crossings of $l'$ at $c$ then
6:          Do Patching, Alg. 3, at point $c$ on line $l'$ left of $l$.
7:          Do Patching, Alg. 3, at point $c$ on line $l'$ right of $l$.
8:      end if
9:      if there are more than two vertical crossings of $l$ at $c$ then
10:         Do Patching, Alg. 3, at point $c$ on line $l$ above $l'$.
11:         Do Patching, Alg. 3, at point $c$ on line $l$ below $l'$.
12:     end if
13: end for
Output: A portal-respecting, light tour $\pi$ with respect to $D$.
```

2.4.1 Patching

Patching ensures that the boundary of each dissection box has at most $r = O(1/\epsilon)$ crossings. Given a line segment s with many tour crossings, Arora's patching procedure augments the tour with copies of s. To consolidate crossings, the tour is modified to use the copies of s to stay on one side of s as long as possible before crossing to the other side. See Figure 2.2 for an example of patching. Lemma 2.4.1 bounds the cost of Patching.

Lemma 2.4.1. *(Patching) Let s be a line segment and π any closed path that crosses s at least thrice. Algorithm 3 augments π with segments of s of total length at most $3length(s)$ such that π is modified to a closed path π' that crosses s at most twice.*

Proof. As $M'_1, \ldots, M'_t$ all lie on segment s, the edges in the minimum cost perfect matching and the edges in the shortest path connecting $\{M_i\}_{i\leq t}$ all lie on segment s. Thus all segments in J' on

Algorithm 3 Patching

```
. Input: Segment $s$ of line $l$ containing $t \geq 3$ crossings of tour $\pi$
1: Break $\pi$ at points $M_1, M_2, \ldots, M_t$, where $\pi$ crosses $s$, to obtain paths $P_0, P_1, P_2, \ldots, P_t$.
2: Make two copies of each $M_i$ denoted $M_i'$ and $M_i''$ and place a copy on either side of $l$.
3: Let $J'$ be a multiset of segments containing (1) the shortest path connecting $M_1'$ and $M_t'$ and
   (2) the segments in the minimum cost perfect matching between $M_2', \ldots M_t'$. Define $J''$ similarly
   for $M_1'' \ldots M_t''$.
4: Let $G$ be a graph with vertices $\{M_i', M_i''\}_{i \leq t}$ and edges $\{P_i\}_{i \leq t}$, $J'$ and $J''$.
5: if  $t$ is odd  then
6:     Add an edge between $M_t'$ and $M_t''$ in $G$
7: else
8:     Add two edges between $M_t'$ and $M_t''$ in $G$
9: end if
10: Let $\pi'$ by an Eulerian traversal of $G$
Output: Return tour $\pi'$ which crosses segment $s$ at most twice.
```

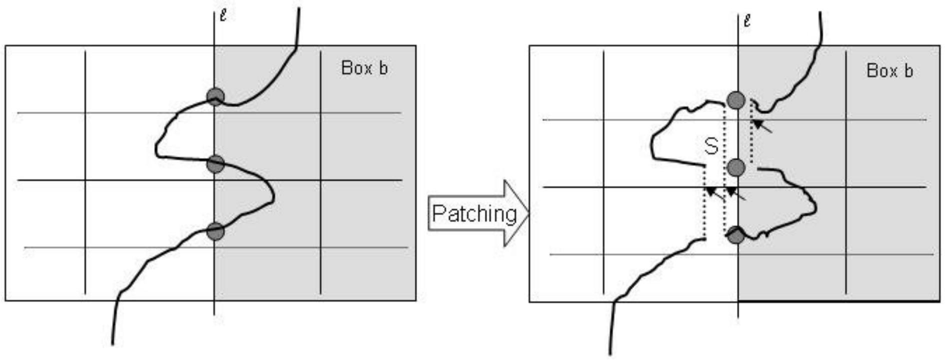

Figure 2.2: Box b is the shaded region on the right. Patching is done on line l at boundary of box b. This reduces the three crossings on the boundary of box b (gray circles) to just one crossing (the bottom crossing). Patching augments the tour with copies of S (dotted) on both sides of l, which introduces new crossings on the horizontal lines marked by arrows.

segment s. The shortest path has cost at most length(s) and minimum cost perfect matching has cost at most length(s)/2, thus the total length of segments in J' is at most 3length(s)/2. The same argument shows that the total length of segments in J'' is at most 3length(s)/2. Since M'_t and M''_t are copies of the same point, the edge between them which is added to G in the *if* statement has zero length. Thus the total length of segments in G is the sum of the lengths of $P_1, P_2, \dots, P_t$, plus the length of segments in J', J'' which is at most length(π) + 3length(s).

Every node in graph G has even degree thus the Eulerian traversal π' includes all its edges. Thus π' visits all edges $P_1 \dots, P_t$ along with all edges in J' and J'' and crosses s twice if t is even and once if t is odd. The additional length of π' over π is 3length(s). □

The Bottom-up-Patching procedure, Algorithm 4, is applied to every dissection line, to make the tour light. To reduce crossings the procedure applies patching bottom up on the box boundaries lying on line l. Lemma 2.4.2 bounds the cost of bottom up patching on line l in terms of the number of crossings on line l.

Algorithm 4 Bottom-up-patching

```
Input: line $l$ with an arbitrary number of tour crossings
1: for $j = \log L$ down to level($l$) do
2:   for each level $j$ dissection box $b$ whose boundary lies on line $l$ do
3:     if segment $l \cap b$ has more than $r - 4$ crossings then
4:       Do patching Algorithm 3 on segment $l \cap b$ to reduce the crossings on $l \cap b$ to at most 2.
         (or at most 4 for wrapped around boxes) (Figure 2.2).
5:     end if
6:   end for
7: end for
Output: The boundary of each box lying on line $l$ has at most $r$ tour crossings.
```

Lemma 2.4.2. *Let π be the salesman tour, l be any dissection line and $t(\pi, l)$ denote the number of times π crosses line l. The expected cost to make tour π light at line l using Algorithm 4 is $O(\epsilon)t(\pi, l)$.*

Proof. Suppose that level(l) = i. For each $j \geq i$ let $p_{l,j}$ denote the number of times patching is applied in the j-th iteration of the for loop of Algorithm 4. Each application of patching replaces at least $r - 4 + 1$ original crossings in l by at most 4 crossings. Thus we have

$$\sum_{j \geq i} p_{l,j} \leq \frac{t(\pi, l)}{r - 3} \tag{2.2}$$

As each level j box has side length $L/2^j$, a patching on line l in the j-th iteration adds length $3 \cdot L/2^j$ to π by Lemma 2.4.1. Thus to make π light at line l its length will increase by at most

$$\text{increase in lenth of } \pi \text{ if (level}(l) = \text{i)} \quad \leq \quad \sum_{j \geq i} p_{l,j} \cdot 3 \cdot \frac{L}{2^j} \tag{2.3}$$

If l is a vertical line its level depends only on the horizontal shift a and if l is a horizontal line its level depends on vertical shift b. In either case the level(l) = i with probability $2^i/L$ by Equation

2.1. Thus the expected increase in length of π over the choice of the horizontal random shift is.

$$
\begin{aligned}
&E(\text{ increase in length of } \pi \text{ for line } l) \\
&= \sum_{i\geq 1} \Pr[\text{level}(l) = i] \cdot (\text{ increase in length of } \pi \text{ if level}(l) = \text{i }) \\
&\leq \sum_{i\geq 1} \frac{2^i}{L} \cdot \sum_{j\geq i} p_{l,j} \cdot 3 \cdot \frac{L}{2^j} \\
&= 3 \sum_{j\geq 1} \frac{p_{l,j}}{2^j} \cdot \sum_{i\leq j} 2^i \\
&\leq 3 \sum_{j\geq 1} 2p_{l,j} \\
&\leq \frac{6t(\pi,l)}{r-3} = O(\epsilon)t(\pi,l) \qquad (2.4)
\end{aligned}
$$

where Equation 2.4 follows from Equation 2.2 and as $r = O(1/\epsilon)$. □

2.4.2 Portal Respecting

To make the tour portal respecting the Make-Portal-Respecting procedure (Algorithm 5) replaces each non-portal crossing with a detour to its nearest portal. Let x be a non-portal crossing on line l and s the segment of l between x and the closest portal l. The procedure adds a copy of s on either side of s and takes detours along these copies as shown in Figure 2.3. Lemma 2.4.3 bounds the cost of the procedure.

Algorithm 5 Make-Portal-Respecting

```
Input: Tour $\pi$ with a set $X$ of non-portal crossings
1: for each crossing $x \in X$ do
2:     Let $l$ be the line crossed by $x$, $p$ the portal on $l$ closest to $x$, and $s$ the segment of $l$ between
       $x$ and $p$
3:     Let $s', s''$ be two copies of $s$ one for either side of $l$.
4:     Augment $\pi$ with $s', s''$. Use copy of $s'$ to go from $x$ to $p$, cross at $p$, and then use copy $s''$ to
       go from $p$ to $x$. (See Figure 2.3).
5: end for
Output: $\pi$, augmented into a portal respecting tour.
```

Lemma 2.4.3. *Let π be a tour and let $t(\pi, l)$ denote the number of times π crosses a dissection line l. The expected cost to make π portal respecting (with respect to all lines) using Algorithm 5 is $O(\epsilon)\sum_{\text{line } l} t(\pi, l)$.*

Proof. We show that for any dissection line l the expected cost to make π portal respecting at a line l is $O(\epsilon)t(\pi, l)$. The proof for all lines follows by linearity of expectation.

Consider a line l and suppose that level$(l) = i$. Recall that a level i line has $2^i m$ portals equidistance apart, where $m = O(\log L/\epsilon)$. Thus for any crossing x on l, Algorithm 5 augments π

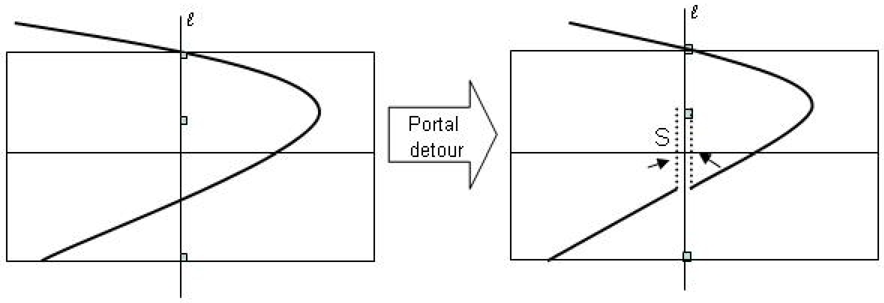

Figure 2.3: Portals on line l are shown as small boxes. The middle crossing is made portal respecting by augmenting the tour with copies of segment S (dotted) on both sides of l, which introduces new crossings on the horizontal line at positions marked by the arrows.

with s' and s'' which have total length at most $\frac{L}{2^i m}$. There are at most $t(\pi, l)$ non-portal crossing on line l thus the total cost for line l is

$$\text{increase in length of } \pi \text{ if } (\text{level}(l) = i) \quad \leq \quad \frac{L}{2^i m} \cdot t(\pi, l)$$

Given that the probability that level$(l) = i$ is $2^i/L$ by Equation 2.1, and that $\ell_{\max} = O(\log L)$, we have that the expected cost to make π portal respecting at line l is

$$\begin{aligned}
&E(\text{ increase in length of } \pi \text{ for line } l) \\
&= \sum_{i=1}^{\ell_{\max}} Pr[\text{level}(l) = i] \cdot (\text{increase in length of } \pi \text{ if level}(l) = i) \\
&\leq \sum_{i=1}^{\ell_{\max}} \frac{2^i}{L} \cdot \frac{L}{2^i m} \cdot t(\pi, l) \\
&= \frac{\ell_{\max}}{m} = O(\epsilon) t(\pi, l)
\end{aligned}$$

□

2.5 Proof of the Structure Theorem

To prove Arora's Structure Theorem 2.2.2 we will prove the correctness of Algorithm 2, that given any input tours π, Algorithm 2 outputs a portal respecting light tour of cost at most $(1+O(\epsilon))$length(π). We start by demonstating that an intuitive approach to making a tour portal respecting and light does not work due to the new crossings added during patching and portal detours. Then we list a few properties of these new crossings and finally in subsection 2.5.3 we prove the correctness of Algorithm 2.

2.5.1 A First Approach

Arora uses the following simple property to relate the cost of the tour to the number of times it crosses the grid lines of the dissection. For any tour π and dissection line l let $t(\pi, l)$ denote the number of crossings of π with l.

Property 2.5.1. *For any tour π, length$(\pi) \geq \frac{1}{2}\sum_{line\ l} t(\pi, l)$.*

Proof. Consider an edge in π as a portion of the tour that goes between two locations with points. We show that each edge contributes at most twice its length to the left side of the equation, so summing over all edges proves the lemma.

Consider an edge e in π that goes between two locations with points and has length s. Let u, v be the horizontal and vertical projections of the edge, such that $u^2 + v^2 = s^2$. Then e contributes at most $u + 1$ plus $v + 1$ to the left hand side and with a bit of calculation it is easy to see that

$$u + v + 2 \leq \sqrt{2(u^2 + v^2)} + 2 \leq \sqrt{2s^2} + 2$$

Since the minimum distance between perturbed points is at least 4, $\sqrt{2s^2} + 2 \leq 2s$. $\square$

One seemingly correct method to make a tour π portal respecting and light tour is to apply Bottom-up-Patching, Algorithm 4, on each dissection line and then apply Make-Portal-Respecting, Algorithm 5. By Lemma 2.4.2 and using linearity of expectation the total expected cost of doing Bottom-up-Patching on all lines is $O(\epsilon)\sum_{\text{line } l} t(\pi, l)$ and by Lemma 2.4.3 Make-Portal-Respecting has expected cost $O(\epsilon)\sum_{\text{line } l}$. Thus the total expected cost to make π portal respecting and light would be $O(\epsilon)\sum_{\text{line } l} t(\pi, l)$, which by Property 2.5.1 is negligible compared to its length, i.e $O(\epsilon)\sum_{\text{line}} t(\pi, l) = O(\epsilon)\text{length}(\pi)$.

However as shown in Figures 2.2 and 2.3, the method described above is incorrect as it does not address the fact that patching and taking portal detours on a vertical line l can add new crossings on to horizontal lines. Both patching and portal detours on line l augments the tour with a vertical segment s, and adds new crossings on horizontal lines that intersect with s. Similarly patching and adding portal detours on a horizontal line l' can add new crossings to vertical lines. As Arora states:

> *"whenever we .... augment the salesman path with some segments lying on (a vertical line) l, these segments could cause the path to cross some horizontal line l' much more than the $t(\pi, l')$ times it was crossing earlier. "* [2]

These new crossings seem problematic to making the tour light as we may need to patch over and over again in different directions: patch on vertical lines l to reduce the new crossings from horizontal patching, then patch again on horizontal lines l' to reduce the new crossings from vertical patchings, and so on. It is not clear how to show that this process terminates or how to bound the total cost incurred. Arora initially gives his analysis ignoring these new crossings and then provides a brief explanation of how to deal with them so that the tour remains light without incurring additional costs. As Arora states:

> *"The patching on a vertical line ... could increase the number of times the path crosses a horizontal line. We ignore this effect for now, and explain at the end of the proof that this costs us only an additional 4 crossings."* [2]

This issue is averted in Algorithm 2 as it ignores all new crossings while it runs Bottom-Up-Patching and the Make-Portal-Respecting procedures. This is presumably what Arora meant by *" ignore this effect for now"* in his second quote above. In the end Algorithm 2 does zero-cost patchings to ensure that at most an additional 4 crossings are incurred as Arora claims above. See subsection 2.5.3 for more details.

2.5.2 Properties of New Crossings

We list some properties of new crossings that will help us prove the correctness of Algorithm 2.

A *new crossing* of π is a crossing that appears from augmenting π during patching and portal detours. A crossing is *horizontal* if it crosses only a horizontal lines l' and it is *vertical* if it crosses only a vertical line l. If a crossing crosses vertical and horizontal lines, i.e it crosses on a box corner, then it a *multi*-dimensional crossing. See Figure 2.4.

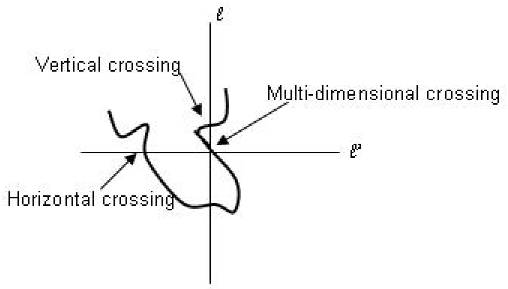

Figure 2.4: The vertical crossing crosses line l, the horizontal one crosses l' and the multidimensional one crosses both and l and l'.

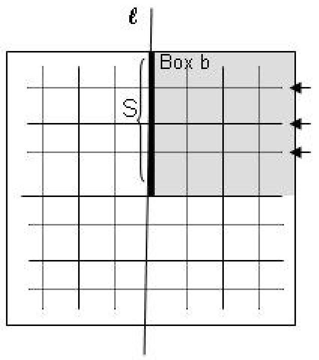

Figure 2.5: Patching on vertical line l in box b augments tour with copies of segment S (dark line), adding new horizontal crossings to the vertical lines marked by the arrows.

The following properties hold for new crossings.

Property 2.5.2. *Each patching on a vertical line l augments the tour with copies of some segment s of l, and adds new horizontal crossings to horizontal lines l' that intersect s. Each such l' has* *level(l') > level(l) and gets at most 2 new horizontal crossings to the left of l and 2 new horizontal crossings to the right of l.*

Proof. A patching on line l augments the tour with copies of segment $s = l \cap b$ (See Line 4 of Algorithm 4) and new crossings are created where s intersects with a horizontal line l'. See Figure 2.5. All horizontal lines l' that intersect s inside the box have level(l') > level(l), since the lines forming the boundary of a box have lower levels than the lines in the box. As Algorithm 3 adds as most 2 copies of s on either side of l, each patching adds at most 4 new horizontal crossings on l'; 2 to the left of l and 2 to the right of l. □

Property 2.5.3. *Each portal detour on a vertical line l augments the tour with copies of a segment s of l and adds new horizontal crossings to horizontal line l' that intersect s. Each such l' has level(l') > level(l) and gets at most one new horizontal crossing to the left of l and one new horizontal crossing to the right of l.*

Proof. A portal detour on l augments the tour by a segment s of l creating new crossings whereever s intersects with horizontal lines l'. As step 3 of Algorithm 5 adds one copy of s on both sides of l, this adds at most one new horizontal crossing on l' to the right of l and 1 new crossing on l' to the left of l. Now we show that new crossings from the portal detour appear on lines l' with level(l') > level(l). As l has at least $2^{level(l)}$ portals placed equidistance apart, length(s) is at most $L/2^{level(l)}$. This is also the side length of the largest dissection box whose boundary lies on line l. The corners of the largest dissection boxes on l are portals by definition, thus s always reaches a portal on l before crossing a horizontal line at level $\leq$ level(l). □

Analogous properties hold for patching and portal detours on a horizontal line.

Property 2.5.4. *Each patching on a horizontal line l' augments the tour with copies of some segment s of l' and adds new vertical crossings to vertical line l that intersect s. Each such l has level(l) > level(l') and gets at most 2 new vertical crossings above l' and 2 new vertical crossings below l'.*

Property 2.5.5. *Each portal detour on a horizontal line l' augments the tour with copies of a segment s of l adding new vertical crossings to each vertical line l that intersects s. Each such l has level(l) > level(l') and gets at most one new vertical crossing above l and one new vertical crossings below l.*

A crossing is counted as a crossing for box b if it crosses one of the boundaries of b.

Property 2.5.6. *For any box b, all the new crossing for b occur at the corners of b.*

Proof. See Figure 2.6. Let line l contain the boundary of box b. Assume for a contradiction that a non-corner point i on the boundary of b on l contains a new crossing. Properties 2.5.2, 2.5.3 and 2.5.4, 2.5.5 show that all new crossings occur at the intersection of two lines so let g denote the grid line intersecting with l at the point i. As i is a non-corner point on boundary of b, level(g) > level(l).

Without loss of generality assume l is a vertical line so g is horizontal. Since the crossing at i is counted as a crossing for b it crosses the boundary on line l and thus is a vertical crossing. By Properties 2.5.2 and 2.5.3 patching and portal detours on a vertical line can create only horizontal crossings so the new crossing at i must have occurred from a patching or a portal detour on the horizontal line g. By Properties 2.5.4 and 2.5.5 patching and portal detours on g creates new crossings only on lines at levels $>$ level(g). We have reached a contradiction as level(l) $< level(g)$. Thus there is no way for a new crossing of b to exist at non-corner point i $\square$

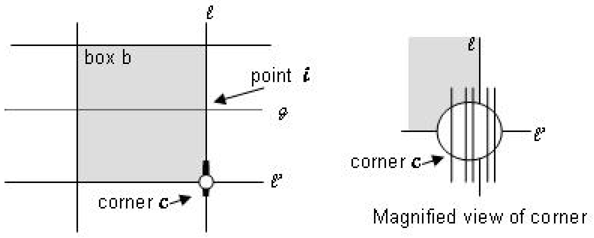

Figure 2.6: New crossings shown as dark segment at corner c. New crossings appear only at the corner of b and never at any intermediate boundary site on b such as i. The right figure shows a magnified view of corner crossings. The three left crossings count as crossings of b.

2.5.3 Correctness of Algorithm 2

We show that given any tour π, Algorithm 2 outputs a portal respecting and light tour of length at most $(1+O(\epsilon))$length(π). Steps (1-3) ignores all new crossings thus after these steps π is portal respecting and light with respect to the *original* crossings. By Lemma 2.4.2 and using linearity of expectation the total expected cost of doing Bottom-up-Patching on all lines is $O(\epsilon)\sum_{\text{ line } l} t(\pi, l)$ and by Lemma 2.4.3 Make-Portal-Respecting has expected cost $O(\epsilon)\sum_{\text{ line } l} t(\pi, l)$. Thus the total expected cost incurred up to step 3 is $O(\epsilon)\sum_{\text{ line } l} t(\pi, l)$, which by Property 2.5.1 is negligible compared to its length, i.e $O(\epsilon)\sum_{\text{ line }} t(\pi, l) = O(\epsilon)$length$(\pi)$.

By Property 2.5.6, for any dissection box, the new crossings introduced in steps (1-3) appear only at the corners of the box. As all corners are already portals by definition, all new crossings are already portal respecting. It remains to show that the tour can be made light. Algorithm 4 ensures that each box boundary has at most $r-4$ original crossings. We show that steps 4-11 of Algorithm 2 ensures that each dissection box corner will contain at most 2 new crossings. Thus the boundary of a dissection box will have at most $r = O(1/\epsilon)$ total crossings and the tour will be light.

Suppose box b has a corner c at the intersection of line l and l' with more than two horizontal crossings. See right side of Figure 2.6. We show that steps (5-8) of Algorithm 2 reduces them to at most 4 horizontal crossings at zero cost and without introducing any new vertical crossings. An analogous argument shows the same for steps (9-12) and the vertical crossings on any corner.

If c has more than two horizontal crossings at c, steps (6-7) does patching at point c on line l'. The patching occurs at a single point c, thus the tour is augmented by a "segment" s, on l'

containing only point c. This implies that s has zero length. Thus by Lemma 2.4.1, as length$(s) = 0$ the patching has zero cost. The patching is done separately on the left and right sides of l so it does not introduce any new vertical crossings on line l. At the end of Step 7, c will contain at most 2 horizontal crossings on the right of l and at most 2 horizontal crossings on the left of l. Since b is only on one side of l, (b is on the left side of l in Figure 2.6) b now has at most 2 horizontal crossings at c.

In summary up to step 3, Algorithm 2 adds additional length $O(\epsilon)$length(π) to π to convert it into a portal respecting light tour with respect to the original crossings. The remaining steps add no additional length and convert it into a portal respecting light tour with respect to all crossings. Thus the final tour has length at most $(1 + O(\epsilon))$length(π).

2.6 Extension to Higher Dimensions

Arora's algorithm extends to higher dimensions $d \geq 2$ while d is a constant independent of n. In $\mathcal{R}^d$ the dissection can be thought of as a $\mathcal{R}^d-$ary tree. Each dissection box is now a d dimensional cube with $2d$ boundaries (facets) and each boundary is a $d-1$ dimensional cube. The boundary of a box contains $m = O(\sqrt{d} \log L/\epsilon)^{d-1}$ portals which are placed as an orthogonal lattice of m-points. A tour is *light* if for each region, the tour crosses the boundary of the region at most $r = O(\sqrt{d}/\epsilon)^{d-1}$ times. Comparing to the values of m and r used in $\mathcal{R}^2$, $O(\log L/\epsilon)$ and $O(1/\epsilon)$, we see that the blowup in $\mathcal{R}^d$ comes mainly from placing the portals in a lattice structure in $d-1$ dimensions [2].

The running time of Arora's algorithm in $\mathcal{R}^d$ is $O(2^d n \log L \cdot m^{O(2dr)}) = n(\log n)^{O(d/\epsilon)^{d-1}}$, as $L = O(n)$. This running time follows as the dissection tree now contains $O(2^d n)$ non empty boxes and as the DP has to "guess" r of the m portals on each of the $2d$ boundaries that portal respecting and light tour tour will use to enter and exit each non empty box. Note that the running time has a doubly exponential dependence on d. Thus the dimensions should be $d = o(\log \log n)$ for the running time to be polynomial.

Chapter 3

Vehicle Routing: Unit Demands and Single Depot

This chapter's results are joint work with Claire Mathieu and have appeared in [21].

3.1 Introduction

This chapter present our QPTAS for Euclidean UnitDem problem with a single depot, in constant dimensions.

Recall the problem: Given a positive integer k denoting the vehicle capacity, a set C of n customers, and a depot, such that C and the depot are points in $\mathcal{R}^d$ where d is a small constant independent of n, find a collection of tours of minimum total length covering all customers in C, such that each tour in the collection starts and ends at the depot and covers at most k customers.

As previously mentioned, the UnitDem problem models scenarios where all customers either have equal demands for items or when customer demands are allowed to be split between multiple tours.

The metric setting of UnitDem has a 2.5-approximation due to Haimovich and Rinnooy Kan [29], but is APX-Hard and admits no PTAS [6]. The Euclidean setting has been conjectured, by Arora and others, to have a PTAS [3]. The conjecture seems natural as UnitDem is the most basic vehicle routing problem and seems very close to the TSP, which admits a PTAS in the Euclidean setting [2, 40]. Partial results already exist and PTASs have been designed for Euclidean UnitDem for certain vehicle capacities. When the vehicle capacity is large, $k = \Omega(n)$, Arora's PTAS for TSP easily extends to a PTAS. In the case of small capacity, $k = O(\log n / \log\log n)$, Asano et al. [7] presented a PTAS extending a known PTAS due to [29]. Recently, for larger capacity, $k \leq 2^{\log^{\delta} n}$ (where δ a function of ϵ), Adamaszek et al. presented a PTAS using the QPTAS described in this chapter as a black box [1]. Designing a PTAS regardless of the value of k remains an active line of research and as noted by Adamaszek et al. *"the case $k = \sqrt{n}$ is the hardcore of the difficulty in obtaining a PTAS for all values of k* [1].

We present a QPTAS for Euclidean UnitDem VRP with single depot. We focus mainly on the setting where the customers are located in $\mathcal{R}^2$ and in Section 3.7 show how to extend the results to $\mathcal{R}^d$ for constant d. We prove the following Theorem.

Theorem 3.1.1. *(Main Theorem) Algorithm 6 is a randomized quasi-polynomial time approximation scheme for the two dimensional Euclidean unit demand vehicle routing problem with a single depot. Given $\epsilon > 0$, it outputs a solution with expected length $(1 + O(\epsilon))OPT$, in time $n^{\log^{O(1/\epsilon)} n}$. The Algorithm can be derandomized.*

Where previous approaches fail. It is natural to start from Arora's PTAS for Euclidean TSP and try to extend it to UnitDem. In fact Arora attempted this as well. However this approach reaches a road block which Arora described in [3] and we now recap below.

Using Arora's randomized dissection we can recursively partition the region of input points into progressively smaller boxes. Then we can search for a solution that goes back and forth between adjacent boxes a limited number of times and always through a small number of predetermined sites, i.e *portals*, that are placed along the boundary of boxes. The next step is to extend the TSP structure theorem to show that there exists a near optimal solution that crosses the boundary of boxes a small number of times, so that dynamic programming can be used to guess these crossings. Unfortunately, as noted by Arora [3],

> *"we seem to need a result stating that there is a near-optimum solution which enters or leaves each area a small number of times. This does not appear to be true. [...] The difficulty lies in deciding upon a small interface between adjacent boxes, since a large number of tours may cross the edge between them. It seems that the interface has to specify something about each of them, which uses up too many bits."*

Indeed, to combine solutions in adjacent boxes it seems necessary to remember the number of points covered by each tour segment and that is too much information to remember, if there are many tour segments.

Overview of our approach. To get around the problem above we introduce a new trick which allows us to remember *approximately* how many points are on each tour segment. This allows us to build tours that cover *approximately* k points. We design a simple randomized technique to drop points from these tours so that the capacity constaint is maintained. Our technique ensures that the dropped points can be covered at low cost with additional tours, and we simply use the 3-approximation of [29] to cover them. See Figure 3.1. Thanks to dropping points, we may assume that the number of points on each tour segment is a power of $(1 + \epsilon/\log n)$, so there are only $O(\log n \log k)$ possibilities. This is a huge saving (when k is $\Omega(\log n)$) compared to the k possibilities that would be required to remember the number of points exactly and it enables us to deal with the difficulties described by Arora: now we have a small interface between adjacent boxes, namely, for every pair of portals and every threshold number of points, we remember the number of tour segments that have this profile. The quasipolynomial running time of our dynamic program (DP) follows as the number of profiles is polylogarithmic and there are at most n tour segments of each profile.

The main technical difficulty consists in showing that the dropped points can be covered at low cost. That cost is split into several components and analyzed separately using a variety of techniques in Section 3.5.

3.2 Algorithm and Proof of Main Theorem

Algorithm 6 presents our QPTAS. It starts the same way as Arora's TSP algorithm by performing a random dissection of the input and placing portals on girdlines. Then it uses a quasipolynomial time DP to find a *structured* solution (Definition 3.2.4), which is near optimal but can include some tours that cover more than k points. A feasible set of tours, which we refer to as the *black* tours, is obtained by dropping points from the tours of the structured solution that cover more than k points. The dropped points which we refer to as the *red* points, are chosen carefully using a randomized procedure to ensure that they can be covered cheaply. We use the 3-approximation of [29] on the red points to obtain a set of red tours. The final output is the union of the red and black tours. See Figure 3.1.

Algorithm 6 QPTAS for Euclidean UnitDem single depot, in two dimensions

```
Input: Customers $C$, a depot in $\mathbb{R}^2$, and integer $k$
1: Perturb instance, perform random dissection, and place portals as in Arora's algorithm of Chap. 2.2.1.
2: Use the DP from Section 3.4 to find a structured solution of Definition 3.2.4.
3: Use the DP's history to construct the structured tours and assign types to points using the randomized type assignment from Subsection 3.2.2.
4: Color a point *black* if it has type 0 and *red* otherwise. Drop all red points from the structured tours, so that the structured tours only cover black points.
5: Use the 3-approximation Algorithm 8 to get solution of the dropped red points.
Output: Union of red tours on the red points and black tours on the black points.
```

In the following subsections we define a *structured* solution, the random type assignment procedure and describe the constant factor approximation. We also state some auxiliary theorems and use them to prove the main theorem (Theorem 3.1.1). The proofs of the auxiliary theorems are presented in later sections.

3.2.1 The Structure Theorem

The structured solution will be allowed to consist of tours that cover approximately k points. To compute the structured solution in quasi-polynomial time we will predefine $\tau = O(\log L \log k)$ *thresholds* in the range $[1, k]$ (recall that $L = O(n)$ is the side length of the bounding box). Rather than remembering the exact number of points covered by a tour segment we will remember a threshold. If a tour segment covers x points we will "round" it to cover a threshold number of points as follows:

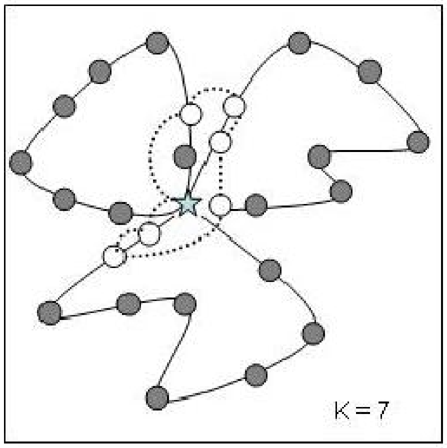

Figure 3.1: A solution computed by Algorithm 6 for $k = 7$. The star is the depot, the solid circles are the "black" points and the empty circles are the "red" points. The solid tours are computed by the DP in step 2. Each covers $\leq k$ "black" points. The dotted tour covers the "red" points and is computed in step 5 using the 3-approximation.

We find the largest threshold value $t < x$ and set the *type* of exactly $x - t$ points on the segment to indicate that they should be dropped from the segment. Once dropped the segment will contain exactly the threshold number of points.

Tour segments can be rounded at any level of the dissection tree. To indicate that a point was rounded/dropped at level ℓ we will set its type to ℓ. In the end, all points with type between $[1, \ell_{\max}]$ will be colored red and dropped from the structured tours. We defined these concepts formally:

Definition 3.2.1. *(Thresholds, rounded segments, rounded box)*

- *Let* $\tau = \lceil \log_{(1+\epsilon/\log L)}(k\epsilon) \rceil + 1/\epsilon$ *be the number of thresholds, where* $L = O(n)$ *the side length of the bounding box. For* $i \in [1, 1/\epsilon]$*, let threshold* $t_i = i$*, and for* $i \in (1/\epsilon, \tau]$ *let threshold* $t_i = t_{i-1}(1 + \epsilon/\log L)$*.*

- *The* type *of a point is an integer in* $[0, \ell_{\max}]$*. A point of type*$= 0$ *is* active *in all levels and a point of type*> 0 *is active in all levels greater than its type.*

- *For any box* b*, a* segment *of* b *is a piece of a tour that enters and exits* b *at most once. A segment of box* b *is* rounded *if it covers exactly a threshold* t_i *number of points inside* b*, that are active at level*(b)*. Otherwise the segment is* unrounded*. See Figure 3.2.*

- *A box is a* rounded *if it contains only rounded segments. Otherwise it is* unrounded*.*

A solution will consist of a collection of k-tours that cover all n customers:

Definition 3.2.2. *(k-tour) A* k*-tour is a tour that cover at most* k *customers and the depot.*

In a *relaxed* set of tours, each tour is allowed to cover slightly more than k customers.

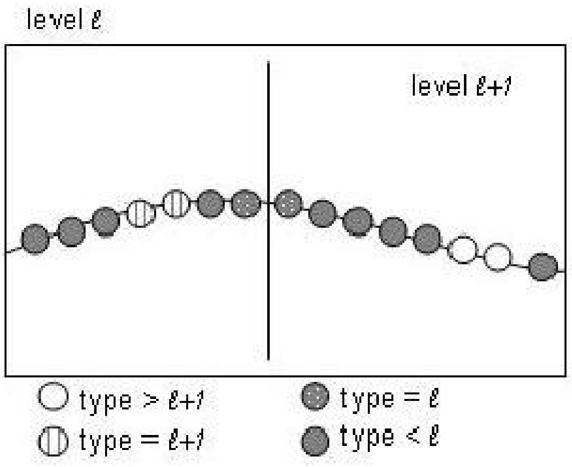

Figure 3.2: The figure shows boxes at levels $\ell + 1$ and ℓ, and four types of points. The points of type $> \ell + 1$ (white) and type $= \ell + 1$ (stripped) are inactive in all boxes shown. Points with type ℓ (dotted) are active at level $\ell + 1$. Points of type $< \ell$ (solid) are active in all shown boxes. If the thresholds are $5, 9$. The segment is rounded at level ℓ covering 9 active points. At level $\ell + 1$, the segment is rounded in the left box covering 5 active points and unrounded in the right box covering 6 active points.

Definition 3.2.3. *(Relaxed) A set of tours satisfies the* relaxed *conditions if they cover all the customers and there exists an assignment of types for the points such that:*

1. *Each tour visits the depot and covers at most k type 0 points.*
2. *Let b be a dissection box and let s be a segment in b that covers t active points at level(b). Then segment s has at most $t(1 + \epsilon/\log L)$ active points at level$(b) + 1$.*
3. *Let $\gamma = \lceil \log^4 L/\epsilon^4 \rceil$ be the rounding size. Any box b with at least γ is a rounded box containing only rounded segments.*

The first condition of the relaxed definition states that each tour can cover at most k type 0 points. These points which will be active in at level 0, and hence included in the final black tours. Each relaxed tour however can cover some additional points of type $i > 0$. Intuitively, the second condition allows a tour to acquire an additional $O(\epsilon/\log L)k$ active points at each, and as there are $O(\log L)$ levels in the dissection, the number of type > 0 points on a relaxed tour is at most $O(\epsilon)k$. The third condition ensures that boxes are rounded when they contain many segments ($\geq \gamma$), when it is too expensive to remember the exact number of points on each segment.

Definition 3.2.4. *(Structured) A set of tours Π is* structured*, with respect to a fixed random dissection D, if Π consists of tours which are portal respecting and light, and there exists a type assignment for the points that satisfies the relaxed conditions.*

We define an extended objective function F that in addition to minimizing the length of the tours will also charge for the number of tour crossings at each level.

Definition 3.2.5. *(Extended Objective Function) Fix a random dissection D and let Π be a set of k-tours. For every level ℓ let $c(\pi, \ell)$ be the number of times a tour $\pi \in \Pi$ crosses the boundaries of level ℓ boxes, and let $d_\ell = L/2^\ell$ denote the length of a level ℓ dissection box. The extended objective function is:*

$$F(\Pi) = \sum_{\pi \in \Pi} \mathit{length}(\pi) + \frac{\epsilon}{\log L} \sum_{\mathit{level}\ \ell} \sum_{\pi \in \Pi} c(\pi, \ell) \cdot d_\ell, \tag{3.1}$$

The structure theorem implies the existence of a near optimal solution consisting of structured tours, as made explicit in Corollary 3.2.7. Theorem 3.2.6 is proved in Section 3.3.

Theorem 3.2.6. *(Structure Theorem) Let Π be a set of k-tours that covers all n customers. Let Π' be the portal respecting and light tours obtained by applying Arora's structure theorem to every tour in Π. There exists a type assignment satisfying the relaxed conditions to make Π' structured, and in expectation over random shifts of the dissection, $F(\Pi') \leq (1 + O(\epsilon))\mathit{length}(\Pi)$.*

Corollary 3.2.7. *(Structure Corollary) Let OPT denote the length of the optimal solution for an instance I of UnitDem, and let OPT^S denote the length of the structured solution that minimizes objective function F. In expectation over random shifts of the dissection, $OPT^S \leq (1 + O(\epsilon))OPT$.*

Proof. Let Π^* be the optimal set of k-tours for I. Thus Π^* covers all n customers and $\text{length}(\Pi^*) = \text{OPT}$. By Theorem 3.2.6 there exists a structured set of tours Π' such that in expectation, $F(\Pi') \leq (1 + O(\epsilon))\text{length}(\Pi^*) = (1 + O(\epsilon))\text{OPT}$.

Let Π^S be the set of structured tours that minimizes F, thus $F(\Pi^S) \leq F(\Pi')$. The corollary follows as $\text{OPT}^S = \text{length}(\Pi^S) \leq F(\Pi^S) \leq F(\Pi') \leq (1 + O(\epsilon))\text{OPT}$. □

By Corollary 3.2.7 we can focus on computing the structured solution that minimizes objective F, which we will denote OPT^S. Section 3.4 presents a dynamic program that computes OPT^S in quasipolynomial time and proves the following theorem.

Theorem 3.2.8. *(Dynamic Program) For any instance I of the UnitDem problem, $OPT^S(I)$ is computed by the dynamic program of Section 3.4 in time $n^{\log^{O(1/\epsilon)} n}$.*

3.2.2 Assigning Types

The type assignment procedure "rounds" a tour segment by choosing the points that should be dropped from the segment. To round a segment S, Algorithm 7 chooses an interval of S, and sets the type of points in the interval to indicate they should be dropped. The interval is chosen such that its length is an $O(\epsilon)$ fraction of the length of S and such its Rad is an $O(\epsilon)$ fraction of the Rad of S. Choosing the interval with these properties will be enough to ensure that the constant factor approximation of the red points has small cost. See Figure 3.3 for an illustration of Algorithm 7.

3.2.3 A Constant Factor Approximation [29]

We use the tour partitioning algorithm of Haimovich and Rinnooy Kan to obtain a solution for the red dropped points. It partitions a TSP of the red points into tours that cover at most k

Algorithm 7 Randomized Type Assignment

```
Input: Level $\ell$ segment $S$ with $x$ active points
1: Choose an active point $p$ uniformly from the active points of $S$.
2: Let $t$ be the largest threshold $\leq x$ and choose an *interval* of $y = x - t$ active points from $S$ as follows:
3: if  $S$ has $y - 1$ active points after $p$ then
4:     Choose $p$ and the next $y - 1$ active points on $S$.
5: else
6:     Choose the $x_p$ active points from $p$ to the end of $S$, and the first $y - x_p$ active points from the start of $S$.
7: end if
8: Assign all $y$ points in the interval to type $\ell$.
Output: Segment $S$ with $t$ active points.
```

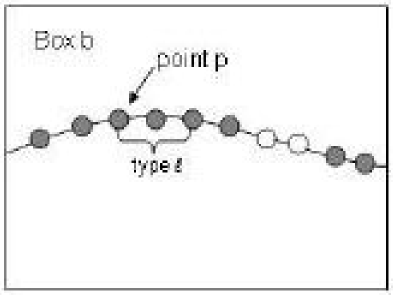

Figure 3.3: b is a level ℓ box with 8 active points (dark circles) and two inactive points (the white circles). If the largest threshold less than 8 is $t = 5$, then $y = 3$ points are marked to be dropped. Thus p and the next two active points are labelled type ℓ.

points. Theorem 3.2.9 was proved by Haimovich and Rinnooy Kan and shows the algorithm is a 3-approximation [29].

Algorithm 8 Tour Partitioning 3-approximation [29]

```
Input: Customers $C$ and a depot in any metric space, and integer $k$
1: Compute a tour $\pi$ of the customers and the depot with the 2-approximation of TSP from
   Lemma 1.3.6.
2: Choose a point $p$ uniformly at random from $\pi$.
3: Go around $\pi$ starting at $p$, and every time $k$ points are visited, take a detour to the depot.
Output the resulting set of $\lfloor n/k \rfloor + 1$ tours.
```

Theorem 3.2.9. *[29, 7] For any input I, an instance of the metric UnitDem problem with single depot, in expectation the output of Algorithm 8 has length at most $Rad(I) + 2 \cdot TSP(I) \leq 3OPT$.*

3.2.4 Proof of Main Theorem 3.1.1

The output of Algorithm 6 has cost equal to the length of the black tours plus the length of red tours. As the black tours are obtained by dropping points from the DP solution, their length is at most the length of the DP tours. By Theorem 4.3.3 the DP outputs OPT^S which by Corollary 3.2.7 has cost at most $(1+O(\epsilon))$OPT. Thus the length of the black tours is at most $(1+O(\epsilon))$OPT. The length of the red tours is at most $O(\epsilon)$OPT by Theorem 3.2.10, which is proved in Section 3.5.

Theorem 3.2.10. *In expectation over the random shifts of the dissection and the random type assignment, the length of the red tours output by Algorithm 6 is $O(\epsilon)OPT$.*

The DP dominates the running time. The derandomization of the Algorithm is discussed in Section 3.6.

3.3 Proof of the Structure Theorem 3.2.6

Given Π and dissection D, obtain Π' by applying Arora's algorithm (Algorithm 2) to each tour of Π. We show that Π' satisfies the requirements of the Theorem using two main Lemmas 3.3.1 and 3.3.3 given below.

For each tour $\pi \in \Pi$, let π^L denote its portal respecting and light version obtained using Arora's Algorithm 2. The first lemma proves that, $F(\pi^L) \leq (1+O(\epsilon)$length$(\pi)$. Its proof, which is presented in Section 3.3.1, is derived only by analyzing Arora's algorithm.

Lemma 3.3.1. *Let π be a tour and let π^L be the portal respecting and light tour obtained when Arora's structure theorem (Algorithm 2) is applied with π. We have that $F(\pi^L) \leq (1+O(\epsilon))length(\pi)$.*

As a corollary of Lemma 3.3.1 we get that,

Corollary 3.3.2. *Let Π be a any set of tours and Π^L be the set of portal respecting and light tours obtained when Arora's structure theorem (Algorithm 2) is applied to each tour in Π. We have that $F(\Pi^L) \leq (1+O(\epsilon))length(\Pi)$.*

Proof. $\text{length}(\Pi) = \sum_{\pi \in \Pi} \text{length}(\pi)$ and similarly $F(\Pi^L) = \sum_{\pi^L \in \Pi^L} F(\pi^L)$. The proof follows by applying Lemma 3.3.1 to each pair π, π^L and using linearity of expectation. □

Lemma 3.3.3 shows that there exists a type assignment which satisfies the relaxed conditions for Π'. See Section 3.3.2 for its proof.

Lemma 3.3.3. *Let* Π *denote a set of* k*-tours covering all customers. There exists a type assignment of the points such that* Π *satisfies the relaxed conditions of Definition 3.2.3.*

Π^L consists of portal respecting light tours and by Corollary 3.3.2 it satisfies the length condition of the structure theorem. As Π^L is a set of k-tours that cover all customers, by Lemma 3.3.3 there exists a type assignment such that Π^L satisfies the relaxed definition. Thus $\Pi' = \Pi^L$ is a structured solution which completes the proof of Theorem 3.2.6.

3.3.1 Proof of Lemma 3.3.1

By Equation, 3.1 $F(\pi^L)$ is,

$$F(\pi^L) = \text{length}(\pi^L) + \frac{\epsilon}{\log L} \sum_{\text{level } \ell} c(\pi^L, \ell) \cdot d_\ell \tag{3.2}$$

We get Equation 3.4 applying Arora's Structure Theorem 2.2.2 to the first term of Equation 3.2. We have that,

$$\begin{aligned}
F(\pi^L) &= (1 + O(\epsilon)) \cdot \text{length}(\pi) + \frac{\epsilon}{\log L} \sum_{\text{level } \ell} c(\pi^L, \ell) \cdot d_\ell && (3.3)\\
&\le (1 + O(\epsilon)) \cdot \text{length}(\pi) + \frac{\epsilon}{\log L} \sum_{\text{level } \ell} O(2 + \epsilon) \cdot \text{length}(\pi) && (3.4)\\
&\le (1 + O(\epsilon)) \cdot \text{length}(\pi) + \frac{\epsilon}{\log L} O(\log L) O(2 + \epsilon) \cdot \text{length}(\pi) && (3.5)\\
&\le (1 + O(\epsilon)) \cdot \text{length}(\pi) + O(\epsilon) \cdot \text{length}(\pi) && (3.6)
\end{aligned}$$

Equation 3.4 follows by applying Lemma 3.3.4 (given below) to each level ℓ. Equation 3.5 follows as there are $\ell_{\max} = O(\log L)$ levels in the dissection. Equation 3.6 shows that $F(\pi^L) \le (1 + O(\epsilon))\text{length}(\pi)$, proving the lemma.

Lemma 3.3.4. *For any level* ℓ, $E[c(\pi^L, \ell) \cdot d_\ell] \le O(2 + \epsilon) \cdot \textit{length}(\pi)$.

Proof. Let π represent any tour and π^L represent the portal respecting light tour output by Algorithm 2. As discussed in the previous chapter π^L may contain some new crossings that π does not have. We bound the number of crossings in π^L in terms of the crossings in π.

Recall that $c(\pi, \ell)$ denotes the numbers of times π crosses the boundaries of level ℓ boxes, and that $t(\pi, l)$ denotes the number of times tour π crosses dissection line l. The boundaries of level ℓ boxes are formed by the lines at levels $\le \ell$ and by Equation 2.1 the probability that a line has level $\le \ell$ is $2^{\ell+1}/L$. Thus for any level ℓ, we have,

$$E(c(\pi, \ell)) = \sum_{\text{line } l} t(\pi, l) \cdot \Pr[l \text{ is boundary of level } \ell \text{ box }] = \frac{2^{\ell+1}}{L} \sum_{\text{line } l} t(\pi, l) \tag{3.7}$$

The following property relates the length of the tour to the number of crossings on any level, and is an implication of Property 2.5.1..

Property 3.3.5. *Let π be a tour and ℓ a level of the dissection D. Let $d_\ell = L/2^\ell$ denote the side length of a level ℓ dissection box. In expectation we have, length$(\pi) \geq O(d_\ell)E[c(\pi, \ell)]$.*

Proof. Combine Property 2.5.1 with Equation 3.7 to get,

$$\text{length}(\pi) \;\geq\; \frac{1}{2}\sum_{\text{line l}} t(\pi, l) \;=\; \frac{1}{2}\cdot\frac{L}{2^{\ell+1}}E[c(\pi,\ell)] \;=\; \frac{d_\ell}{4}\cdot E[c(\pi,\ell)] = O(d_\ell)E[c(\pi,\ell)]$$

□

We apply Lemma 3.3.6 (given below) to bound the expected number of crossings in π^L in terms of the number of crossings in π, and then apply Property 3.3.5 to complete the proof as,

$$E[c(\pi^L,\ell)\cdot d_\ell] \leq (2+O(\epsilon))E[c(\pi,\ell)]\cdot d_\ell \leq O(2+\epsilon)\text{length}(\pi)$$

□

Lemma 3.3.6. *For any level ℓ $E[c(\pi^L, \ell)] \leq (2 + O(\epsilon))E[c(\pi, \ell)]$.*

Proof. We bound the expected number of crossings in π^L in terms of the number of crossings in π, by counting the number of new crossings in π^L introduced from patching and portal detours.
New crossings from patching. Suppose l is a line at level$(l) = i < \ell$. Let $p_{l,j}$ denote the number of times patching is applied to line l in the jth iteration of the for loop of Algorithm 3, to reduce original crossings (i.e from steps 1 or 2 of Algorithm 2). Note that $p_{l,j} = 0$ if $j < i$. Each application of patching replaces at least $r - 4$ original crossings on l by at most 4 original crossings. Thus we have,

$$\sum_{j\geq i} p_{l,j} \leq \frac{t(\pi, l)}{r-3} \tag{3.8}$$

Patching on line l in the j-th iteration augments π with a segment s of length $\leq L/2^j$. For $j < \ell$ we have,

$$(\#\text{level}(\ell)\text{ boxes lying on each side of } s) = \frac{L/2^j}{L/2^\ell} = 2^{\ell-j}$$

By Properties 2.5.2 and 2.5.4, patching adds at most 2 new crossings to the boxes lying on s. As there are $2^{\ell-j}$ level ℓ boxes lying on each side of l, each patching adds at most $4\cdot 2^{\ell-j}$ new crossings to level ℓ boxes. As Algorithm 3 can patch line l for each $j \geq i$, the total number of new crossings introduced to level ℓ boxes from patching on line l is

$$(\#\text{ new crossings to boxes at level}\ell) \leq \sum_{j\geq i} p_{l,j}\cdot 4\cdot 2^{\ell-j}$$

Patching line l contributes new crossings only if level$(l) = i$, thus the expected number of new crossings on level ℓ boxes is

$$\begin{aligned}
&E(\text{\# new crossings at level } \ell \text{ from patching on line } l) \\
&= \sum_{\ell > i \geq 1} \Pr[\text{level}(l) = i] \cdot \sum_{\ell > j \geq i} \left(p_{l,j} \cdot 4 \cdot 2^{\ell - j}\right) \\
&\leq 4 \sum_{\ell > j \geq 1} p_{l,j} \cdot 2^{\ell - j} \sum_{i \leq j} \Pr[\text{level}(l) = i] \\
&= 4 \sum_{\ell > j \geq 1} p_{l,j} \cdot 2^{\ell - j} \sum_{i \leq j} \frac{2^i}{L} \qquad (3.9) \\
&= 4 \sum_{\ell > j \geq 1} p_{l,j} \cdot 2^{\ell - j} \cdot \frac{2^{j+1}}{L} \\
&\leq 4 \cdot \frac{t(\pi, l)}{r - 3} \cdot \frac{2^{\ell+1}}{L} \qquad (3.10)
\end{aligned}$$

where Equation 3.9 follows by Equation 2.1 and Equation 3.10 follows from Equation 3.8.

Equation 3.10 gives the expected number of new crossings from patchings on any level $i < \ell$ line. Summing over all lines l, using Equation 3.7, and the fact that $r = O(1/\epsilon)$ we have that,

$$\begin{aligned}
E(\text{\# new crossings at level } \ell \text{ from patching}) &\leq \sum_{\text{line } l} 4 \cdot \frac{t(\pi, l)}{r - 3} \cdot \frac{2^{\ell+1}}{L} \\
&\leq 4 \cdot \frac{E[c(\pi, \ell)]}{r - 3} = O(\epsilon) E[(c(\pi, \ell)] \qquad (3.11)
\end{aligned}$$

New crossings from portal detours. Recall that a level i line has $2^i m$ portals, equidistance apart, where $m = O(\log L/\epsilon)$. Thus a portal detour on a level i augments the tour with a segment s of length at most $\frac{L}{2^i \cdot m}$. For any level $\ell > i$, there are at most $\frac{L/(2^i m)}{L/2^\ell} = 2^{\ell - i}/m$ level ℓ boxes lying on each side of segment s. By Properties 2.5.3 and 2.5.5, s adds one new crossing to each of these boxes adding a total of $2 \cdot 2^{\ell - i}/m$ new crossings at level ℓ. At most $E[(c(\pi, i)]$ portal detours are required for level i lines. Recall that by Property 2.5.6, all new crossings introduced from patching were at corners of boxes which are portals. Thus we have that,

$$\begin{aligned}
&E(\#\text{ new crossing at level } \ell \text{ from portal detours})\\
&= \sum_{i\le\ell} (\#\text{ detours at level } i) \cdot \frac{2^{\ell-i}}{m}\\
&\le \sum_{i\le\ell} E[(c(\pi, i)] \cdot \frac{2^{\ell-i}}{m}\\
&= \sum_{i\le\ell} \left(\frac{2^{i+1}}{L} \sum_{\text{line } l} t(\pi, l) \right) \cdot \frac{2^{\ell-i}}{m} \qquad (3.12)\\
&= \sum_{i\le\ell} \left(\frac{2^{\ell+1}}{L} \sum_{\text{line } l} t(\pi, l) \right) \cdot \frac{1}{m} \qquad (3.13)\\
&= \sum_{i\le\ell} \frac{E[(c(\pi, \ell)]}{m}\\
&\le O(\log L) \frac{E[c(\pi, \ell)]}{m} = E[c(\pi, \ell)] \qquad (3.14)
\end{aligned}$$

Equation 3.12 and 3.13 follow from Equation 3.7 and Equation 3.14 as there are $O(\log L)$ levels.

Total crossings. The expected number of π^L crossings at level ℓ is at most the original crossing of π at level ℓ plus the number of new crossings from patching and detours derived in Equations 3.11 and 3.14.

$$E[c(\pi^L, \ell)] \quad \le \quad E[c(\pi, \ell)] + O(\epsilon)E[c(\pi, \ell)] + E[c(\pi, \ell)] \quad = \quad (2 + O(\epsilon))E[c(\pi, \ell)]$$

□

3.3.2 Proof of Lemma 3.3.3

Given a set of k-tours Π that covers all customers, Algorithm 9 assign types to points such that Π satisfies the relaxed conditions of Definition 3.2.3.

Algorithm 9 Relaxed type assignment for a set of k-tours [29]

```
Input: $\Pi$ a set of $k$-tours covering all customers in $C$
1: Label all points as type 0
2: for level $\ell$ from $\ell_{\max}$ to 0 do
3:    for each level $\ell$ box $b$ with more than $\gamma$ segments do
4:       for each segment $s$ in $b$, round $s$ as follows do
5:          Let $x$ be the number of active points on segment $s$, and $t_i$ the largest threshold s.t $t_i \le x$.
            Pick any $x - t_i$ active points on $s$ and label them as type $\ell$.
6:       end for
7:    end for
8: end for
Output: $\Pi$ and the type assignment for points.
```

Observe that Algorithm 9 only labels points with types and does not modify the structure of any tour. As Π is initially a set of k-tours covering all n customers, this is also true after Algorithm

9 completes. Initially each tour in Π covers at most k customer points. Thus after the first step of the algorithm, all tours in Π cover at most k points of type 0, thus the first condition of Definition 3.2.3 is satisfied. Since Algorithm 9 only labels points with types ≥ 0, every tour in Π continues to cover at most k type 0 points after the Algorithm completes. Thus the first condition of Definition 3.2.3 will be satisfied.

Now consider the third condition. Consider a box b at level ℓ containing more than γ segments. Recall that the active points in b are points of type $< \ell$. The for loop in lines 4-6 rounds each segment of b by increasing the type of some points on the segment to ℓ so that the segment covers exactly a threshold number of active points. Thus the third condition of Definition 3.2.3 holds for b once Algorithm 9 has finished working on level ℓ. While the algorithm works at levels $j < \ell$, points are only labelled with types less than ℓ. Thus the number of active points remains the same at level ℓ and the condition continues to hold in box b.

Now consider the second condition. Consider a segment prior to and after it is rounded at level ℓ. Prior to rounding all points on the segment either have type 0 or a type strictly greater than ℓ, so the segment has the same number of active points, at level ℓ and at level $\ell + 1$. Let x be the number of active points prior to rounding such that $t_i \leq x \leq t_{i+1}$. After rounding, the segment has $x - t_i$ points labelled with type ℓ. This leaves t_i active points at level ℓ and x active points at level $\ell + 1$. As $t_i(1 + \epsilon/\log L) = t_{i+1} > x$, the second condition of Definition 3.2.3 is satisfied.

3.4 The Dynamic Program

The DP Table. For each dissection box b, the DP table will have an entry for all the possible ways to extend a solution of b. Observe that given a solution of b, to extend it outside, only requires us to know about the tour segments that cross the boundary of b, and we can essentially ignore any tours that are completely contained inside b. Thus we define a *configuration* C of a dissection box b, to describe only the tour segments that cross the boundaries of b. A box is either described by a rounded configuration or an unrounded configuration.

- Rounded Configuration: Consists of a list of numbers $r_{p,q,t,d}$, for every pair of portals p, q of b, every threshold $t \in \{t_1, \ldots, t_\tau\}$, and depot indicator bit $d \in \{0, 1\}$. The number $r_{p,q,t,d}$, is the number of rounded tour segments that use portals p and q to enter and exit b, cover exactly t active points, and visit the depot as indicated by d.

- Unrounded Configuration: Contains a list of γ tuples of the form (p, q, m, d). The j-th tuple $(p, q, m, d)_j$ denotes the j-th unrounded tour segment that covers exactly m points and enters and exits b from portals p to q and visits the depot as indicated by d.

The DP has a table entry $L_b[C]$ for each dissection box b and each configuration C of b. $L_b[C]$ is the minimum cost of placing tour segments in b which are compatible with C, and are structured as defined by Definition 3.2.4, where cost is computed by objective F. The DP returns OPT^S as the minimum table entry over all configurations of the root level box.

Computing the table entries. The table entries are computed in bottom-up order. The base case computes $L_b[C]$ for a leaf box b. First assume that b contains no depot. If C is a rounded configuration whose segments together cover t points, then C is feasible for b if the segments cover all but at most $t(\epsilon/\log L)$ points of b. Otherwise if C is an unrounded configuration, C is feasible for b if all points of b are covered by the segments described by C. It is easy to compute the cost of a feasible C for leaf boxes as all points are located in the center of the box. Now suppose that b contains a depot. Then all points of b which are not covered by any segment described in C can be covered at zero cost by tours that stay completely inside (i.e at the center of) b.

Inductively, let b be a box at level ℓ and let b_1, b_2, b_3, b_4 be the children of b at level $\ell + 1$. As every tour is structured (and in particular portal respecting and light), a tour segment (or a tour) in b crosses the boundaries of b_1, b_2, b_3, b_4 inside b, at most $4r$ times, and always through portals. Every tour segment and tour in b is the concatenation of at most $4r + 1$ pieces, where a piece uses two portals of b_1, b_2, b_3, b_4, and either visits a depot or not. Each piece is either a rounded segment or one of the at most γ unrounded segments inside a child of b. Thus each piece can be described by a tuple (p, p', x, d), where p, p' are portals, d is the depot indicator flag and x is either a threshold t_i for some $i < \tau$, or x is a number $j \leq \gamma$ indicating that the piece is the j-th unrounded tour in a child box of b. Each tour segment and tour in b is described by a *concatenation profile* $\Phi = (p_1, p_2, x_1, d_1), (p_2, p_3, x_2, d_2), \ldots (p_s, p_{s+1}, x_s, d_s)$, which is a list of the at most $4r + 1$ tuples it is constructed from. For a given Φ, let x_Φ denote the sum of active points picked up by all pieces in Φ. If a piece $(p, p'x, d)$ comes from a rounded box, it contributes $x = t_i$ (some threshold t_i) number of active points to x_Φ. Otherwise if the piece is from an unrounded box, x is a number less than or equal to γ, and the piece contributes m_x, (the exact) number of active points picked up by the x-th unrounded segment of a child box of b.

A profile Φ with $p_1 = p_{s+1}$ describes a tour (rather than a tour segment) in b, It is feasible if at least one of its tuples visits the depot, i.e has indicator $d = 1$ and $x_\Phi \leq k$. Suppose Φ describes a tour segment in b. If C is an unrounded configuration for b, the segment described by Φ is unrounded and the DP counts the segment as having exactly x_Φ active points. Otherwise if C is a rounded configuration for b, the segment described by Φ is rounded and the DP counts the segment as having t_i active points where t_i is the largest threshold less than x_Φ i.e, $t_i \leq x_\Phi < t_i(1 + \epsilon/\log L)$.

Let φ denote the number of different concatenation profiles possible for b. Let n_i, for $i \leq \varphi$, denote the number of tour segments in b with concatenation profile Φ_i. We define an *interface vector* $\mathcal{I} = (n_i)_{i \leq \varphi}$ as a list of φ entries. Intuitively, the vector $\mathcal{I}$ describes the composition of *all* the segments in b and provides the interface between how tour segments in b are formed by concatenating the segments of b's children.

Let C_0 be a configuration for box b. The calculation of $L_b(C_0)$ is done in a brute force manner by iterating through all possible values of the interface vector $\mathcal{I}$ and all possible combinations of configurations in b's children, C_1, C_2, C_3, C_4. A combination $C_0, \mathcal{I}, C_1, C_2, C_3, C_4$ is *consistent* if $\mathcal{I}$ describes at most γ unrounded segment, and if gluing C_1, C_2, C_3, C_4 according to $\mathcal{I}$ yields configuration C_0.

The cost of configurations $\{C_i\}_{i\leq 4}$ is stored in lookup tables $L_{b_i}(C_i)$. Let $c_b(\mathcal{I})$ be the total number of tour segments in b as specified by $\mathcal{I}$. The cost according to objective function F, of $(C_1, C_2, C_3, C_4, \mathcal{I})$ is $(\epsilon/\log L)\cdot 2c_b(\mathcal{I})$ plus the sum of the costs of $L_{b_i}(C_i)$ for each child box $i \leq 4$. Entry $L_b(C_0)$ is set equal to the cost of the tuple $(C_1, C_2, C_3, C_4, \mathcal{I})$ that is consistent with C_0 and has minimum cost.

Running time of dynamic program. The number of possible configurations for a box b is the number of possible rounded configurations plus the number of unrounded configurations. A rounded configuration consists of a list of at most $O(\tau \log^2 L)$ entries as there are $O(\log^2 L)$ different pairs of portals (p, q), τ different thresholds, and two possible values for indicator d. As $\tau = O(\log^2 L)$, the rounded configuration contains a list of $O(\log^4 L)$ numbers. Each number in the list $r_{p,q,t,d}$ is an integer that is at most n, thus there are $n^{O(\log^4 L)}$ different rounded configurations. Each unrounded configuration contains a list of $\gamma = O(\log^4 L)$ tuples (p, q, m, d) and there are $\log^2 L \cdot n \cdot 2$ possibilities for each tuple. Thus there are $(2n\log^2 L)^\gamma = n^{O(\log^4 L)}$ possible unrounded configurations. As each box is described by either a rounded or an unrounded configuration, there are $n^{O(\log^4 L)}$ configurations per box. As there are $O(n)$ non empty boxes, the DP table has overall size $n^{O(\log^4 L)}$ which is $n^{O(\log^4 n)}$ as $L = O(n)$.

The number of possible concatenation profiles for segments in box b is computed as follows. Each Φ has a list of $O(r)$ tuples (p, p', x, d). There are $O(\log^2 L)$ choices of portals p, p', two choices for d, and $\gamma + \tau$ choices of x, so there are $O((\tau+\gamma)\log^2 L) = O(\log^6 L)$ possibilities for each tuple. As $r = O(1/\epsilon)$, there are $\varphi = (\log^6 L)^{O(r)} = (\log L)^{O(1/\epsilon)}$ possible concatenation profiles.

The interface vector $\mathcal{I}$ has φ entries, and each entry is an integer less than n. Thus we have n^φ, possible interface vectors for a box b, a quasi-polynomial number of possibilities.

Checking consistency for a particular interface vector $\mathcal{I}$ and configurations $\{C_i\}_{i\leq 4}$ can be done in polynomial time in the size of their descriptions. There are $n^{O(\log^4 L)}$ possible values for each C_i and $n^{\log^{O(1/\epsilon)} L}$ possible values for $\mathcal{I}$. Thus to run through all combinations of $\mathcal{I}, C_1, C_2, C_3, C_4$ and to compute the lookup table entry at $L_b[C_0]$ takes time polynomial in $n^{\log^{O(1/\epsilon)} L}$ which is $n^{\log^{O(1/\epsilon)} n}$ as $L = O(n)$.

Remark. The DP only verifies that for the set of k-tours it returns there exists a type-assignment satisfying the relaxed Definition 3.2.3. However the DP does not actually label points with specific types. Instead it records the number of active points the tour segments it constructs should have. Once the value of OPT^S is found, we can trace through the DP history and make type assignments while constructing the tours of OPT^S. If we construct a tour segment with x active points at level ℓ that the DP's history recorded as having t active points, choosing *any* $x - t$ active points from the segment and labelling them with type ℓ would satisfy the relaxed definition. However choosing the $x - t$ active points arbitrarily may not always be sufficient to ensure that the labelled points, which will be dropped later, can all be covered with small cost. To ensure that we will use the randomized type assignment Algorithm 7 to choose the points that will be labelled.

3.5 Proof of Theorem 3.2.10

For any input I, let R denote the points colored red by Algorithm 6. The red tours are found using the 3-approximation Algorithm 8 of [29] by Theorem 3.2.9 they have length at most $\text{Rad}(R) + 2\text{TSP}(R \cup depot)$. The following lemmas bounds each of these terms by $O(\epsilon)\text{OPT}$, which proves that the theorem.

Lemma 3.5.1. *For any instance I, let R denote the points of I colored red by Algorithm 6. In expectation over the random type assignment, $Rad(R) = O(\epsilon)OPT$*

Lemma 3.5.2. *For any instance I, let R denote the points of I colored red by Algorithm 6. In expectation over the random dissection and type assignment $TSP(R \cup depot) = O(\epsilon)OPT$*

3.5.1 Proof of Lemma 3.5.1

For each level $\ell \in [0, \ell_{\max}]$ let R_ℓ denote the points labelled as type ℓ. Thus the red points R, is the union of all R_ℓ over all levels. Thus,

$$\Pr[i \in R] = \sum_{\ell > 0} Pr[i \in R_\ell] \tag{3.15}$$

We compute the probability that a point i is in R_ℓ. If $i \in R_\ell$, it has been labelled type ℓ. Thus i must have been chosen in the interval by Algorithm 7 while rounding the level ℓ segment S that covers i. Property 3.5.3 (given below) shows the probability that i is chosen in the interval, and hence included in R_ℓ, is $\Pr[i \in R_\ell] = O(\epsilon/\log L)$. As there are $\ell_{\max} = O(\log L)$ levels, Equation 3.15 reduces to,

$$\Pr[i \in R] = O(\epsilon)$$

Thus we have that,

$$E[\text{Rad}(R)] = \frac{2}{k}\sum_{i\in I} d(i,o)\cdot Pr[i \in R] \le \frac{2}{k}\sum_{i \in I} d(i,o)\cdot O(\epsilon) = O(\epsilon)\text{Rad}(I).$$

The lemma follows as $\text{Rad}(I)$ is a lower bound on OPT by Lemma 1.3.2.

Property 3.5.3. *Let S be a level ℓ segment that is rounded by Algorithm 7, and let i be an active point covered by S prior to rounding. Then point i is in interval chosen by Algorithm 7, with probability at most $O(\epsilon/\log L)$.*

Proof. Let x be the number of active points covered by S prior to its rounding, and let threshold t_i be the largest threshold such that $t_i \le x < t_{i+1}$. To round S the type assignment procedure Algorithm 7, labels $y = x - t_i$ active points of S as type ℓ. The y points are selected as an *interval* by starting at a uniformly selected active point p, and then selecting the $y - 1$ active points lying after p, wrapping around s if necessary.

There are a total of x different intervals, each one starting at a different active point p of S, and point i lies in y different intervals. As Algorithm 7 picks uniformly among these intervals, the probability that i lies in the selected interval is y/x. By definition of thresholds (Definition 3.2.1), $t_{i+1} \leq t_i(1 + \epsilon/\log L)$, so we have that,

$$Pr[i \text{ is in the interval}] = \frac{y}{x} = \frac{x - t_i}{x} < \frac{t_{i+1} - t_i}{t_i} \leq \frac{\epsilon}{\log L}.$$

□

3.5.2 Proof of Lemma 3.5.2

For each level $\ell \in [0, \ell_{\max}]$ let R_ℓ denote the type ℓ points of I, and note that $R = \cup_{\ell > 0} R_\ell$. We prove that for each ℓ, $E[\text{TSP}(R_\ell \cup \textit{depot})] \leq O(\epsilon/\log L)\text{OPT}$. This implies the lemma since there are $\ell_{\max} = O(\log L)$ levels so the tours of R_ℓ, for all ℓ, can be pasted together at the depot to get a tour of R of cost at most $O(\epsilon)\text{OPT}$.

Focus on a single level ℓ and let B_ℓ be the boxes at level ℓ containing points from R_ℓ. We upper bound the cost of $\text{TSP}(R_\ell \cup \textit{depot})$ by thinking of the tour in three parts. For each box $b \in B_\ell$, let R_b denote the points of R_ℓ lying inside b. The *inside* parts consists of tours of the points in R_b connected to the boundary of b, for each box $b \in B_\ell$. The *boundary* part includes the boundaries of all boxes in B_ℓ. Given the *inside* and the *boundary* of each $b \in B_\ell$, to get a tour of R_ℓ we only need a tour of the boxes B_ℓ and the depot. We refer to this as the *outside* part. Thus we have that,

$$\text{TSP}(R_\ell \cup \textit{depot}) \leq \sum_{b \in B_\ell} (\text{TSP of } R_b \text{ connected to boundary of } b) + (\text{boundary of } b) + \text{ outside cost}$$

For the *outside* cost we define a complete graph $G = (V, E)$ such that V contains a vertex for each box in B_ℓ and one additional vertex to represent the depot. The cost of the edges of G are as follows: the edge between two box vertices b, b' has cost equal to the length of the shortest path from any portal of b to any portal of b'. The cost of an edge between a box vertex b and the depot vertex is equal to the length of the shortest path, from any portal of b to the depot. By Lemma 1.3.6 the *outside* cost; i.e a tour of the depot and boxes in B_ℓ, is at most $2MST(G)$. Thus we have that:

$$TSP(R_\ell \cup \textit{depot}) \leq \sum_{b \in B_\ell} (\text{TSP of } R_b \text{ connected to boundary of } b) + (\text{boundary of } b) + 2MST(G) \tag{3.16}$$

By Lemmas 3.5.4 and 3.5.5 the *inside* and *boundary* costs are both $O(\epsilon/\log L)F(\text{OPT}^S)$. Lemma 3.5.6 shows that $2MST(G)$ is $O(\epsilon/\log L)\text{OPT}^S$. As $\text{OPT}^S \leq F(\text{OPT}^S)$, we have that $2MST(G) \leq O(\epsilon/\log L)F(\text{OPT}^S)$, which by the structure Corollary 3.2.7 is at most $(1 + O(\epsilon))\text{OPT}$. Thus all three costs $O(\epsilon/\log L)\text{OPT}$, proving the lemma.

Lemma 3.5.4. *(Inside cost) In expectation over the random dissection and the random type assignment,* $\sum_{b \in B_\ell}(\textit{TSP of } R_b \textit{ connected to boundary of } b) \leq O(\epsilon/\log L)F(\textit{OPT}^S)$

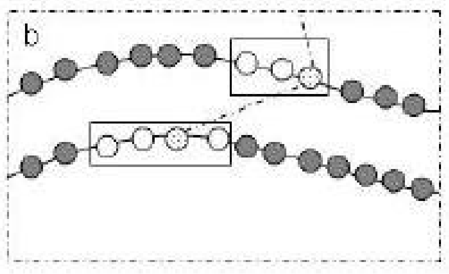

Figure 3.4: The figure shows a box $b \in B_\ell$. The white points have type ℓ and were dropped. Claim 3.5.7 shows that total length of the white intervals (boxed segments) is small. Claim 3.5.9 shows that the cost of connecting the white intervals to the boundary is small (dashed lines).

Lemma 3.5.5. *(Boundaries cost) In expectation over the random dissection and the random type assignment, $\sum_{b \in B_\ell}(\textit{boundary of } b) \leq O(\epsilon/\log L)F(OPT^S)$.*

Lemma 3.5.6. *(Outside cost) In expectation over the random shifted dissection, $E[2MST(G)] \leq O(\epsilon/\log L)OPT^S$.*

Proof of Lemma 3.5.4

Recall that for each $b \in B_\ell$, the type assignment procedure (Algorithm 7) chooses R_b by selecting intervals of the segments of b, and labelling the points in the interval as type ℓ. The TSP of R_b connected to the boundary of b is at most the sum the lengths of these intervals plus the cost of connecting the intervals together and to the boundary of b. See Figure 3.4.

Claim 3.5.7 shows that in expectation over the random type assignment the sum of the lengths of these intervals over all boxes in B_ℓ is $O(\epsilon/\log L)\text{OPT}^S$. Define the *connection cost* of b to be the cost of connecting the intervals of b together and to the boundary of b. Let $C(\ell)$ denote the sum of the connection costs for all boxes $b \in B_\ell$. Claim 3.5.9 shows that $C(\ell) = O(\epsilon/\log L)F(\text{OPT}^S)$. The proof of the lemma follows by combining Claims 3.5.7 and 3.5.9.

Claim 3.5.7. *In expectation over the random type assignment, the sum of the lengths of type ℓ intervals across all boxes in B_ℓ is $O(\epsilon/\log L)OPT^S$.*

Proof. Property 3.5.8 shows that for any segment S the expected length of its type ℓ interval is $O(\epsilon/\log L)$ times the length of S. Thus by linearity of expectation the sum of all type ℓ intervals over all boxes B_ℓ is at most $O(\epsilon/\log L)$ times the total length of all segments in B_ℓ. The claim follows as the sum of all segments in B_ℓ is at most OPT^S. □

Property 3.5.8. *For a segment S the interval chosen by Algorithm 7 when rounding S has expected length $O(\epsilon/\log L)\textit{length}(S)$.*

Proof. Let x be the number of active points on segment S. List the active points in the order they appear on segment S: $p_1, p_2, \ldots, p_x$. Thus p_1 is the first active visited by S after crossing the boundary and p_x is the last active point visited by S before crossing the boundary. For $i < x$, let

l_i denote the length of segment S between p_i and p_{i+1} and let l_x be the length of S from p_x to boundary plus the length of S from the boundary to p_1. Thus we have that,

$$\text{length}(S) = \sum_{i \le x} l_i$$

Let threshold t_i be such that $t_i \le x < t_{i+1}$. Algorithm 7 chooses the interval by uniformly selecting a starting active point and then selecting the next $y = x - t_i$ consecutive active points, wrapping around if necessary. Thus if point i is in the interval, it contributes at most l_i to the length of the interval. By Property 3.5.3, we have that

$$\begin{aligned} E[\text{length}(S)] &\le \sum_{i \le x} Pr[i \text{ is in the interval}] \cdot l_i \\ &\le \sum_{i \le x} \frac{\epsilon}{\log L} l_i \\ &= \frac{\epsilon}{\log L} \text{length}(s) \end{aligned}$$

□

Claim 3.5.9. *The total connection cost for level ℓ, $C(\ell)$ is $O(\epsilon/\log L)F(OPT^S)$.*

Proof. Recall that $C(\ell)$ is the sum of the connection cost for all boxes in B_ℓ. To prove the lemma we will show that the connection cost for each $b \in B_\ell$ we let OPT_b^S denote the cost of OPT^S inside b, and prove that,

$$\text{Connection cost of } b \le O(\epsilon/\log L)F(\text{OPT}_b^S) \tag{3.17}$$

Then summing Equation 3.17 over all $b \in B_\ell$ proves the lemma.

Fix any $b \in B_\ell$. The connection cost of box b is the cost of connecting the type ℓ rounded intervals inside b together with the boundary of b. Select set R' to contain one representative type ℓ point from each type ℓ rounded interval in b. Let d_ℓ denote the side length of b, then the connection cost of b is at most,

$$\text{Connection cost of } b \le MST(R') + d_\ell/2 \tag{3.18}$$

We bound the connection cost of b using the following bound for TSP [29][7]. (See [29] for a proof).

Theorem 3.5.10. *[29][7] Let U be a finite set of points in a 2-dimensional box of side length $d_{\max}$. Then $TSP(U) = O(d_{max}\sqrt{|U|})$*

In our context, $d_{max} = d_\ell$, and $U = R'$. Combining Theorem 3.5.10 with Equation 3.18 we get

$$\text{Connection cost of } b \le MST(R') + d_\ell = O(d_\ell\sqrt{|R'|})$$

As b is a rounded box, the segments in b can be partitioned into $g_b \ge 1$ groups, each containing exactly γ rounded segments, and possibly one additional group containing less than γ rounded segments. As R' contains one representative from each rounded segment in b, R' has size at most $\gamma(g_b+1) \leq 2\gamma \cdot g_b$. Thus we have that

$$\text{Connection cost of } b \leq O(d_\ell\sqrt{2\gamma \cdot g_b}) \tag{3.19}$$

Each tour segment of b contributes two crossings at a level ℓ box and objective function F (Equation 3.1) charges OPT^S the amount $(\epsilon/\log L)\cdot d_\ell$, for each crossing at a level ℓ box. Thus Equation 3.20 gives the charge for b and we have that,

$$F(\text{OPT}^S_b) \geq 2\frac{\epsilon}{\log L}\cdot d_\ell \cdot \gamma \cdot g_b \tag{3.20}$$
$$\geq \frac{\epsilon}{\log L}\cdot\sqrt{2\gamma g_b}\cdot(\text{ Connection cost of } b) \tag{3.21}$$
$$\geq O\left(\frac{\log L}{\epsilon}\right)\cdot\sqrt{g_b}\cdot(\text{ Connection cost of } b) \tag{3.22}$$

Above Equation 3.21 follows by combining Equation 3.20 with Equation 3.19. Equation 3.22 follows as $\sqrt{\gamma} = \log^2 L/\epsilon^2$. As $g_b \geq 1$ Equation 3.17 is proved. □

Proof of Lemma 3.5.5

The proof follows the proof of Claim 3.5.9. Let $|B_\ell|$ denote the number of level ℓ boxes containing type ℓ points. The sum of the boundaries of these boxes is

$$\sum_{b\in B_\ell}(\text{boundary of } b) = 4d_\ell|B_\ell| \tag{3.23}$$

Each box $b \in B_\ell$ contains at least γ rounded segments. Each rounded segment has two crossings with the boundary of a level ℓ dissection box. As objective function F (Equation 3.1), charges each crossing of a level ℓ box, $(\epsilon/\log L)d_\ell$ Equation 3.24 follows, and we have that,

$$F(\text{OPT}^S) \geq 2\frac{\epsilon}{\log L}d_\ell\gamma|B_\ell| \tag{3.24}$$
$$\geq 2\frac{\epsilon}{\log L}d_\ell\gamma\sum_{b\in B_\ell}(\text{boundary of } b) \tag{3.25}$$
$$\geq O\left(\frac{\log^3 L}{\epsilon^3}\right)\sum_{b\in B_\ell}(\text{boundary of } b) \tag{3.26}$$

Equation 3.25 follows by combining equation 3.24 with Equation 3.23. Equation 3.26 follows as $\gamma = \log^4 L/\epsilon^4$. This shows that $\sum_{b\in B_\ell}(\text{boundary of } b) = o(\epsilon/\log L)F(\text{OPT}^S)$.

Proof of Lemma 3.5.6

As each $b \in B_\ell$ contains points labelled ℓ there are at least γ rounded segments in b and thus OPT^S has at least γ tour segments crossing into b. As each tour in OPT^S is structured (and in particular light), each tour crosses b at most $4r$ times. Thus there are at least $\gamma/4r$ tours entering b.

Let P be a multigraph that denotes the projection of OPT^S onto graph G where every edge in OPT^S between two boxes $b, b' \in B_\ell$ and between the depot and a box $b \in B_\ell$ is represented by an

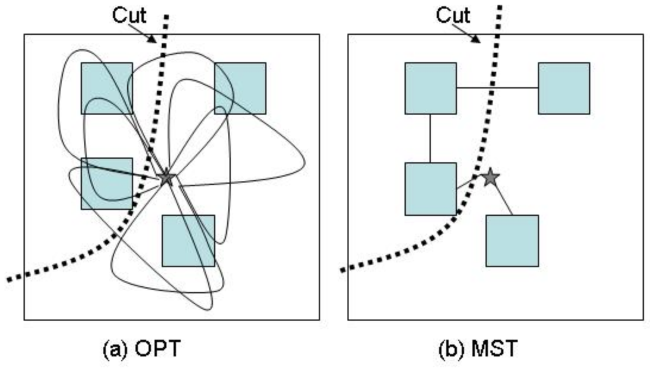

Figure 3.5: The shaded boxes are the boxes of B_ℓ. (a) Given that OPThas at least 3 tours entering each box, OPTcrosses all non-trivial cuts at least 6 times. This is made explicit in Equation 3.27. (b) The MST crosses all non-trivial cuts at least once as expressed in Equation 3.28

edge in P. Clearly $cost(P) \leq \text{OPT}^S$ as the cost of edges in G are always at most the distances of the edges in I. As there are has at least $\gamma/4r$ tours in OPT^S crossing each $b \in B_\ell$, P has at least $\gamma/4r$ edges crossing any cut separating the depot vertex from any box vertices. See Figure 3.5. Consider the linear program in Equation 3.27 on G with V denoting the vertices of G and w_e the cost of edge e in G. The linear program finds w the minimum cost way to selected a set of edges in G such that each non-trivial subset of V is crossed at most $\gamma/4r$ times. Thus the $cost(P) \geq w$.

$$w = \min \sum_{\text{edge } e} w_e x_e \quad \text{s.t.} \begin{cases} \sum_{e\in\delta(S)} x_e \geq \gamma/4r & \forall S \subset V \\ & S \neq \emptyset \\ & S \neq V \\ x_e \geq 0 \end{cases} \tag{3.27}$$

Consider linear program in Equation 3.28, which is the IP relaxation of MST on G.

$$w' = \min \sum_{\text{edge } e} w_e x_e \quad \text{s.t.} \begin{cases} \sum_{e\in\delta(S)} x_e \geq 1 & \forall S \subset V \\ & S \neq \emptyset \\ & S \neq V \\ x_e \geq 0 \end{cases} \tag{3.28}$$

Observe that for any solution s of linear program 3.27, $s \cdot 4r/\gamma$ is a solution for linear program 3.28. If w is the minimum solution of linear program 3.27, then w' the minimum solution of linear program 3.28 is $w' = w \cdot (4r)/\gamma$. The MST relaxation of Equation 3.28 is known to have integrality gap at most 2 i.e $w' \geq \frac{1}{2} \cdot MST(G)$ [52]. Thus we have that

$$\text{OPT}^S \geq cost(P) \geq w = w' \cdot \frac{\gamma}{4r} \geq MST(G) \cdot \frac{\gamma}{8r}$$

Thus $(8r/\gamma) \cdot \text{OPT}^S \geq MST(G)$. As $8r/\gamma = o(\epsilon/\log L)$, the lemma is proved.

3.6 Derandomization

Arora's dissection can be derandomized by trying all choices for the shifts a and b. More efficient derandomizations are given in Czumaj and Lingas and in Rao and Smith [18, 45]. As for the randomized type assignment Algorithm 7, to guarantee that the cost of the dropped points is small, when selecting an interval Y to drop from a segment S, we only need to pick Y such that (1) $\text{Rad}(Y) \leq O(\epsilon/\log L)\text{Rad}(S)$ and (2)$\text{length}(Y) \leq O(\epsilon/\log L)\text{length}(S)$. In Lemma 3.5.1 and Property 3.5.8 we prove that these two conditions hold at the same time, in expectation when Y is chosen by first selecting a point uniformly from S and then selecting the next $|Y|-1$ consecutive points. To derandomize we can test the at most $|S|$ intervals of length $|Y|$ in S, (each starting from a different point in S), and select any interval that satisfies both conditions.

3.7 Extension to Higher Dimensions

Algorithm 6 extends to higher dimension $d \geq 2$ while d is a constant independent of n. Our algorithm uses two main methods: The first is a generalization of Arora's TSP algorithm to find the *black* tours and the second is the rounding scheme that keeps track of capacity constraints. As Arora's TSP algorithm extends to $d \geq 2$ dimension for constant d our generalization of this method also does the same. Our rounding scheme has no geometric dependency and does not change in higher dimensions. To show that the *red* tours have small cost we did use a geometric property that upper bounds the TSP of a set of points in a 2-dimensional box (i.e Theorem 3.5.10). However there exists generalization of that Theorem for $\mathcal{R}^d$. In fact Arora also uses Theorem 3.7.1 to generalize his patching lemma for $\mathcal{R}^d$.

Theorem 3.7.1. *[32] Let U be a finite set of points in the d-dimensional cube with side length L. There exists a constant c_d such that $tsp(S) \leq c_d \cdot L \cdot |U|^{d-1/d}$.*

As in Arora's TSP algorithm in d dimensions, each dissection box is now a d dimensional cubes with $2d$ boundaries (facets) and each box boundary is a $d-1$ dimensional cube. The boundary of a box contains $m = O(\sqrt{d}\log L)^{d-1}$ portals and each tour is $r = O((\sqrt{d}/\epsilon)^{d-1})$ light. The dissection tree will now contain $O(2^d n)$ non empty boxes. The running time of the QPTAS will be $O(2^d n \cdot n^{m^{O(2dr)}}) = n^{(\log L)^{O(d/\epsilon)^{d-1}}}$, since for every box in the dissection the DP will now guess the number of tours of each type that are present in the box, and there are $m^{O(2dr)}$ tour types. Thus for the running time of our approximation scheme to remain quasipolynomial we would need $(d)^{(d-1)}$ to be a constant.

Chapter 4

Extension to Multiple Depots

This chapter's results are joint work with Claire Mathieu and will be included in the journal version of [21].

4.1 Introduction

This chapter presents our QPTAS for the Euclidean UnitDem problem with a multiple depots, in constant dimensions.

Recall the problem: Given a positive integer k denoting the vehicle capacity, a set C of n customers and a set D of depots, such that C and D are points in $\mathcal{R}^d$ where d is a small constant independent of n, find a collection of tours of minimum total length covering all customers in C, such that each tour in the collection starts and ends at a depot and covers at most k customers.

The UnitDem problem with multiple depots models the setting where the delivery company has multiple warehouses. The multiple depot setting also has applications to the design of telecommunication networks where user nodes must be connected to one of many possible hubs using links with limited capacities [38].

All hardness results from the single depot setting extend to the multiple depot setting. Thus in the metric setting the problem is APX-Hard and admits no PTAS [6], but there is a 6-approximation due to Li and Simchi-Levi [38]. It remains an open question whether the Euclidean setting of the problem has a PTAS for all k. As in the single depot setting, Arora's PTAS for TSP can be extended into a PTAS when the vehicle capacity is large i.e $k = \Omega(n)$. Cardon et al. [14] extend the methods of Asano et al. [7] and Haimovich and Rinnooy Kan [29] to design a PTAS for the setting when the vehicle capacity and the number of depots is small i.e $k, |D| = O(\log n/\log\log n)$. We conjecture that we should also be able to use the QPTAS presented in this chapter and the methods of Adamaszek et al. [1] to get a PTAS for any number of depots and $k \leq 2^{\log^{\delta} n}$ (where δ a function of ϵ).

We extend our QPTAS from the single depot setting to handle multiple depots. As always we focus mainly on the setting where the customers are located in $\mathcal{R}^2$. Our algorithm can be extended to $\mathcal{R}^d$ for constant d as described in Section 3.7. We prove the following Theorem.

Theorem 4.1.1. *(Main Theorem) Algorithm 10 is a randomized quasi-polynomial time approximation scheme for the two dimensional Euclidean unit demand vehicle routing problem with multiple depots. Given $\epsilon > 0$, it outputs a solution with expected length $(1 + O(\epsilon))OPT$, in time $n^{\log^{O(1/\epsilon)} n}$. The Algorithm can be derandomized.*

Overview of our approach. Unlike the TSP problem and single depot setting, a solution for the multiple depots problem is not necessarily connected. Thus the farthest distance between two points, and the TSP of the points, are no longer lower bounds for OPT. Our algorithm and analysis from the single depot setting must be updated where ever these lower bounds were used.

We can no longer place our entire instance into a single bounding box. But we use a technique introduced by Borradaile et al. for the Steiner forest problem, where the solution is also disconnected, to partition our instance into sub-instances such that it suffices to solve each sub-instance independently.

Each sub-instance may still contain multiple depots however each can be solved similarly to the the single depot setting. For each sub-instance we perform a random dissection and place portals on the dissection lines. As before we use rounding to remember *approximately* the number of points covered by each tour segment. Then a quasipolynomial time dynamic program is used to find a *structured* solution, that is a solution consisting of *portal respecting and light* tours each covering approximately k customers. With multiple depots, the dynamic program's tour configurations describes whether a tour segment visits *some* depot rather than the depot. But this requires only a bit of information so the running time does not change. To get a solution which is feasible with respect to the vehicle capacity, we use the same randomized procedure as before to drop points from the tours so that each one covers at most k points. We use a different constant factor approximation, this time the 6-approximation of [38] on the dropped points to obtain solution for them.

The TSP is no longer a lower bound thus our analysis showing that it is cheap to cover the dropped red points must be updated. The output of the 6-approximation of [38] can be bounded in terms of the TSP of a slightly different instance (the *virtual instance*) as well as the Rad of the original instance. Thus we only need to extend our analysis from the single depot setting to show that the TSP of the virtual instance is small. Figure 4.1 shows a solution computed by Algorithm 10.

4.2 Preliminaries

For any two points $i, j \in I$ let $dist(i, j)$ denote the shortest distance between i, j. Recall that the radius of a customer $i \in C$ is defined as $r_i = \min_{d \in D} dist(i, d)$. For each customer $i \in C$ let $D(i)$ denote the closest depot to i. We define *virtual instance* the virtual instance of I denoted by $\tilde{I}$ as follows:

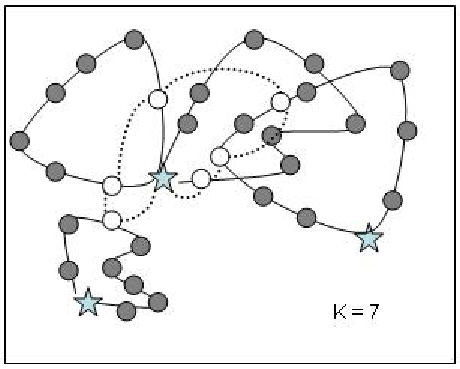

Figure 4.1: A solution computed by Algorithm 10 for $k = 7$. The stars are the depots, the solid circles are the "black" points and the empty circles are the "red" points. The solid tours are computed by the DP in step 2. Each covers $\leq k$ "black" points. The dotted tour covers the "red" points and is computed in step 5 using the 6-approximation.

Definition 4.2.1. *(Virtual Instance) For any input $I = \{C, D\}$ let the* virtual instance $\tilde{I}$*, contain all customers in C and a single virtual depot v. Define the distances of points in $\tilde{I}$ as follows: for any customer $i \in C$ the distance from i to virtual depot v is $\widetilde{dist}(i, v) = r_i$ and for any two customers $i, j \in C$ let $\widetilde{dist}(i, j) = \min\{r_i + r_j, dist(i, j)\}$, i.e the minimum of the distance from i to j in I and the distance from i to j going through v. For any two customers i, j if $\widetilde{dist}(i, j) \leq dist(i, j)$ the edge between i, j in $\tilde{I}$ is called a* virtual edge.

We use OPT($\tilde{I}$) to denote the optimal solution of the virtual instance. Note that OPT($\tilde{I}$) $\leq$ OPT as distance are in $\tilde{I}$ are at most the distances in I and that the distances in $\tilde{I}$ satisfy the triangle inequality. The virtual instance will is used by the constant factor algorithm and our in our analysis.

4.3 Algorithm and Proof of Main Theorem

We present a quasi-polynomial time approximation scheme for the problem with multiple depots and prove the main theorem. We partition the instance into sub-instances and solve each sub-instance independently and output the union of their solutions.

A sub-instance may contain multiple depots. Nevertheless it is solved very similarly to the single depot problem. The dynamic program needs to be slightly reinterpreted to handle multiple depots and we will use a different constant factor approximation algorithm for the red points.

In the following subsections we define the new constant factor approximation and give some straightforward extension of the definitions from the single depot case to prove the Main Theorem 4.1.1. The crux of the analysis is showing that the constant factor approximation on the red points still has negligible cost compared to opt.

Algorithm 10 QPTAS: Euclidean UnitDem, multiple depots, two dimensions

```
Input: Customers $C$, depots $D$, and integer $k$
1: Partition instance into sub-instances as described in Section 4.3.1.
2: for each sub-instance do
3:   Perturb sub-instance, perform random dissection and place portals as described in Section 4.3.2
4:   Use the DP from Section 4.4 to find a structured solution of Definition 3.2.4.
5:   Use the DP's history to construct the structured tours and assign types to points using the randomized type assignment as in the single depot case.
6:   Color points black and red as in the single depot case and drop all red points from the structured tours.
7:   Use the 6-approximation Algorithm 12 to get solution for dropped red points.
8:   The solution of the sub-instance is the union of red tours on the red points and black tours on the black points
9: end for
Output: Union of the solutions of all sub-instances.
```

4.3.1 Partitioning into Sub-Instances

Recall that in Arora's TSP algorithm, all input points are placed inside a bounding box of side length L. In problems like TSP and UnitDem with single depot where the solution is connected, the cost of the optimal solution at least is the distance between the two farthest points, so $L = O(\text{OPT})$. In the case of multiple depots, the bounding box of all points does not in general have length $O(\text{OPT})$. To overcome this difficulty we use a trick similar to one introduced by Borradaile et al. [12] for the Euclidean Steiner Forest problem, where this difficulty also arises. We partition the instance into sub-instances, using Algorithm 11, and prove in Lemma 4.3.1 that it suffices to solve each sub-instance independently to get a solution for the whole instance.

Algorithm 11 Partition into Sub Instances

```
Input: Customers $C$, depots $D$, and integer $k$
1: Run the 6-approximation, Algorithm 12, to get a solution of cost $A$.
2: Define a graph that has an edge between customers $c$, and $c'$ if and only if they are within distance $A$ of each other
3: Let $V_1, V_2, \dots V_x$ be the connected components of the resulting graph.
4: for each component $V_i$ for $i = 1, \dots x$ do
5:   for each customer $c$ in component $V_i$ do
6:     Let $D(c)$ be the set of depots that are within distance $A$ to customer $c$.
7:     Include $D(c)$ in component $V_i$.
8:   end for
9: end for
10: Let $Q_1, \dots Q_x$ be the resulting components.
Output: Sub instances $Q_1, \dots Q_x$.
```

Lemma 4.3.1. *Let $Q_1, Q_2, \dots Q_x$ be the sub-instances returned by Algorithm 11, $(n_i)_{i \leq x}$ denote the number of customers in Q_i, and let A denote the cost of 6-approximation computed in Line 1. Let*

L_i be the maximum distance between any two points in Q_i. We have that

$$1. \quad \sum_i OPT(Q_i) = OPT$$

$$2. \quad L_i \leq (n_i + 1)A$$

Proof. We prove the first property. For a contradiction suppose that a customer $c \in Q$ and $c' \in Q'$ where $Q \neq Q'$, are covered by the same tour in OPT. As c and c' are in different sub-instances they are also in different components in Line 3 of Algorithm 11 . Thus the distance between c and c' is greater than $A \leq 6$OPT, a contradition. A similar argument shows that a customer c in Q cannot be covered by a depot in which is in $Q' \neq Q$.

We prove the second property. Let A be the cost of the 6-approximation computed in Line 1 of Algorithm 11. Fix any Q_i and and let points p, p' be the points which are farthest apart in Q_i. If p, p' are both customers there is an Euclidean path from p, p' such that $dist(p, p') \leq (n_i - 1)A$. If both p, p' are depots, then $dist(p, p') \leq (n_i + 1)A$. If one point is a depot and the other a customer then $dist(p, p') \leq n_i A$. □

4.3.2 Preprocessing

The sub-instances $Q_1, \ldots Q_y$ are preprocessed independently using the Arora's technique with slightly different parameters.

Perturbation For any Q_i, define its bounding box as the smallest box whose side length L_i is a power of 2 that contains all points in Q_i. Let n_i be the number of customers in Q_i. Let A denote the cost of the 6-approximation computed in Line 1 of Algorithm 11. Place a grid of granularity $\delta_i = A\epsilon/(2 \cdot n_i \cdot 6)$ inside the bounding box. Move every input point (customers and depots) to the center of the grid box it lies in. Several points may map to the same grid box center, treat them as multiple points which are located in the same location. Scale distances in Q_i by $4/\delta_i$ so that all coordinates become integral and the minimum distance between any two grid centers that contain points is least 4. By Property two of Lemma 4.3.1, after scaling the maximum distance between points and hence the side length of the bounding box is $L_i = O(n_i^2/\epsilon)$.

A solution for the perturbed instance of Q_i can be extended into a solution for Q_i by taking detours from the grid centers to the original locations of the points. At most $2n_i$ detours will be required for Q_i; n_i detours for the customers and at most n_i detours for the depots as there are at most n_i tours. The cost of each detour (before scaling) is $2\delta_i$, thus the cost of all the detours for Q_i (before scaling) is at most $2n_i \cdot \delta$. Over all $Q_1, \ldots, Q_x$ the cost of detours is at most $\sum_i 2n_i\delta_i \leq A\epsilon/6$ which is $\leq \epsilon$OPT by Theorem 4.3.4. Scaling does not change the structure of the optimal solution and we can always re-scale to get the cost of the original instance. Thus the total cost of the perturbation is within the required ϵ error parameter.

Randomized Dissection and Portals. The randomized dissection, and placement of portals are done for each Q_i just as in Arora's TSP algorithm using L_i in place of L, and n_i in place of n. See Section 2.2.1.

4.3.3 Extending the Structure Corollary

We extend the definitions from Section 3.2.1 to a multiple depot problem. These extensions apply to a sub-instance and the whole instance.

Most extensions can be done by replacing the words "the depot" with "some depot". For example a k-tour, and each tour in a relaxed set, now must visit *some* depot rather than *the* depot. With these reinterpreted definitions, the Structure Theorem 3.2.6 also holds for multiple depots. Thus we get Corollary 4.3.2, corresponding to Corollary 3.2.7, which states that the structured set of k-tours that minimizes F is a near optimal solution.

Corollary 4.3.2. *(Structure Corollary) Let I be an instance (or a sub-instance) of UnitDem with multiple depots. Let OPT denote the length of the optimal solution of I, and let OPT^S denote the length of the structured solution of I that minimizes objective function F. In expectation over random shifts of the dissection, $OPT^S \leq (1 + O(\epsilon))OPT$.*

By Corollary 4.3.2 we can focus on computing the structured solution that minimizes objective F, which we will denote as OPT^S. In section 4.4 we show that the essentially the same DP as the single depot case can be used for the multiple depot setting, which proves the following:

Theorem 4.3.3. *(Dynamic Program) Let I be an instance (or sub-instance) of the UnitDem problem with multiple depots with n customers. Then $OPT^S(I)$ is computed by the dynamic program of Section 4.4 in time $n^{\log^{O(1/\epsilon)} n}$.*

4.3.4 A Constant Factor Approximation for Multiple Depots

The constant factor approximation we use was presented by Li and Simchi-Levi in 1990. The algorithm uses the tour partitioning 3-approximation (Algorithm 8) to get a solution for the virtual instance $\tilde{I}$, which is then translated into a solution for I. Using the solution of $\tilde{I}$ directly may result in paths (rather than tours) that start and end at different depots. Each such path is converted into a tour by taking an extra detour to one of its depots. Theorem 4.3.4 was proved by Li and Simchi-Levi and shows that Algorithm 12 is a 6-approximation.

Theorem 4.3.4. *[38] For any input I, an instance of the metric UnitDem problem with multiple depots, in expectation the output of Algorithm 12 has length most $Rad(\tilde{I}) + 2 \cdot TSP(\tilde{I}) \leq 3OPT(\tilde{I}) \leq 6OPT$.*

Proof. As the virtual instance $\tilde{I}$ is a single depot problem, by Theorem 3.2.9, Algorithm 8 returns a solution, $\tilde{\Pi}$, of expected length $\mathrm{Rad}(\tilde{I}) + 2\mathrm{TSP}(\tilde{I}) \leq 3\mathrm{OPT}(\tilde{I})$. Replacing the virtual edges of $\tilde{\Pi}$ does not increase its length, as each virtual edge $e = (i, j)$ of cost $r_i + r_j$ is replaced by two actual

Algorithm 12 Tour Partitioning for multiple depots 6-approximation [38]

```
Input: Customers $C$ and depots $D$ in metric space, and integer $k$
1: Let $\tilde{\Pi}$ denote the solution returned by Algorithm 8 on virtual instance $\tilde{I}$.
2: Let $P$ be a set to be populated with paths between two depots, with $\leq k$ customers
3: for each tour $\tilde{\pi}$ in $\tilde{\Pi}$ do
4:     Replace each virtual edge $e = (i, j)$ in $\tilde{\pi}$ between customers $i, j$ with edges $e_i = (i, D(i))$,
       $e_2 = (j, D(j))$ where $D(i)$ and $D(j)$ are the depots closest to $i, j$.
5:     Add the set of paths created in the previous step to $P$.
6: end for
7: for each path $p \in P$ do
8:     Let $i, j$ be the first and last customers in $p$ and $D(i), D(j)$ be their closest depots.
9:     if $r_i + dist(j, D(i)) \leq r_j + dist(i, D(j))$ then
10:        Turn $p$ into a tour $\pi$ that starts and ends at depot $D(i)$.
11:    else
12:        Turn $p$ into a tour $\pi$ that starts and ends at depot $D(j)$.
13:    end if
14: end for
Output the resulting tours $\Pi = (\pi)$.
```

edges of cost r_i and r_j. Thus the length of the paths in P is equal to length($\tilde{\Pi}$) $\leq 3\text{OPT}(\tilde{I})$. By the triangle inequality converting the paths in P into tours Π will at most double the length of each path in P, thus we have that the length of the output is at most $2\text{length}(P) \leq 6\text{OPT}(\tilde{I})$. The Theorem follows as $\text{OPT}(\tilde{I}) \leq \text{OPT}$. □

4.3.5 Proof of Main Theorem 4.1.1

We first prove that Algorithm 10 obtains solutions of cost $(1 + O(\epsilon))\text{OPT}(Q_i)$ for each sub-instance Q_i. Fix a sub-instance Q_i. To show that a solution of cost $(1+O(\epsilon))\text{OPT}(Q_i)$ is obtained, follow the proof of Theorem 3.1.1, replacing Corollary 3.2.7 with Corollary 4.3.2, Theorem 3.2.8 with Theorem 4.3.3 and Theorem 3.2.10 with Theorem 4.3.5.

Theorem 4.3.5. *Let I be an instance (or sub-instance) of the UnitDem problem with multiple depots and let OPT denote the optimal solution of I. In expectation over the random shifts of the dissection and the random type assignment, the length of the red tours for I output by Algorithm 10 is $O(\epsilon)OPT$.*

We use the first property of Lemma 4.3.1 to show that the union of the solutions of all sub-instances gives a solution of cost $(1 + O(\epsilon))\text{OPT}$ for the whole instance.

The DP for each sub-instance dominates the running time. Let n_i denote the number of customers in Q_i. The run time is at most,

$$\sum_i^y n_i^{\log^{O(1/\epsilon)} n_i} \leq n^{\log^{O(1/\epsilon)} n}$$

The derandomization of the Algorithm can be done by derandomizing the procedure for all sub-instances individually as discussed in Section 3.6.

4.4 Extending the Dynamic Program

A slight modification to the configurations of the single depot DP allows it to also handle multiple depots. The main insight is that for each tour segment we only need to remember whether the segment visits some depot or not.

Recall that in the single depot DP of Section 3.4, a rounded configuration represents segments by numbers $r_{p,q,t,d}$ and an unrounded configuration represent them by tuples (p, q, m, d). In both cases p, q are portals, t and m represent the (approximate) number of points on the segment and d indicates whether the segment visited the depot. For the multiple depot setting the only modification to the DP is to update the meaning of indicator d. Now $d = 1$ will represent that the segment visited *a* depot, rather than the depot.

Recall that a configuration profile $\Phi = (p_1, p_2, x_1, d_1), (p_2, p_3, x_2, d_2), \dots (p_s, p_{s+1}, x_s, d_s)$ with $p_1 = p_{s+1}$ indicates that the segments form a tour. In the single depot case such a Φ is feasible if at least one segments had $d_i = 1$ indicating that it visited the depot. In the multiple depot setting the same condition implies that at least some depot was visited, and so the profile is still feasible. If multiple depots were vistied we can shortcut around all but one of the depots in the final solution.

4.5 Proof of Theorem 4.3.5

Let R denote the points marked red by Algorithm 10. By Theorem 4.3.4 the 6-approximation of R has cost at most $\text{Rad}(\tilde{R}) + 2 \cdot \text{TSP}(\tilde{R})$, where $\tilde{R}$ is the virtual instance with the points in R and all the depots. The proof of Theorem 4.3.5 follows from Lemmas 4.5.1 and 4.5.2 which show that in expectation both quantities are $O(\epsilon)\text{OPT}$.

Lemma 4.5.1. *For any instance I with multiple depots, let R denote the points of I colored red by Algorithm 10 and $\tilde{R}$ denote the virtual instance with the points in R and depots in I. In expectation over the random type assignment, $Rad(\tilde{R}) = O(\epsilon)OPT(I)$*

Proof. The proof of the corresponding lemma for the single depot case (Lemma 3.5.1) implies that $\text{Rad}(\tilde{R}) = O(\epsilon)\text{Rad}(\tilde{I})$. As the Rad bound sums the distance of each point to its closest depot, $\text{Rad}(\tilde{I}) = \text{Rad}(I)$. The Lemma follows as $\text{Rad}(I) \le \text{OPT}(I)$. □

Lemma 4.5.2. *For any instance I with multiple depots, let R denote the points of I colored red by Algorithm 10 and $\tilde{R}$ denote the virtual instance with the points in R and depots of I. In expectation over the random dissection and type assignment $TSP(\tilde{R}) = O(\epsilon)OPT(I)$*

4.5.1 Proof of Lemma 4.5.2

For each level $\ell \in [0, \ell_{\max}]$ let R_ℓ denote the type ℓ points of I and note that $R = \cup_{\ell>0} R_\ell$. Let $\tilde{R}_\ell$ denote the virtual instance on the set of customers in R_ℓ and all the depots in the instance. We will prove that for each level ℓ, $E[\text{TSP}(\tilde{R}_\ell)] \le O(\epsilon/\log L)\text{OPT}$. This implies the lemma as the tours of $\tilde{R}_\ell$ from all $O(\log L)$ levels can be pasted together at virtual depot v to yield a tour of $\tilde{R}$.

We follow the proof of Lemma 3.5.2 and upper bound the cost of TSP($\tilde{R}_\ell$) by thinking of the tour of $\tilde{R}_\ell$ in three parts: *inside, boundary* and *outside.* The *inside* and *boundary* parts are defined just as in the proof Lemma 3.5.2, using distances of the actual instance rather than the virtual instance. This still gives an upper bound on the cost of the virtual TSP as distances in the virtual instance are always at most the distances in the actual instance. Given the *inside* and *boundary* to get a tour of $\tilde{R}_\ell$ we only need a tour of the boxes of B_ℓ and the virtual depot. We refer to this as the *outside* part and we will use virtual distances for this part.

Define a complete graph $\tilde{G} = (V, E)$, which corresponds to the G in the proof of Lemma 3.5.2, but uses virtual distances. $\tilde{G}$ contains a vertex for each box in B_ℓ and one additional vertex to represent the virtual depot. The edge costs in $\tilde{G}$ are defined as follows: the cost of the edge between two box vertices b, b' is equal to the length of the shortest path in $\tilde{I}$ from any portal of b to any portal of b', the cost of an edge between a box vertex b and the virtual depot vertex is equal to the length of the shortest path in $\tilde{I}$ from any portal of b to the virtual depot. By Lemma 1.3.6 the *outside* cost; i.e the cost of a tour of the virtual depot and boxes B_ℓ, is at most $2MST(\tilde{G})$. Thus we have that:

$$TSP(\tilde{R}_\ell \cup v) = \sum_{b \in B_\ell} (\text{TSP of } R_b \text{ connected to boundary of } b) + (\text{boundary of } b) + 2MST(\tilde{G}) \tag{4.1}$$

The first two terms (i.e the *inside* and *boundary* costs) are defined exactly as in Lemma 3.5.2 using the actual distances, thus they can be analyzed using Lemmas 3.5.4 and 3.5.5 and shown to be $O(\epsilon/\log L)F(\text{OPT}^S)$. By Corollary 4.3.2 $F(\text{OPT}^S) \leq (1 + O(\epsilon))\text{OPT}$, thus we have that,

$$\sum_{b \in B_\ell} (\text{TSP of } R_b \text{ connected to boundary of } b) + (\text{boundary of } b) \leq O(\epsilon/\log L)\text{OPT}$$

By Lemma 4.5.3 (below) we have that $MST(\tilde{G}) \leq O(\epsilon/\log L)\text{OPT}^S$. By definition of F that is at most $O(\epsilon/\log L)F(\text{OPT}^S)$, which by Corollary 4.3.2 is at most $O(\epsilon/\log L)\text{OPT}$. Thus we have shown that TSP($\tilde{R}_\ell \cup v$) $\leq O(\epsilon/\log L)$OPT overall.

Lemma 4.5.3. *(Outside cost) In expectation over the random shifted dissection,* $E[2MST(\tilde{G})] \leq O(\epsilon/\log L)OPT$.

Proof. The proof follows by replacing G with $\tilde{G}$ in the proof of Lemma 3.5.6 and interpreting OPT^S as the structured solution for the multiple depot problem. □

Chapter 5

Vehicle Routing: Unsplittable Demands, Single Depot in One Dimension

This chapter's results are joint work with Shay Mozes and Claire Mathieu and have appeared in [20].

5.1 Introduction

This chapter presents our APTAS for 1-dimensional Unsplit VRP with a single depot.

Recall the problem: Given a positive integer k denoting the vehicle capacity, a set $C = \{(p_i, w_i)\}_{i \leq n}$ of n customers each with a position p_i on the line and a demand $w_i \leq k$, and a single depot also with a position on the line, find a collection of tours of minimum total length covering the demands of all customers in C, such that each tour starts and ends at the depot, delivers at most k demand and such that no customer's demand is split up among multiple tours.

Problem applications. As previously mentioned, from the VRP perspective the Unsplit problem models the case where all customers require their entire demand in one delivery, e.g residential customers ordering take out, or groceries. The 1-dimensional setting may be a suitable model for train routing, where a train delivers cargo from a harbor to stations located along the railroad.

The problem can be equivalently viewed as a scheduling problem of minimizing the makespan on a single batch machine with non-identical job sizes or a generalization of bin-packing.

In the scheduling setting, integrated circuits are tested by subjecting them to thermal stress for an extended period of time (burn-in). Each circuit has a prespecified burn-in time (p_i) and a number of boards (w_i) it requires. Since circuits may stay in the oven for a period longer than their burn-in time, it is possible to place different circuits as a batch in the oven simultaneously as long as the capacity of the oven (the number of boards in the oven) k is not exceeded. The processing time of each batch is the longest burn-in time required among the circuits in the batch. Once processing is begun on a batch, no product can be removed from the oven until the processing of the batch is complete. The goal is to find a partition of the circuits into batches so that the total processing time of all batches (the makespan) is minimized.

In the generalized bin packing setting, we are given a set C of n items of sizes $w_1, \ldots, w_n$ and costs $p_1, \ldots, p_n$. The goal is to find the minimum cost packing of items into k sized bins where the cost of a bin is the maximum cost item in the bin. As an application of the generalized problem, consider packing temperature sensitive products into k-sized storage bins where each product has a size and a maximal temperature at which it may be stored safely. The lower the temperature the higher the cost to store the product. Packing all the products, so that none are damaged while keeping the cost of operations as low as possible requires solving the generalized bin packing problem.

Hardness. The classical bin-packing problem reduces to the Unsplit problem (for any dimensions and with any number of depots) by setting all p_i equal. It is well known [27] that bin-packing is strongly NP-hard and does not have a polynomial time approximation algorithm with approximation ratio better than 3/2 unless P=NP, hence no PTAS is possible. Thus this also applies to the Unsplit problem, even in 1-dimension. The hardness results however, do not exclude APTAS (asymptotic PTAS), and in fact these exist for bin packing when the cost of the optimal solution is at least $1/\epsilon$ (that is, at least $1/\epsilon$ bins are necessary).

Our result. We design an APTAS for the 1-dimensional Unsplit single depot problem. The problem does not admit an asymptotic approximation scheme in the usual sense. The reason is that the cost of the solution is determined by the positions p_i, so any instance can be scaled so that the cost of an optimal solution is arbitrarily large without changing the solution itself. Therefore, to define a notion of asymptotic for our problem we restrict the ratio of the optimal solution and the maximal position. In other words, scale the input so that $\max_i p_i = 1$. The asymptotic regime for our algorithm occurs when the cost of the scaled input is $\Omega(1/\epsilon^6)$.

Theorem 5.1.1. *For any instance of the* 1*-dimensional Unsplit single depot problem such that* $\max_i p_i = O(\epsilon^6)OPT$*, Algorithm 13 outputs a solution of cost* $(1 + O(\epsilon))OPT$ *in time* $O(n \log(n)) + \log(n)(1/\epsilon)^{O(1/\epsilon)}$.

Related work. We are not aware of any prior work from the VRP community that specifically considers the 1-dimensional setting of the Unsplit problem. For the metric setting, Haimovich, Rinnooy Kan, and Stougie give a 3.5-factor approximation [30]. Bramel et al. give a probabilistic analysis for the Euclidean plane where customer demands are drawn i.i.d from any distribution [13]. Labbé et al. [36] give a 2-approximation for the problem on a tree.

The problem in its formulation as minimizing the makespan on a single batch machine with non-identical job sizes has been extensively studied in the past two decades, and the result of this chapter gives the first APTAS for the most general setting of this problem. Previously, Zhang and

Cao [56] designed an APTAS for the symmetric setting where $p_i = w_i$ for all i, which we compare our result to below. Uzsoy [51] was the first to consider the problem of minimizing the makespan on a single batch machine. He proved that it is NP-Hard and presented a few heuristics that were evaluated empirically. Many others have considered the problem since and various heuristics are given in [8, 23, 24, 43] to name just a few. In terms of approximation algorithms, Zhang et al. [55] prove constant approximation ratios for some heuristics, with 7/4 being the best.

Comparing our result to the APTAS of Zhang and Cao [56], our algorithm does not require the assumption that $p_i = w_i$ for all i as in [56]. Our algorithm also has a much better running time than the APTAS of [56]; $O(n \log) + \log n \cdot (1/\epsilon)^{O(1/\epsilon)}$ versus $n^{(1/\epsilon)^{O(1/\epsilon)}}$. However our speedup in running time comes at the price of having a more severe asymptotic assumption. An alternative version of our algorithm which uses enumeration (rather than an LP) as in the work of Zhang and Cao [56], has the less severe asymptotic assumption, $\max_i p_i = O(\epsilon)\mathrm{OPT}$, but runs in time $n^{(1/\epsilon)^{O(1/\epsilon)}}$.

The bin packing problem has a very long history and many variants of the problem have been studied (see [16] for a survey). However none of these variants capture the generalization we describe above. APTAS and AFPTAS have been designed for classical bin packing as well as some of its variants. Fernandez de la Vega and Lueker [26] designed a PTAS for classical bin packing in 1981. In the subsequent year, Karmarkar and Karp [33] extended their framework and gave an AFPTAS. Variants of bin packing which have APTASs or AFPTAs include bin packing with variable sized bins [41] and generalized cost structure [25], among others. A variant which was shown to be APX-Hard (i.e does not admit APTAS) is multi dimensional bin packing [10].

Overview of our approach. Our techniques draw on those used in the literature for the bin-packing problem, namely rounding demands and separate handling of small and big demands. We simplify our instance by rounding the positions of all customers and then apply the De La Vega and Lueker [26] rounding scheme from bin packing on the big demands at each position, yielding a small number of distinct big demands at each position. Next we try a few different ways ($O(1/\epsilon)$) to partition the customers into disjoint regions. For each possible partition, we solve the problem in each region independently and combine to get a solution for the whole instance. In the end we output the solution of the partition with the minimum cost. Figure 5.1 shows a solution computed by our algorithm.

To solve the problem inside each region of a partition, we using solve a *relaxed* problem, where we pretend that the small demands are fluid and can be split arbitrarily among different tours. We find a near optimal solution for the relaxed problem by rounding the solution of a linear program as was done in [25, 17]. Finally we extend the relaxed solution greedily into a feasible solution that respects the unsplittable constraint for small demands.

Due to rounding each region contains only a constant number of positions and each position has a constant number of distinct big demands. This allows us to solve the relaxed problem in each region in constant time. Our running time follows as there are a logarithmic number of regions in each partition.

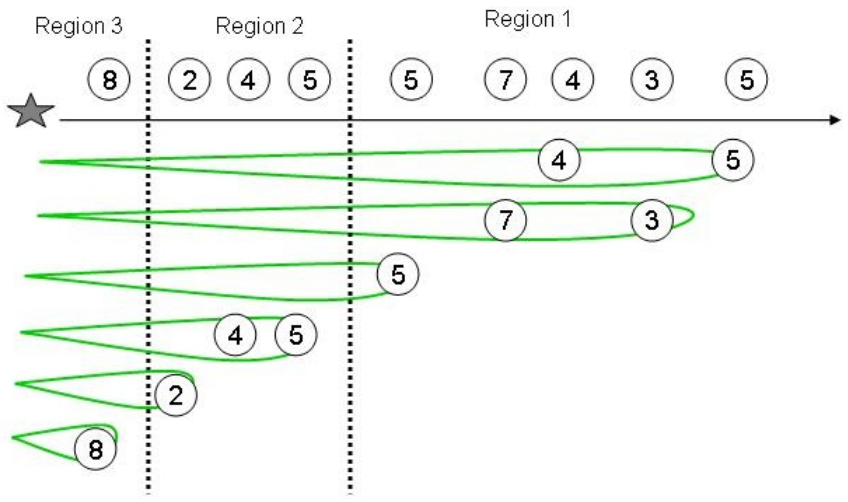

Figure 5.1: A solution computed by Algorithm 13 for $k = 10$. The star is the depot, the circles on the line are the customers with demands. The dashed lines partition the instance into regions. All tours (shown below the line) cover customers from only one region and at most k total demand.

For the analysis we use a shifting technique as in [9, 1] to show that if we try a few ($O(1/\epsilon)$) different ways to partition the instance into regions at least one yields a near-optimal solution. Finally we show that it is possible to construct a near-optimal solution by greedily inserting the small demands into the relaxed solution we compute.

5.2 Preliminaries

Without loss of generality we assume that our input is preprocessed as defined below:

Definition 5.2.1. *(Preprocessed) An instance is preprocessed if:*

- *The depot's position is the origin.*
- *All customers are located to the right of the depot.*
- *No customer is closer than $(\epsilon \cdot p_{\max})/n$ from the depot, where $p_{\max} = \max_i p_i$.*

We can always define the depot's position to be the origin and reinterpret the positions of the customers such that the position of each customer also represents its distance to the depot. If there are customers on the right and left of the depot, we can solve each side separately, as they are analogous to one another, and return the union of the two solutions. We denote $p_{\max}$ as the furthest position from the depot. If there are any customers located at positions closer than $\epsilon \cdot p_{\max}/n$ from the depot, we can serve them each with a separate tour. The overall cost for this is at most $2\epsilon p_{\max}$, which is at most ϵOPT as any solution has cost at least $2p_{\max}$.

Observe that if p is the maximum position of any customer on tour t then the cost of t is $2p$.

5.3 Algorithm and Proof of Main Theorem

We present the APTAS in Algorithm 13 and prove Theorem 5.1.1. Figure 5.1 shows a solution computed by our algorithm.

Algorithm 13 APTAS for Unsplit with single depot in one dimension

Input: A preprocessed instance with customres $(p_i, w_i)_{1\leq i\leq n}$ and vehicle capacity k
Precondition: $\max_i p_i \leq \epsilon^6 \text{OPT}$

```
1: Round the input using Algorithm 14 described in section 5.3.1.
2: for $1 \leq j \leq 1/\epsilon$ do
3:     Partition the instance into regions $R_1, R_2, \ldots$ using partition $P_j$ (as in Definition 5.3.2).
4:     for each non-empty region $R_i$ of partition $P_j$ do
5:         Solve the relaxed problem in $R_i$ treating small demands as fluid using Algorithm 15.
6:         Extend the relaxed solution into a feasible solution, $Sol_i$, for small demands in $R_i$ using
           Algorithm 16.
7:     end for
8:     Let $\text{Best}(P_j) = \cup_{R_i \in P_j} Sol_i$ be the solution found using partition $P_j$.
9: end for
```

Output: $\min_j \text{Best}(P_j)$, the minimum cost solution found over all partitions.

5.3.1 Rounding

Algorithm 14 uses rounding to reduce the number of positions and the number of distinct big demand sizes at each position. Due to preprocessing, the position of each customer represents its distance from the depot. Thus in Line 1 rounding a position p_i, means to move customer i to a new position so that its distance from the depot is the smallest integer power of $(1+\epsilon)$ that is at least p_i. A demand w_i is called *big* if $w_i \geq \epsilon k$ and small otherwise. Lines 3-8 uses the rounding technique of bin packing algorithms due to Fernandez de la Vega and Lueker [26] to reduce the number of distinct big demands at each position.

By Lemma 5.3.1 an optimal solution of the rounded instance I' can be extended into a near optimal solution for I. Thus we can focus on computing a solution for the rounded instance I'. See Section 5.4 for the proof of the lemma.

Lemma 5.3.1. *Given a preprocessed instance I, Algorithm 14 outputs an instance I' such that:*

1. *Each customer's position is at distance $(1+\epsilon)^j$ for some integer j, from the depot.*
2. *Each position has at most $1/\epsilon^2 + 1$ distinct sizes of big demands.*
3. *Any feasible solution for I' is also feasible for I.*
4. *$OPT(I') \leq (1 + O(\epsilon))OPT(I)$.*

Algorithm 14 Rounding

```
Input: Preprocessed instance $I$ with vehicle capacity $k$, customers $(p_i, w_i)_{1\le i\le n}$
1: Round up each customer's position, $p_i$ (i.e its distance to the depot), to the next integer power
   of $(1+\epsilon)$
2: Partition demands $(w_i)_i$ into big ($\ge k\epsilon$) and small ($< k\epsilon$).
   Rounding big demands:
3: for each position $p$ with $n_p$ number of big demands s.t $n_p \ge 1/\epsilon^2$ do
4:   Consider the $n_p$ big demands in decreasing order and partition them into $\lceil 1/\epsilon^2 \rceil$ groups s.t
     each group (except possibly one) has cardinality exactly $\lfloor n_p \epsilon^2 \rfloor$.
5:   for each group $g$ do
6:     Round the size of all demands in $g$ to the size of the maximum demand in $g$.
7:   end for
8: end for
Output: Rounded instance $I'$.
```

5.3.2 Partitioning into Regions

We define $1/\epsilon$ ways to partition the instance into disjoint *regions* in Definition 5.3.2. Lemma 5.3.3 states one of the main structural properties, that for at least one of these partitions, solving each region independently and combining the solutions yields a near optimal solution for the whole instance.

Definition 5.3.2. *(Partitions) Let I' be a rounded instance of the problem where $p_{\max} = \max_i p_i$ is the farthest position from the depot. Partitions are defined in terms of* blocks *and* regions*.*

A block*, is described by an integer $i \ge 0$ where the i-th block consists of the customers at positions $(p_{\max}\epsilon^{i+1}, p_{\max}\epsilon^{i}]$. A* region *consists of at most $1/\epsilon$ consecutive blocks.*

We define $1/\epsilon$ ways to partition I' into regions. For each $0 \le j < 1/\epsilon$, in partition *P_j, the farthest region (from the depot) consists of the customers at positions $(\epsilon^j p_{\max}, p_{\max}]$, and for all $i \ge 0$, the i-th farthest region consists of the customers at positions $(\epsilon^j p_{\max}\epsilon^{(i+1)/\epsilon}, \epsilon^j p_{\max}\epsilon^{i/\epsilon}]$.*

Observe that in partition P_j the farthest region consists of j blocks, and for all $i \ge 0$ the i-th farthest region consists of $1/\epsilon$ blocks.

Lemma 5.3.3 allows us to reduce the problem to solving each region of I' separately. Its proof uses a simple structural property about the tours, and a *shifting* technique similar to that of Baker [9] and Hochbaum and Maass [31]. See Section 5.5.

Lemma 5.3.3. *Let I' be a rounded instance of the problem and let P_j, for $0 \le j < 1/\epsilon$, be the partition of I' into regions. There exists a partition P_j, such that $\sum_{R_i \in P_j} OPT(R_i) \le (1 + O(\epsilon))OPT(I')$.*

5.3.3 Solving a Region

Using Lemma 5.3.3 we focus on solving the problem in each region. We define a *relaxed* problem (Definition 5.3.4) that can be solved in constant time. Then we extend the relaxed solution into a feasible solution for all demands.

Definition 5.3.4. *(Relaxed problem) Fix a region R of the rounded instance I'. In the* relaxed problem*, the small demands of R are treated as fluid and allowed to be split up among multiple tours and placed fractionally on tours.*

Solving the relaxed problem. We describe how to handle the (solid) big demands for solving the relaxed problem. As a result of rounding, each region R contains just a constant number of positions and each position contains a constant number of distinct big demands. Thus the total number of distinct big demands in R is a constant. The big demands in R can be described concisely as:

Definition 5.3.5. *(Big demand type) Fix a region R of the rounded instance I'. A big demand* type *is a pair (p, b) where p is the position of a big demand and b is one of the at most $1/\epsilon^2$ (rounded) big demand sizes at position p. Let $n(d)$ denote the total number of demands of type $d = (p, b)$ in R.*

A tour can cover only a constant number (at most $1/\epsilon$) of big demands. Thus we can describe tour of R concisely by a multiset of big demand types. The *configuration* of a tour roughly describes which demands it will cover: for each occurrence of a big demand type (p, b) in its multiset the tour covers one of the big demands from position p with size b.

Definition 5.3.6. *(Configuration) Fix a region R of rounded instance I'. A* configuration *f of a tour in R consists of a position m_f, which is the furthest position of the tour, and a multiset D_f of big demand types, each with position at most m_f, whose (rounded) sizes sum up to at most k.*

For each configuration f, let c_f denote the remaining capacity of a tour with configuration f (i.e., $c_f = k - \sum_{(p,b)\in D_f} b$). For any big demand type d, let $n_f(d)$ denote the multiplicity of d in D_f.

Given the set of configurations $\mathcal{F}$ of region R a linear program is used to solve the relaxed problem. Let S be the set of small demands in region R. The linear program will consider the small demand as fluid at each position in R. The linear program has one variable x_f for each tour configuration $f \in \mathcal{F}$ and the objective is to select a minimum cost set of tour configurations such that two constraints are satisfied: Constraint 5.2 ensures that all big demand types are covered by the selected tour configurations and constraint 5.3 ensures that for each position p, the small demands further right than p can be covered with the remaining capacities of the tour configurations that extend to the right of p.

$$
\begin{aligned}
\min \quad & \sum_{f\in\mathcal{F}} 2\cdot m_f x_f && && (5.1)\\
s.t \quad & \sum_{f\in\mathcal{F}} x_f n_f(d) \ge n(d) && \forall \text{ demand types } d && (5.2)\\
& \sum_{f:m_f\ge p} c_f x_f \ge \sum_{\substack{(p_i,w_i)\in S\\ p_i\ge p}} w_i && \forall \text{ positions } p && (5.3)\\
& x_f \ge 0 && && (5.4)
\end{aligned}
$$

Algorithm 15 rounds a basic solution of the linear program above to obtain a solution to the relaxed problem. Let OPT(R) denote the optimal (unrelated) solution of region R.

```
Algorithm 15 Solve Relaxed Region
Input: A region $R$ with a set $S$ of small demands.
1: Let $\mathcal{D}$ be the set of big demand types for region $R$ (Definition 5.3.5).
2: Let $\mathcal{F}$ be the set of tour configurations for region $R$ (Definition 5.3.6).
3: Let $x^* = (x^*_f)_{f\in\mathcal{F}}$ denote a basic optimal solution of the linear program of (Equations 5.1-5.4).
4: Let $\bar{x}_f = \lceil x^*_f \rceil$ for each $f \in \mathcal{F}$.
5: Cover the big demand types in $\mathcal{D}$ with tours specified by the $(\bar{x}_f)_{f\in\mathcal{F}}$. 1
Output: The resulting set of tours covering $\mathcal{D}$.
```

By Lemma 5.3.7, Algorithm 15 returns in constant time a solution covering all big demands in R, whose cost is bounded in terms of OPT(R) and p_R, where p_R is the farthest postion in R. Note that later we will use the "asymptotic" assumption to show that the additive p_R term is small compared to OPT. The proof appears in Section 5.6.

Lemma 5.3.7. *Let R be a region of rounded instance I', p_R denote the farthest position in R and $OPT(R)$ denote the cost of the optimal (unrelaxed) solution of R. Given R, Algorithm 15 outputs in time $(1/\epsilon)^{O(1/\epsilon)}$ a set of tours T covering all big demands in R such that $cost(T) \le OPT(R) + p_R \cdot ((1/\epsilon)^2 \log(1/\epsilon))(2 + 1/\epsilon^2)$.*

Extending a relaxed solution. Let $T = (m_t, c_t)_t$ denote the list of tours output by Algorithm 15 where m_t is the furthest position and c_t denotes the remaining capacity of tour t after it has covered the big demands. Algorithm 16 takes the list of tours $(m_t, c_t)_t$ as input and greedily extends them to cover the small demands of R in a feasible way (i.e., without splitting any of them).

By Lemma 5.3.8 the output of Algorithm 16 is bounded by the cost of the relaxed tours T and p_R. See Section 5.7 for the proof.

Lemma 5.3.8. *Let G be tours output by Algorithm 16 on input $T = (m_t, c_t)_t$. Then $cost(G) \le (1 + 2\epsilon)cost(T) + 2p_R$.*

Algorithm 16 Greedy Extension

```
Input: Small demands $(p_i, w_i)_i$ and a list $T$ of tours $(m_t, c_t)_t$, where $m_t$ is the furthest position of tour $t$ and $c_t$ is its remaining capacity.
1: for each small demand $(p_i, w_i)$ in order of decreasing $p_i$ do
2:    if there is a tour $t \in T$ with $m_t \geq p_i$ and $c_t \geq w_i$ then
3:       cover $(p_i, w_i)$ with $t$ and set $c_t := c_t - w_i$
4:    else
5:       add a new tour $t$ with $c_t = k$ and $m_t = p_i$
6:       cover $(p_i, w_i)$ with $t$ and set $c_t := c_t - w_i$
7:    end if
8: end for
Output: the resulting tours.
```

5.3.4 Proof of Main Theorem 5.1.1

Correctness of Algorithm 13. By Lemma 5.3.1 the optimal solution of the rounded instance is a near optimal solution for the original instance. To solve the rounded instance Algorithm 13 tries all $1/\epsilon$ ways to partition it into regions. Lemma 5.3.3 shows that for at least one of these partitions, P^*, a near optimal solution is obtained by solving in each region independently and combining the solutions. For the rest of the analysis, focus on the execution of Algorithm 13 that uses partition P^*. Let $R_1^*, R_2^*, \ldots, R_r^*$ be the regions of P^*. We show below that the cost of the solution found for each R_i^* can be bounded as follows,

$$\text{cost of our solution of } R_i^* \leq (1+2\epsilon)\text{OPT}(R_i^*) + \ell(\epsilon)p_{R_i^*}$$
$$\text{where } \ell(\epsilon) = (1+2\epsilon)(1/\epsilon)^2 \log(1/\epsilon)(2+1/\epsilon^2) + 2 = O(1/\epsilon^5). \tag{5.5}$$

Assuming Equation 5.5 for now, by Lemma 5.3.3 the output of Algorithm 13 has cost

$$\sum_{i \leq r} (1+2\epsilon)\text{OPT}(R_i^*) + \ell(\epsilon)p_{R_i^*} = (1+O(\epsilon))\text{OPT} + \ell(\epsilon)\sum_{i \leq r} p_{R_i^*}.$$

By definition of regions, for each i, $p_{R_i^*} \leq p_{\max}\epsilon^{i/\epsilon}$. Thus, $\ell(\epsilon)\sum_{i \leq r} p_{R_i^*} \leq \ell(\epsilon)\sum_{i \geq 0} p_{\max}\epsilon^{i/\epsilon} \leq 2\ell(\epsilon)p_{\max}$, and so the cost of the solution is at most

$$(1+O(\epsilon))\text{OPT} + 2\ell(\epsilon)p_{\max} \tag{5.6}$$

As $\max_i p_i \leq O(\epsilon^6)\text{OPT}$, the additive cost incurred is $O(\epsilon)\text{OPT}$ which is within the desired approximation factor.

Now we prove Equation 5.5 holds. Consider a region R_i^* of P^*. Algorithm 15 is used to find a set of tours T that covers all big demands in R_i^* and all the fluid small demands. Given T, Algorithm 16 produces a solution that covers the small demands feasibly (not as fluid) and by Lemma 5.3.8 the cost of the resulting solution for R_i^* is at most $(1+2\epsilon)\text{cost}(T) + 2p_{R_i^*}$, where $p_{R_i^*}$ is the maximum position in R_i^*. Using Lemma 5.3.7 to bound the $cost(T)$ shows Equation 5.5.

Running Time of Algorithm 13. Rounding positions, rounding the big demands at each position, and partitioning the instance into regions can all be done in time $O(n\log(n))$. By preprocessing

Definition 5.2.1, no customer is located closer than $\epsilon \cdot p_{\max}/n$ from the depot. Thus by Definition 5.3.2 there are at most $O(\log n/\log(1/\epsilon))$ non-empty blocks containing customers. As stated in the observation after Definition 5.3.2 each partition P_j, partitions the instance into regions, where all but at most one region consists of $1/\epsilon$ blocks. Thus each partition P_j has $O(\epsilon \log(n)/\log(1/\epsilon))$ regions.

For a given partition of the instance we solve the relaxed problem and compute the greedy extension in each region. By Lemma 5.3.7, Algorithm 15 solves the relaxed problem in time $(1/\epsilon)^{O(1/\epsilon)}$ and by inspection one can see that the greedy extension can be computed in at most $O(n)$. As Algorithm 13 computes a solution for each of the $1/\epsilon$ possible partitions, the final run time of Algorithm 13 is $O(n\log(n)) + \log(n) \cdot (1/\epsilon)^{O(1/\epsilon)}$.

5.4 Proof of Lemma 5.3.1

The first property follows from the Line 1 of Algorithm 14 where each position is rounded up to the next power of $(1+\epsilon)$. The second property follows from Lines 3-8 of Algorithm 14. Consider a position p with more than $1/\epsilon^2$ big demands. In Line 4, the demands are partitioned into $\lceil 1/\epsilon^2 \rceil$ groups and in Lines 5-6, for each group, all demands in the group are rounded up to the maximum demand size in the group. Thus afterward, position p has at most $1/\epsilon^2 + 1$ distinct sizes of big demands. In I' the positions of customers are the same or further from the depot and the sizes of demands are the same or larger than in I. Thus any solution that covers all customers in I' is also feasible for I, hence the third property holds.

We focus on the last property and analyze rounding positions. Let I_1 denote the instance obtained from I after rounding the positions (Line 1). Any length d tour in $\mathrm{OPT}(I)$ can be transformed into a feasible tour for I_1 by extending its length by at most ϵd, so

$$\mathrm{OPT}(I_1) \leq (1+\epsilon)\mathrm{OPT}(I). \tag{5.7}$$

Next we analyze rounding demands, by carrying out the de la Vega and Lueker bin packing analysis at each position [26]. Let R be the set of positions where big demands were rounded (i.e the position of I_1 with more than $1/\epsilon^2$ big demands). Let $\overline{I}$ be the instance obtained by partitioning the demands at each position in R into $1/\epsilon^2$ equal sized groups and rounding *up* the demands of each group to the maximum demand of the group (i.e after Line 8). Let $\underline{I}$ be the instance that would be obtained if instead, the demands of each group were rounded *down* to the size of the maximum demand of the next lower group (See Figure 5.2). Clearly as demands in $\overline{I}$ have sizes same or larger than in $\underline{I}$,

$$\mathrm{OPT}(\underline{I}) \leq \mathrm{OPT}(I_1) \tag{5.8}$$

Let n_p be the number of big demands at position $p \in R$. There are at most $\lfloor n_p\epsilon^2 \rfloor$ big demands in all groups in p. Observe that a covering of the demands at p in $\underline{I}$ yields a covering of all but the at most $\lfloor n_p\epsilon^2 \rfloor$ largest demands at p in $\overline{I}$ (See Figure 5.2). Using a single tour to cover each of those

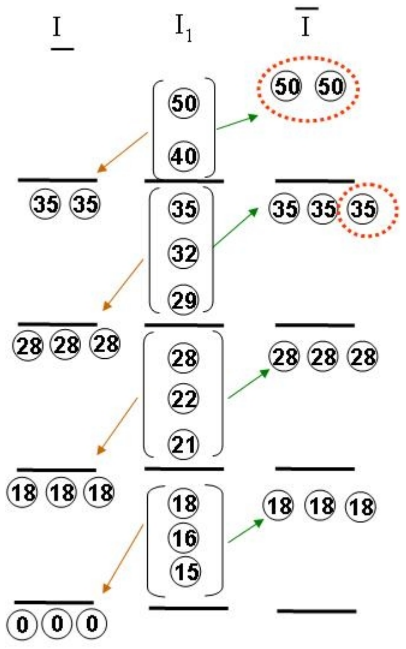

Figure 5.2: The figure shows the big demands at position p. The big demands in instance I_1 (center) are sorted and partitioned into groups. The big demands in $\overline{I}$ (on right) are obtained by rounding demands up to the maximum of the group, and in $\underline{I}$ (on left) by rounding demands down to the maximum of the next lower group. Each group, except the highest group, contains 3 demands. A covering of the demands in $\underline{I}$ yields a covering of all but the 3 largest demands (dotted circles) of $\overline{I}$.

demands, at each position $p \in R$, yields

$$\mathrm{OPT}(\overline{I}) \leq \mathrm{OPT}(\underline{I}) + \sum_{p \in R} 2p \cdot n_p \epsilon^2 \tag{5.9}$$

Using the Rad Lower bound (Lemma 1.3.2) for $\mathrm{OPT}(I_1)$ and the fact that big demands have size at least ϵk, we get

$$\mathrm{OPT}(I_1) \geq \frac{2}{k} \sum_{p \in R} p \cdot n_p \cdot \epsilon k = \sum_{p \in R} 2p \cdot n_p \epsilon \tag{5.10}$$

Thus starting from Equation 5.9 we have

$$\begin{aligned} \mathrm{OPT}(\overline{I}) &\leq \mathrm{OPT}(\underline{I}) + \sum_{p \in R} 2p \cdot n_p \epsilon^2 \\ &\leq \mathrm{OPT}(I_1) + \sum_{p \in R} 2p \cdot n_p \epsilon^2 \qquad (5.11) \\ &\leq \mathrm{OPT}(I_1) + \epsilon \mathrm{OPT}(I_1) \qquad (5.12) \end{aligned}$$

where Line 5.11 follows by Equation 5.8 and Line 5.12 follows from Equation 5.10. Thus $\mathrm{OPT}(\overline{I}) \leq (1+\epsilon)\mathrm{OPT}(I_1)$ which is $(1+O(\epsilon))\mathrm{OPT}$ by Equation 5.7. The proof is complete as $\overline{I}$ is the instance the algorithm outputs.

5.5 Proof of Lemma 5.3.3

We define a general notion, *small expanse* (Definition 5.5.1), for tours covering points in one dimension. In our setting small expanse tours cover points in at most two regions and Lemma 5.5.2 implies the existence of a near optimal solution that consists of only small expanse tours.

Definition 5.5.1. *(Small expanse) For any tour t that starts and ends at origin θ, let d be the distance to the farthest point from θ and d' be the distance to the closest point to θ of all points covered by t. The* expanse *of tour t is d/d'. A* small *expanse tour has expanse at most $1/\epsilon$.*

Note that in the single depot case, d, d' are simply the maximum and minimum positions of the customers covered by a tour. Thus a tour has small expanse if it covers customers between positions $p' \leq p$, such that $p/p' \leq 1/\epsilon$.

Lemma 5.5.2. *Let t be a tour covering points on a line. For any $\epsilon \leq 1/2$, there exists a collection of small expanse tours $(t_i)_{i\geq 0}$ which cover the same customers covered by t and have total cost at most $(1+2\epsilon)cost(t)$.*

Proof. Assume t starts at some origin continues in one direction until the last point is visited and then returns back to the origin along the same path, i.e each point is visited twice by t. If that is not case, t can be replaced with cheaper tour that fits the above description and covers the same points as t.

Let p be the farthest point from the origin covered by t. For every $i \geq 0$, define a new tour t_i that starts and ends at the origin and covers only the points covered by t at distances $(\epsilon^{i+1}p, \epsilon^i p]$. See Figure 5.3. Together, the tours $(t_i)_i$ cover exactly the points that t cover and by construction, each t_i has small expanse. We have: $\text{cost}(t) = 2p$, and the cost the collection of $(t_i)_i$ is

$$2(1+\epsilon+\epsilon^2+\ldots)p < cost(t)/(1-\epsilon) \leq (1+2\epsilon)\text{cost}(t).$$

□

Applying Lemma 5.5.2 to every tour in $\text{OPT}(I')$ yields a set of small expanse tours of cost $(1+2\epsilon)\text{OPT}(I')$ which we will use to prove Lemma 5.3.3. Intuitively, since each region has large expanse, they can be solved independently to get a near optimal small expanse solution, as only a few tours of the optimal small expanse solution will cover customers in more than one region.

Let S be the optimal solution of I' that uses only small expanse tours. Fix a particular partition P_j of I' and let T_j be the set of tours of S that cover customers in more than one region under P_j. Since each region has large expanse ($\geq 1/\epsilon$) and each tour in $t \in T_j$ has small expanse, t covers customers in at most two regions of partition P_j. For each $t \in T_j$, make two copies of t, and assign one copy to cover the customers in the first region and the second copy to cover the customers in the second region of t. After these modifications all tours cover customers in only one region. For partition P_j we obtain:

$$\sum_{R\in P_j} \text{OPT}(R) \leq S + \sum_{t\in T_j} \text{cost}(t).$$

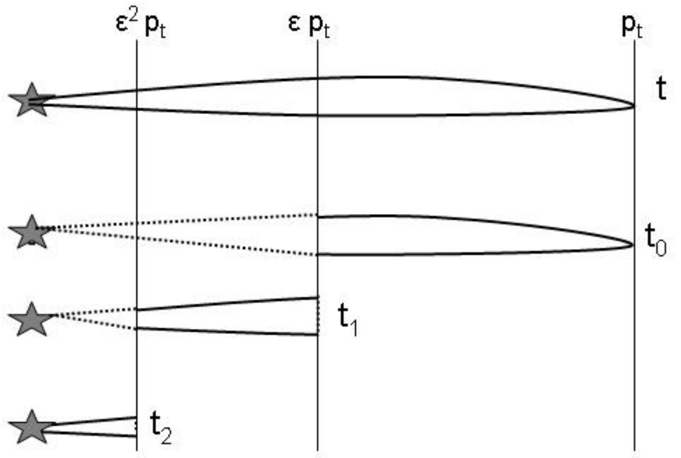

Figure 5.3: Lemma 5.5.2. The depot is the star. Tour t of length p_t is replaced with tours t_0, t_1, t_2, by adding the dashed segments from the depot. No points are covered by the dashed segments so t_i only covers points in $(p_t\epsilon^{i+1}, p_t\epsilon^i]$.

Summing over all $1/\epsilon$ partitions, we obtain:

$$\sum_{0\le j<1/\epsilon}\sum_{R\in P_j}\mathrm{OPT}(R) \le \sum_{0\le j<1/\epsilon}\left(S+\sum_{t\in T_j}\mathrm{cost}(t)\right) \tag{5.13}$$

Note that for partitions P_i, P_j such that $j \neq i$, T_i and T_j are disjoint; a tour $t \in T_j$ spans across two consecutive regions in P_j and thus two consecutive blocks. These consecutive blocks are in the same region in partition P_i, thus $t \notin T_i$. This implies that the right hand side of Equation 5.13 is at most $(1/\epsilon + 1)S$. Thus we have that,

$$\sum_{0\le j<1/\epsilon}\sum_{R\in P_j}\mathrm{OPT}(R) \le (1/\epsilon+1)S$$

As the sum on the left hand side has $1/\epsilon$ terms, there must exist a term j^* for which $\sum_{R\in P_{j^*}}\mathrm{OPT}(R) \le (1+\epsilon)S$. Another way to see this is that the sum of solutions obtained for all $1/\epsilon$ possible partitions is $(1/\epsilon + 1)S$. Thus at least one partition yields a solution of cost $\le \frac{(1/\epsilon+1)S}{(1/\epsilon)} = (1+\epsilon)S$. By Lemma 5.5.2 the optimal small expanse solution $S \le (1+2\epsilon)\mathrm{OPT}$. Thus for P_{j^*} we have that $\sum_{R\in P_{j^*}}\mathrm{OPT}(R) \le (1+O(\epsilon))\mathrm{OPT}$, which completes the proof.

5.6 Proof of Lemma 5.3.7

First we analyze the running time of Algorithm 15. The bottleneck is the time required to solve the linear program, which requires polynomial time in the number of variables and constraints. The linear program has one variable for each tour configuration and a constraint for each demand type and each position in region R. Below we show that there are $(1/\epsilon)^{O(1/\epsilon)}$ tour configurations, $(1/\epsilon)^2\log(1/\epsilon)(1+1/\epsilon^2)$ demand types and $(1/\epsilon)^2\log(1/\epsilon)$ positions. This implies that the running time of Algorithm 15 is $(1/\epsilon)^{O(1/\epsilon)}$ which is constant as $(1/\epsilon)$ is a constant.

Now we analyze the number of positions, big demand types and tour configurations in region R. As R spans $(\epsilon^{1/\epsilon}p, p]$ for some p, by Lemma 5.3.1, all positions are at powers of $(1+\epsilon)$ thus there are at most $c_{loc} = (1/\epsilon)^2 \log(1/\epsilon)$ positions in R.[2] Each position has only $1/\epsilon^2+1$ distinct big demand sizes. Thus the number of big demand types is at most $c_{type} = c_{loc} \cdot (1 + 1/\epsilon^2) = (1/\epsilon)^2 \log(1/\epsilon)(1 + 1/\epsilon^2)$. As big demands have value at least $k\epsilon$, at most $1/\epsilon$ big demands can be covered by any tour of capacity $\leq k$. Thus the number of tour configurations is at most $c_{loc} \cdot \sum_{j \leq 1/\epsilon} (c_{type})^j = (1/\epsilon)^{O(1/\epsilon)}$.

Let T be the tours output by Algorithm 15. The basic solution x^* is rounded to get $\bar{x}$ and T is obtained by creating tours that cover the big demands as specified by $\bar{x}$. Since x^* satisfies constraint 5.3, the tours associated with $\bar{x}$ will cover all the big demands in R.

Now we bound cost(T). Let OPT(R) be the optimal solution of (the unrelaxed problem) R. Since x^* can cover small demands as fluid and use fractional tour configurations, $\sum_{f \in \mathcal{F}} m_f x^*_f \leq \text{OPT}(R)$. Using the values of c_p and c_{type} derived in the analysis of the running time , the linear program has $c = c_{loc} + c_{type} = ((1/\epsilon)^2 \log(1/\epsilon))(2 + 1/\epsilon^2)$ constraints other than the non-negativity constraints. Thus a basic optimal solution x^* has at most c fractional coordinates. The farthest position m_f of tour configuration f, is at most p_R, thus,

$$\sum_{f \in \mathcal{F}} m_f \bar{x}_f \leq (\sum_{f \in \mathcal{F}} m_f x^*_f) + c \cdot p_R \leq \text{OPT}(R) + c \cdot p_R$$

5.7 Proof of Lemma 5.3.8

Let A be the new tours added by Algorithm 16. Let $(d_j)_{j \geq 1}$ be the opening position of each new in A, sorted in increasing order, and define $d_0 = 0$ for convenience. Thus $(d_j)_{j \geq 1}$ are the distances from the opening position to the depot. Let A_j be the set of tours in A with opening position $\geq d_j$. We have:

$$\text{cost}(G) = \text{cost}(T) + \sum_{j \geq 1} 2(d_j - d_{j-1})|A_j|. \tag{5.14}$$

Let $T_j = \{t \in T : m_t \geq x_j\}$ denote the tours of T whose farthest position is at least d_j i.e., $m_t \geq d_j$. Thus T_j represents the set of tours that pass beyond position d_j. Let $cap(j)$ denote the total remaining capacity (before any small demands are added) of tours of T_j, i.e., $cap(j) = \sum_{t \in T_j} c_t$. Recall that $S = (w_i, p_i)_i$ is the set of small demands and let $small(j) = \sum_{(w_i,p_i) \in S : p_i \geq d_j} w_i$ denote the total small demand at positions at least d_j.

The algorithm opens a new tour at position p only if no other tour passing the position has enough remaining capacity to cover a small demand at the position. Thus as a new tour is opened at position d_j, it must be that every tour in T_j has remaining capacity less than ϵk (i.e the maximum size of a small demand). Thus the total amount of small demand assigned by the algorithm to the tours in T_j is at least

$$(\text{small demand added to } T_j) \;\geq\; cap(j) - |T_j|\epsilon k,$$

[2]The number of positions can be less than c_{loc} since we don't need to count positions with no customers.

and the total amount of small demand remaining further than position d_j is at most

$$\text{(small demand remaining beyond } d_j) \;\leq\; \mathit{small}(j) - \mathit{cap}(j) + |T_j|\epsilon k \;\leq\; |T_j|\epsilon k. \tag{5.15}$$

Then second inequality in Equation 5.15, follows as constraint (5.3) of the linear program implies that the total small demand to the right of position d_j is equal to the remaining capacity of the tours that go to the right of d_j, i.e $\mathit{small}(j) - \mathit{cap}(j) \leq 0$ for all j. As the algorithm opens a new tour only when no other new tour has enough remaining to cover a some small demand, all but one new tour that goes to the right of d_j is almost full, we have:

$$|A_j| \leq \frac{|T_j|\epsilon k}{(1-\epsilon)k} + 1 = \frac{\epsilon|T_j|}{(1-\epsilon)} + 1. \tag{5.16}$$

Substituting Equation 5.16 into Equation 5.14 we get that,

$$\mathrm{cost}(G) = \mathrm{cost}(T) + \frac{\epsilon}{(1-\epsilon)} \sum_{j\geq 1} 2(d_j - d_{j-1})|T_j| + 2\max_j d_j.$$

As $\mathrm{cost}(T) \geq \sum_{j\geq 1} 2(d_j - d_{j-1})|T_j|$ and $\max_j d_j \leq p_R$ the maximum position in R, we obtain for $\epsilon \leq 1/2$,

$$\mathrm{cost}(G) \leq \frac{\mathrm{cost}(T)}{1-\epsilon} + 2p_R \leq (1+2\epsilon)\mathrm{cost}(T) + 2p_R.$$

Chapter 6

Extension to Constant Number of Depots

This chapter's results are joint work with Shay Mozes and Claire Mathieu and will be included in the journal version of [20].

6.1 Introduction

This chapter presents an APTAS for 1-dimensional Unsplit VRP with a constant number of depots.

Recall the problem: Given a positive integer k denoting the vehicle capacity, a set $C = \{(p_i, w_i)\}_{i \le n}$ of n customers each with a position p_i on the line and a demand $w_i \le k$, and a set of m depots D each represented also by a position on the line, find a collection of tours of minimum total length covering the demands of all customers in C, such that each tour starts and ends at some depot, delivers at most k demand and such that no customer's demand is split up among multiple tours.

With multiple depots, the problem is no longer equivalent to the scheduling problem on a single batch machine or the generalization of bin-packing problem mentioned in the introduction of Chapter 5. It also cannot be viewed as the *multiple* versions of those problems, *e.g.* scheduling with *multiple* batch machines or the generalized bin packing with *multiple* bin types, as these problems have a much more general cost structure than the 1-dimensional setting allows.

The bin packing problem still reduces to the Unsplit problem with multiple depots: set all p_i equal, and have all but one of the depots located at very far from all customers. Hence by the hardness results of bin packing, this problem does not admit a PTAS unless unless P=NP.

As in the single depot setting, the hardness result does not exclude an asymptotic PTAS, and we extend our APTAS of Chapter 5 to the setting with constant number of depots and prove Theorem 6.1.1. As before any instance of this problem can be scaled so that the cost of an optimal solution is arbitrarily large without changing the solution itself. With multiple depots the cost of the solution is determined by the radii of customer r_i. Recall that the *radius* of customer $i \in C$ is

$r_i = \min_{d \in D} dist(i, d)$, i.e. the distance of customer i to its nearest depot. Therefore, to define a notion of asymptotic we restrict the ratio of the optimal solution and the maximum radius. In other words, scale the input so that $\max_i r_i = 1$. The asymptotic regime for our algorithm occurs when the cost of the scaled input is $\Omega(m/\epsilon^6)$, where m is the number of depots. Thus our algorithm is only an APTAS when the number of depots m is a constant.

Theorem 6.1.1. *For any instance I of 1-dimensional Unsplit problem with m depots, such that $\max_i r_i = O(\epsilon^6/m)OPT(I)$, Algorithm 17 outputs a solution of cost $(1+O(\epsilon))OPT$ in time $O(n\log(n)) + m\log(n)(1/\epsilon)^{O(1/\epsilon)}$.*

Overview of our approach. We reduce the problem on m depots to solving $m+1$ smaller problems, each with at most two depots. We solve the smaller problems independently and combine to get a solution for the original problem. The smaller problems with a single depot are solved using the algorithm of Chapter 5, and we design a new algorithm (Alg. 18) for the ones with two depots. Algorithm 18 starts by rounding positions (this time with respect to both depots), and then applies the De La Vega and Lueker [26] rounding to obtain a small number of distinct big demands at each position. We try a few different ways ($O(1/\epsilon)$) to partition the customers into disjoint regions, solve the problem in each region independently and combine to get a solution for the small problem. We pick the solution of minimum cost over all possible partitions we tried.

The partitioning scheme in Algorithm 18 is different than in the single depot setting. Now the instance is partitioned into three regions, *left*, *right* and *center*. The *left* and *right* regions are both covered only by tours originating from a one depot, hence they can be solved by the algorithm of Chapter 5. For the *center* region we find a near optimal solution to the *relaxed* problem, where the small demands are fluid, using a variant of the linear program of Chapter 5. We use a new greedy method tailored for two depots, to extend the solution so that it respects the unsplittable constraint for small demands.

The analysis involves using the shifting technique (as in the single depot case) to show that one of the ($O(1/\epsilon)$) partitions yields a near-optimal solution, and proving that a near-optimal solution can be constructed by greedily inserting the small demands into the relaxed solution.

6.2 Algorithm and Proof of Theorem 6.1.1

The APTAS for the 1-dimensional Unsplit problem with constant number of depots is presented in Algorithm 17. The algorithm partitions the original instance with m depots into 2 single depot instances and $m-1$ *between two depots* instances as shown in Figure 6.1. The single depot instances are solved using the Algorithm 13 of Chapter 5 and the *between two depots* instances are solved using the Algorithm 18 of Section 6.3.

```
Algorithm 17 APTAS for Unsplit with constant depots, in one dimension
Input: Depots $D$, customers $(p_i, w_i)_{1 \le i \le n}$ with vehicle capacity $k$
Precondition: $\max_i r_i \le \epsilon^6 \mathrm{OPT}$
1: Consider depots and customers in increasing order of location and partition $I$ into instances
   $I_1, \dots, I_{m+1}$ such that:
   • $I_1$ is a single depot Unsplit problem with the first depot and the customers to its left.
   • For $j \in [2, m]$, $I_j$ is a *between two depots* Unsplit problem (defined in subsection 6.2.1) with
   depots $j - 1$ and $j$ and the customers at positions in between.
   • $I_{m+1}$ is a single depot Unsplit problem with depot $m$ and the customers to its right.
2: Solve $I_1$ and $I_{m+1}$ using Algorithm 13 of Chapter 5.
3: Solve each $I_j$ for $j \in [2, m]$ with Algorithm 18.
Output: The union of the solutions found for $I_1, \dots, I_{m+1}$.
```

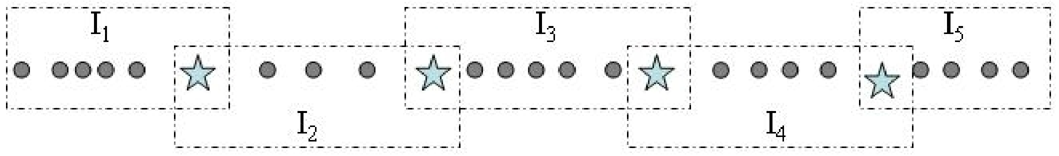

Figure 6.1: An instance of the multiple depots problem with four depots (stars) is partitioned into 5 smaller instances. Instances I_1 and I_5 are single depot instances, and instances I_2, I_3, I_4 are each instances between two depots with customers (gray circles) in between the two depots.

6.2.1 The Unsplit Problem Between Two Depots.

The 1-dimensional Unsplit problem *between two depots* is: Given two depots θ_l and θ_r, located distance L apart on the line, a positive integer k the vehicle capacity, a set $C = \{(p_i, w_i)\}_{i \le n}$ of n customers[1] each with a demand $w_i \le k$ and a position between the depots on the line, find a collection of tours of minimum total cost covering the demands of all customers in C, such that each tour starts and ends at a depot, delivers at most k demand and such that each customer $i \in C$ receives its entire demand w_i by a single tour. In other words the customer's demand is not allowed to be split up among multiple tours.

We can assume that an instance is prepocessed as defined below:

Definition 6.2.1. *(Preprocessed) An instance of the* 1*-dimensional Unsplit problem between two depots with* n *customers is preprocessed if:*

- *The depot* θ_l *is located at the origin, and* θ_r *is located at position* L.
- *All customers positions are* $0 \le p_i \le L$ *(i.e between the depots).*
- *No customer is located closer than* $\epsilon \cdot r_{\max}/n$ *from the depot, where* $r_{\max} = \max_i r_i$.

We can always define the positions of the depots to be the origin and L and reinterpret the positions of the customers accordingly. As all customer positions were between the depots now they

[1]The number of customers in a between two depots problem can be different from the number of customers in the constant number of depots problem.

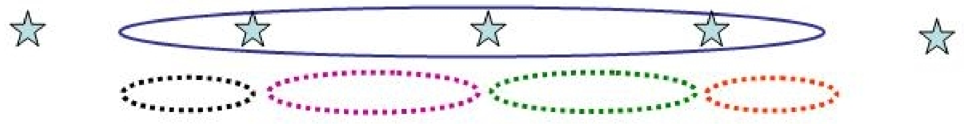

Figure 6.2: An instance of the multiple depots 1-d Unsplit problem, with five depots (stars). The solid tour covers points in four instances and is replaced by the four dashed tours, each covering points in one instance.

will have positions between 0 and L. If there are any customers located closer than $\epsilon \cdot r_{\max}/n$ from either depot, serve each with a separate tour originating from the closest depot. The overall cost for these tours is at most $2\epsilon r_{\max}$, which is at most ϵOPT as any solution has cost at least $2r_{\max}$.

Observe that if p is the maximum position of any customer on tour t then t has cost $2p$ if it originates from θ_l or cost $2(L-p)$ if it originates from θ_r. The radius of customer i is $r_i = \min\{p_i, L-p_i\}$.

6.2.2 Proof of Main Theorem 6.1.1

Correctness of Algorithm 17. Let I be an instance of 1-dimensional Unsplit with multiple depots and OPT denote its optimal solution. Let $I_1, \dots I_{m+1}$ be the smaller instances created by Algorithm 17. We can assume that each tour in OPT covers customers in only one instance I_j: Any tour π covering customers in $x > 1$ instances $I_{i_1}, I_{i_2}, \dots, I_{i_x}$ can be replaced x tours $\pi'_1, \pi'_2, \dots, \pi'_x$ where π'_p is the portion of π that lies in instance I_{i_p}. See Figure 6.2. The sum of the lengths of the $(\pi')_p$ is equal to length of π. Thus we have that

$$\text{OPT}(I) = \text{OPT}(I_1) + \text{OPT}(I_2) + \dots + \text{OPT}(I_{m+1}) \tag{6.1}$$

We show below that Algorithm 17 computes solutions of cost $(1+O(\epsilon))\text{OPT}(I_j)$ for each I_j. Thus by Equation 6.1 this yields a solution of cost $(1+O(\epsilon))\text{OPT}(I)$ overall.

The between two depots instances $I_2, \dots I_m$ are solved by Algorithm 18 presented in Section 6.3. Fix an I_j for any $j \in [2, m]$. By the proof of Theorem 6.3.1, the solution of I_j has cost at most $(1+O(\epsilon))\text{OPT}(I_j) + O(1/\epsilon^5)r_{I_j}$ (see Equation 6.10), where r_{I_j} denotes the maximum radius of I_j. The single depot instances I_1 and I_{m+1} are solved by Algorithm 13 from Chapter 5 and the proof of Theorem 5.1.1 shows that the solutions for I_1 and I_{m+1} have cost at most $(1+O(\epsilon))\text{OPT}(I_1) + O(1/\epsilon^5)r_{I_1}$ and $(1+O(\epsilon))\text{OPT}(I_m) + O(1/\epsilon^5)r_{I_{m+1}}$ respectively (see Equation 5.6).

As $r_{\max} \geq r_{I_j}$ for all $j \in [1, m+1]$, the union of the solutions of all I_j is at most

$$(1+O(\epsilon)) \sum_{j=1}^{m+1} \text{OPT}(I_j) + O(1/\epsilon^5)(m+1) \cdot r_{\max}$$

which is at most $(1+O(\epsilon))\text{OPT}(I) + O(\epsilon)\text{OPT}(I) = (1+O(\epsilon))\text{OPT}(I)$ by Equation 6.1 and the asymptotic assumption that $r_{\max} \leq \text{OPT} \cdot O(\epsilon^6/m)$.

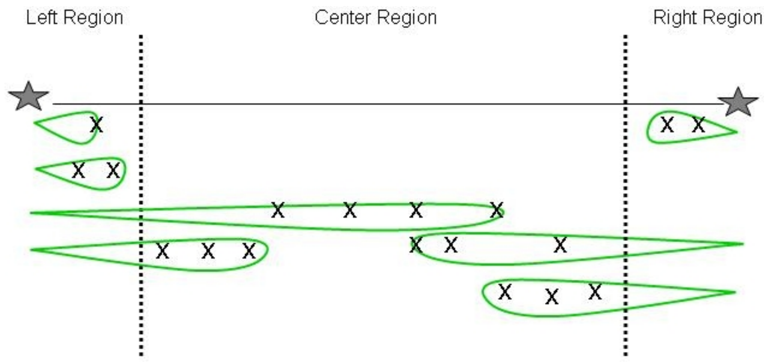

Figure 6.3: A solution computed by Algorithm 18. The stars are the depot. The instance between two depots are partitioned into $left$, *center* and *right*, and each tour covers only customers (marked "x") from one region.

Running time of Algorithm 17. By Theorem 5.1.1 and Theorem 6.3.1 each I_j is solved in time $O(n_j \log(n_j)) + \log(n_j)(1/\epsilon)^{O(1/\epsilon)}$, where n_j is the number of customers in instance I_j. Thus over all the running time is bounded by $O(n \log(n)) + m \log(n)(1/\epsilon)^{O(1/\epsilon)}$.

6.3 Algorithm for Unsplit Between Depots and Proof of Theorem 6.3.1

We present Algorithm 18 an APTAS for the 1-dimensional Unsplit problem between two depots (defined in Section 6.2.1) and prove Theorem 6.3.1. We note that in all the remaining sections of this chapter notation OPT and $r_{\max}$ refers to an instance of the problem between two depots.

As before the problem does not admit an asymptotic approximation scheme in the usual sense, since any instance can be scaled so that the cost of an optimal solution is arbitrarily large without changing the solution itself. We define the asymptotic regime to be when OPT $= \Omega(1/\epsilon^6)r_{\max}$. Or in other words, if the input is scaled so that $\max r_i = 1$, the asymptotic regime occurs when the cost of the scaled input is $\Omega(1/\epsilon^6)$. Figure 6.3 shows a solution computed by our algorithm.

Theorem 6.3.1. *For any instance I of the 1-dimensional Unsplit problem between two depots, such that $\max_i r_i = O(\epsilon^6)OPT(I)$, Algorithm 18 outputs a solution of cost $(1 + O(\epsilon))OPT$ in time $O(n \log(n)) + \log(n)(1/\epsilon)^{O(1/\epsilon)}$.*

6.3.1 Rounding

Algorithm 19 reduces the number of positions and the number of distinct big demand sizes at each position. Lines 1 and 2 places markers at distances that are powers of $(1 + \epsilon)$ from either depot and then all customers are moved to their closest marker. As in the single depot case, we call a demand w_i *big* if $w_i \geq \epsilon k$ and *small* otherwise. The big demands are rounded using the Fernandez de la Vega and Lueker scheme [26].

Algorithm 18 APTAS for Unsplit between two depots, in one dimension

Input: A preprocessed instance with depots θ_l, θ_r located distance L apart, customers $(p_i, w_i)_{1\leq i\leq n}$, and vehicle capacity k

Precondition: $\max_i r_i \leq \epsilon^6\mathrm{OPT}$

```
1: Round the input using Algorithm 19, described in section 6.3.1.
2: for $1 \leq j \leq 1/\epsilon$ do
3:     Partition the instance into regions $R_l, R_m, R_r$ using partition $P_j$ (Definition 6.3.3).
4:     Solve the *relaxed* two depot problem in $R_m$ treating small demands as fluid using Algorithm
       15 with the linear problem of Chapter 5 with Equations 6.2-6.5.
5:     Extend the *relaxed* solution into a feasible solution, $Sol_m$, for all customers in $R_m$ using
       Algorithm 20.
6:     Define $I_l, I_r$ as instances of the *single depot* Unsplit problem such that $I_l$ consists of customers
       in $R_l$ and depot $\theta_l$ and $I_r$ consist of customers in $R_r$ and depot $\theta_r$.
7:     Find solutions $Sol_l$, $Sol_r$ for $I_l$ and $I_r$ using Algorithm 13 of Chapter 5.
8:     Let $\mathrm{Best}(P_j) = Sol_l \cup Sol_m \cup Sol_r$ be the solution found using partition $P_j$.
9: end for
```

Output: $\min_j \mathrm{Best}(P_j)$, the minimum cost solution found over all partitions.

Algorithm 19 Rounding

Input: Preprocessed instance I with depots θ_l,θ_r distance L apart, customers $(p_i, w_i)_{i\leq n}$ such that $p_i \leq L$ for all i

```
1: Start at distance $l_0 = \frac{\epsilon r_{\max}}{n}$ from the left depot, go right and place a marker at each distance
   $l_0(1+\epsilon)^j$ for $j \geq 0$ until distance $L$ is reached.
2: Start at distance $l_0$ from the right depot, go left and place a marker at each distance $l_0(1+\epsilon)^j$
   for $j \geq 0$ until distance $L$ is reached.
3: Round each customer's position $p_i$ to the nearest marker.
4: Round big demands at each position as in Lines 3-6 of Algorithm 14 of Chap. 5.
```

Output: rounded instance I'.

The following lemma shows that OPT of I' can be modified into a near optimal solution for instance I. See Section 6.4 for the proof.

Lemma 6.3.2. *Given an instance I of the 1-dimensional Unsplit problem between two depots, Algorithm 19 outputs an instance I' such that:*

1. *Each customer's position is at distance $d_{\max}\epsilon/n(1+\epsilon)^j$, for integer $j \geq 0$, from a depot*
2. *Each position has at most $1/\epsilon^2 + 1$ distinct sizes of big demands.*
3. *A solution of I' of cost C' can be converted into a solution for I of cost $(1+\epsilon)C'$.*
4. *$OPT(I') \leq (1 + O(\epsilon))OPT(I)$.*
5. *$OPT(I')$ can be modified into a feasible solution of I' of cost $(1 + O(\epsilon))OPT(I)$.*

6.3.2 Partitioning into Regions

We define $1/\epsilon$ ways to partition the instance into disjoint *regions* R_l, R_m, R_r as given in Definition 6.3.3. Lemma 6.3.4 shows that for at least one of these partitions, solving each region independently and taking the union of the solutions yields a near optimal solution for the whole instance.

Definition 6.3.3. *(Partitions) Let I' be a rounded instance of the 1-dimensional Unsplit problem between two depots. Define $1/\epsilon$ ways to partition I' into regions R_l, R_m, R_r as follows: For $j \in [1, 1/\epsilon]$, in* partition *P_j region R_l consists of customers at positions $p \leq \epsilon^j L$, R_m the customers at positions $(\epsilon^j L, (1-\epsilon^j)L)$ and R_r the customers at positions $p \geq (1-\epsilon^j)L$.*

Observe that in all partitions region R_l consists of customers within distance $\epsilon^j L$ from depot θ_l, region R_r consists of customers within distance $\epsilon^j L$ from depot θ_r and region R_m consists of the customers with $r_i > \epsilon^j L$, i.e those farther than $\epsilon^j L$ from both depots.

Lemma 6.3.4 states the main structural property that allows us solve each region separately. Its proof uses small expanse tours (Definition 5.5.1) and the *shifting* technique used in the proof of Lemma 5.3.3. See Section 6.5.

Lemma 6.3.4. *Let I' be a rounded instance of the 1-dimensional Unsplit problem between two depots and let P_j, for $j \in [1, 1/\epsilon]$, be the partition of I' into regions R_l, R_m, R_r. Define I_l and I_r to be instances of the* single depot *1-dimensional Unsplit problem where I_l consists of the customers in R_l and depot θ_l, and I_r consist of the customers in R_r and depot θ_r.*

There exists a partition P_j, such that $OPT(I_l) + OPT(R_m) + OPT(I_r) \leq (1 + O(\epsilon))OPT(I')$.

6.3.3 Solving the Middle Region

We focus on solving the problem in region R_m of a partition. As in the single depot setting, we first solve the *relaxed* problem (Definition 5.3.4) for R_m and extend that solution into a feasible solution for all demands in R_m.

Solving the relaxed problem. Due to rounding, region R_m contains only a constant number of positions with customers, and each position contains a constant number of distinct big demands. We will describe big demand types of R_m exactly as in the single depot case as given in Definition 5.3.5. To describe a tour in R_m we extend the single depot tour configuration to include the originating depot θ of the tour and we revise the meaning of m_f to be the farthest position vistied by the tour from its originating depot θ.

Definition 6.3.5. *(Augmented Configuration) An* augmented configuration *f of a tour in R_m consists of a depot θ, a postion m_f that denotes the farthest position from θ visited by the tour, and a multiset D_f of big demand types each with position at most m_f, whose (rounded) sizes sum up to at most k.*

Let cost(f) denote the length of the tour described by augmented configuration f. Thus if $\theta = \theta_l$ then cost$(f) = 2m_f$ and if $\theta = \theta_r$, cost$(f) = 2(L - m_f)$. Let c_f denote the remaining capacity of an augmented tour configuration f (i.e., $c_f = k - \sum_{(p,b) \in D_f} b$). For any big demand type d, let $n_f(d)$ denote the multiplicity of d in D_f.

A slight variant of the linear program from the single depot case solves the relaxed problem. Let S be the set of small demands in region R_m. Instead of considering the small demand fluid at *each* position the linear program will consider the small demand as fluid between *every pair* of positions.

The linear program has one variable x_f for each extended tour configuration $f \in \mathcal{F}$ and the objective is to select a minimum cost set of tour configurations such that two constraints are satisfied: Constraint 6.3 ensures that all big demand types are covered by the selected tour configurations and Constraint 6.4 ensures the total small demand can be covered by the remaining capacities of the tour configurations appearing between each pair of positions, where a configuration f appears between positions $[p', p'']$ if $p' \le m_f \le p''$.

$$\min \quad \sum_{f \in \mathcal{F}} \text{cost}(f) \cdot x_f \tag{6.2}$$

$$s.t \quad \sum_{f \in \mathcal{F}} x_f n_f(d) \ge n(d) \qquad \forall \text{ demand types } d \tag{6.3}$$

$$\sum_{\substack{f \in \mathcal{F}: \\ p' \le m_f \le p''}} c_f x_f \ge \sum_{\substack{(p_i, w_i) \in S \\ p' \le p_i \le p''}} w_i \qquad \forall \text{ positions } p' \le p'' \tag{6.4}$$

$$x_f \ge 0 \tag{6.5}$$

We will use Algorithm 15 with Line 3 modified to solve the linear program of Equations 6.2-6.5 to obtain a solution for the relaxed problem in R_m. It rounds a basic solution to obtain a feasible relaxed solution. Let OPT(R_m) denote the optimal (unrelaxed) solution. Lemma 6.3.6 is a version of Lemma 5.3.7 and it bounds the cost of the solutions in terms of OPT(R_m) and $r_{\max}$. Later we will use the "asymptotic" assumption (so that the additive $r_{\max}$ term is small compared to OPT). Note that the factor multiplying $r_{\max}$ is different from the factor in Lemma 5.3.7 because the linear program in Equations 6.2-6.5 has a different number of constraints from the linear program of Equations 5.1-5.4. The proof appears in Section 6.6.

Lemma 6.3.6. *Let R_m be the middle region of rounded instance I', and let $OPT(R_m)$ denote the cost of the optimal (unrelaxed) solution of R_m. Given R_m and using the linear program of Equations 6.2-6.5, Algorithm 15 outputs in time $(1/\epsilon)^{O(1/\epsilon)}$ a set of tours T covering all big demands in R_m such that $cost(T) \leq OPT(R_m) + r_{\max} \cdot (2/\epsilon^2)\log(1/\epsilon)[1 + 1/\epsilon^2 + (2/\epsilon^2)\log(1/\epsilon)]$.*

Extending a relaxed solution. Let $T = (\theta_t, m_t, c_t)_t$ denote the list of tours output by Algorithm 15 where θ_t denotes the originating depot, m_t denotes the farthest position from θ_t and c_t denotes the remaining capacity of tour t after it has covered the big demands. Algorithm 20 greedily extends these tours to cover the small demands of R_m in a feasible way (i.e., without splitting any of them). Algorithm 20 checks for each small demand (p_i, w_i) if any existing tours passing by p_i have capacity to cover w_i. If not, a new tour originating from the depot closest to p_i is created to cover i.

Algorithm 20 Greedy Extension

Input: Small demands $(p_i, w_i)_i$, a list T of tours $(\theta_t, m_t, c_t)_t$ s.t tour t originates from depot θ_t, has remaining capacity c_t, and its farthest position from θ_t is m_t

```
1: for each small demand $(p_i, w_i)$ in order of decreasing radii $r_i$ do
2:   if there is a tour $t \in T$ that passes by $p_i$ (i.e $p_i$ is between $\theta_t$ and $m_t$) with $c_t \geq w_i$ then
3:     cover $(p_i, w_i)$ with $t$ and set $c_t := c_t - w_i$
4:   else
5:     add a new tour $t$ with $m_t = p_i$, $c_t = k$ and set its depot, $\theta_t$, to the closest depot from $p_i$
6:     cover $(p_i, w_i)$ with $t$ and set $c_t := c_t - w_i$
7:   end if
8: end for
```

Output: the resulting tours.

Lemma 6.3.7 bounds the cost of Algorithm 16 in terms of $cost(T)$ and $r_{\max}$, and is proved in Section 6.7. Note that later we will use the "asymptotic" assumption to show that the additive $r_{\max}$ term is small compared to OPT.

Lemma 6.3.7. *On input $T = (\theta_t, p_t, w_t)_t$, Algorithm 20 outputs G with $cost(G) \leq (1+2\epsilon)cost(T) + 2r_{\max}$.*

6.3.4 Proof of Theorem 6.3.1

Correctness of Algorithm 18. By Lemma 6.3.2 the optimal solution of the rounded instance is a near optimal solution for the original instance. To solve the rounded instance Algorithm 18 tries all $1/\epsilon$ ways to partition it into regions R_l, R_m, R_r. Lemma 6.3.4 shows that for at least one of the $1/\epsilon$ partitions, P^*, a near optimal solution is obtained by solving the single depot instances I_l and I_r and the problem between two depots in region R_m independently and combining the solutions. For the rest of the analysis focus on the execution of Algorithm 18 that uses partition P^* and let I_l^*, R_m^*, I_r^*

be the subproblems from partition P^*. Below we show that Algorithm 18 obtains solutions of I_l^*, I_r^* and R_m of such that

$$\text{cost}(I_l^*) \leq (1+O(\epsilon))\text{OPT}(I_l^*) + O(1/\epsilon^5)r_{\max} \tag{6.6}$$

$$\text{cost}(I_r^*) \leq (1+O(\epsilon))\text{OPT}(I_r^*) + O(1/\epsilon^5)r_{\max} \tag{6.7}$$

$$\text{cost}(R_m^*) \leq (1+O(\epsilon))\text{OPT}(R_r^*) + O(1/\epsilon^5)r_{\max} \tag{6.8}$$

Thus the union of the solutions of I_l^*, I_r^*, R_m^* yields a solution of the rounded instance I' of cost at most,

$$(1+O(\epsilon))(\text{OPT}(I_l^*) + \text{OPT}(R_m^*) + \text{OPT}(R_r^*)) + O(1/\epsilon^5)r_{\max}$$

$$= (1+O(\epsilon))\text{OPT}(I') + O(1/\epsilon^5)r_{\max} \tag{6.9}$$

$$= (1+O(\epsilon))\text{OPT} + O(1/\epsilon^5)r_{\max} \tag{6.10}$$

$$= (1+O(\epsilon))\text{OPT} \tag{6.11}$$

Equation 6.9 follows by Lemma 6.3.4 for P^* and Equation 6.10 follows by the last property of Lemma 6.3.2. Applying assumption $r_{\max} \leq \epsilon^6\text{OPT}$ shows that the additive cost is within the desired approximation factor.

Now we show that Equations 6.6-6.8 hold. Algorithm 13 is used to solve the single depot problems I_l and I_r. Since the maximum distance between any position and the depot in I_l, I_r is at most $r_{\max}$ the proof of Theorem 5.1.1 up to Equation 5.6, shows that Equations 6.6 and 6.7 hold.

Consider the subproblem involving region R_m^*. Using Algorithm 15 with the linear program of Equations 6.2-6.5 finds a set of tours T covering all big demands and all small demands as fluid. Given T, Algorithm 20 produces a solution that covers the small demands feasibly (without splitting) and by Lemma 6.3.7 the cost of the resulting solution for R_m^* is at most $(1+2\epsilon)\text{cost}(T) + 2r_{\max}$. Using Lemma 6.3.6 to bound the $cost(T)$, we get that

$$\text{cost}(R_m^*) \leq (1+2\epsilon)\text{OPT}(R_m^*) + \ell(\epsilon)r_{\max}$$

where $\ell(\epsilon) = (2/\epsilon^2)\log(1/\epsilon)[1 + 1/\epsilon^2 + (2/\epsilon^2)\log(1/\epsilon)] = O(1/\epsilon^5)$. Thus Equation 6.8 also holds.

Running Time of Algorithm 18. Rounding positions, rounding the big demands at each position, and partitioning the instance into regions can all be done in time $O(n\log(n))$. Each partition P_j yields instances I_l, R_m, I_r. Region R_m is solved in time $O(n) + (1/\epsilon)^{O(1/\epsilon)}$ as by Lemma 6.3.6 the relaxed problem can be solved in time $(1/\epsilon)^{O(1/\epsilon)}$ and by inspection one can see that the greedy extension can be computed in at most $O(n)$. By Theorem 5.1.1 solving I_r and I_l using Algorithm 13 of Chapter 5 requires time $O(n\log(n)) + \log n \cdot (1/\epsilon)^{O(1/\epsilon)}$. Thus overall partition P_j can be solved in time $O(n\log(n)) + \log n \cdot (1/\epsilon)^{O(1/\epsilon)}$

As Algorithm 18 does the above computation for $1/\epsilon$ possible partitions, the final run time is $O(n\log(n)) + \log(n) \cdot (1/\epsilon)^{O(1/\epsilon)}$.

6.4 Proof of Lemma 6.3.2

The first property follows from Lines 1-3 of Algorithm 19 as each customer is moved to its nearest marker and each marker is within distance $(r_{\max}\epsilon/n)(1+\epsilon)^j$ to one of the two depots. The second property holds for the same reason as in proof of Lemma 5.3.1. The fifth property follows by combining the third and fourth properties.

We focus on the third property. Let I_1 denote the instance obtained after rounding the positions (Line 3) and I' denote the instance obtained after rounding demands. Observe that a solution for I' is feasible for I_1 as demands in I_1 are less than or equal to demands in I'. Thus we only need to show that each tour π_1 from a solution of I_1 can be converted into a feasible tour for I of cost at most $(1+\epsilon)\text{cost}(\pi_1)$.

Consider a tour π_1 that originates from depot θ_l and covers its farthest customer from marked position m (An analogous argument applies when π_1 originates from depot θ_r). The marker at distance m implies that there must also be a marker at distance $m(1+\epsilon)$. If in I customer i is located at $p > m$, it must be that $p < m(1+\epsilon)$ as i is rounded to its closest marker in Line 3. Thus π_1 can be transformed into a feasible tour for I by extending its length by at most ϵm. If $p < m$, the length of π_1 has to be shortened to make it feasible for I, thus the tour in I will have cost $\leq (1+\epsilon)\text{cost}(\pi)$ trivially.

We focus on the forth property and start by analyzing rounding positions. Consider a tour π of $\text{OPT}(I)$ of length d that originates from depot θ_l. (An analogous argument applies when π originates from depot θ_r). The farthest customer i covered by π has $p = d$. By Line 1, there exists a marker within distance at most $(1+\epsilon)d$ from i. As the position of i is rounded to its nearest marker in Line 3 the rounded position is at distance at most $(1+\epsilon)d$. Thus π can be transformed into a feasible tour for I_1 by extending its length by at most ϵd, and so

$$\text{OPT}(I_1) \leq (1+\epsilon)\text{OPT}(I). \tag{6.12}$$

Next we analyze rounding demands. The analysis is similar to the proof (of the forth property) of Lemma 5.3.1 and we only list the equations that need modification. To convert the covering of $\underline{I}$ into a covering of $\overline{I}$ we still have at most $\lfloor n_p\epsilon^2 \rfloor$ demands at each position $p \in R$ that are not covered by the covering of $\underline{I}$, where n_p is the number of big demands at position p. Each of those demands is covered with a single tour that originates from its closest depot. For each position $p \in R$ let r_p be the radius of p, then we have

$$\text{OPT}(\overline{I}) \leq \text{OPT}(\underline{I}) + \sum_{p\in R} 2r_p \cdot n_p\epsilon^2 \tag{6.13}$$

Using the Rad Lower bound (Lemma 1.3.2) we have

$$\text{OPT}(I_1) \geq \frac{2}{k}\sum_{p\in R} r_p \cdot n_p \cdot \epsilon k = \sum_{p\in R} 2r_p \cdot n_p\epsilon \tag{6.14}$$

The rest of the analysis to show that $\text{OPT}(I') = (1+O(\epsilon))\text{OPT}$ follows the proof of Lemma 5.3.1 using Equation 6.13 in place of Equation 5.9 and Equation 6.14 in place of Equation 5.10.

6.5 Proof of Lemma 6.3.4

Definition 6.5.1. *(Lengthy Tour) Let I' be an instance of the problem between two depots. Denote the* far left *to be the customers with positions $p \in [0, \epsilon L]$ and the* far right *to be the customers with positions $p \in [(1-\epsilon)L, L]$. A tour is a* lengthy tour *if it originates from depot θ_l and covers a far right demand or if it originates from depot θ_r and covers a far left demand.*

Lemma 6.5.2. *There exists a near optimal solution of I' that contains no lengthy tours.*

Proof. Let t be a lengthy tour of $\text{OPT}(I')$ that originates from depot θ_l. Thus $\text{cost}(t) \geq 2(1-\epsilon)L$. Replace t by two tours t_l and t_r, where t_l originates at θ_l and covers all the customers covered by t except for those in the far right, and t_r originates at θ_r and covers the customers covered by t in the far right. Then $\text{cost}(t_l) \leq (1-\epsilon)L$ and $\text{cost}(t_r) \leq \epsilon L$. Thus for $\epsilon \leq 1/2$ we have that

$$\begin{aligned}
\text{cost}(t_l) + \text{cost}(t_r) &\leq (1-\epsilon)L + \epsilon L \\
&\leq (1-\epsilon)L + 2(1-\epsilon)\epsilon L \\
&= (1+2\epsilon)(1-\epsilon)L \quad \leq \quad (1+2\epsilon)\text{cost}(t)
\end{aligned}$$

An analogous argument shows that any lengthy tour of $t \in \text{OPT}(I')$ originating from θ_r can be replaced with two tours that are not lengthy, of total cost at most $(1+2\epsilon)\text{cost}(t)$. □

Let S be a solution of I' consisting of tours that have small expanse (Definition 5.5.1) and that are not far tours. By Lemmas 5.5.2 and 6.5.2

$$\text{cost}(S) \leq (1 + O(\epsilon))\text{OPT}(I') \tag{6.15}$$

Fix a particular partition P_j of I' into three regions R_l, R_m, R_r. We analyze the cost of computing I' as the union of I_l , I_r and R_m as defined in the statement of Lemma 6.3.4.

For any partition P_j of I' region R_l is contained in the far left and R_r is contained in the far right. As S consists of no lengthy tours, R_l is served by tours from depot θ_l and R_r is served by tours from depot θ_r. As all tours originate from one of the two depots, no tour in S covers customers in both R_l and R_m.

For partition P_j let T_j be the set of tours of S that cover customers in two regions (either in R_l and R_m or in R_r and R_m). Replace each $t \in T_j$ by tours t' and t'' originating from the same depot as t such that t' covers the customers covered by t in R_m and t'' covers the customers covered by t in its second region (either R_r or R_l). Observe that t', t'' each cover customers in only one region and $\text{cost}(t') + \text{cost}(t'') \leq 2\text{cost}(t)$. After these modifications all tours cover customers in only one region of partition P_j and R_l and R_r are served by depots θ_l and θ_r respectively. Thus for partition P_j we obtain:

$$\text{OPT}(I_l) + \text{OPT}(R_m) + \text{OPT}(I_l) \leq \text{cost}(S) + \sum_{t \in T_j} \text{cost}(t)$$

Summing over all $1/\epsilon$ partitions, we obtain:

$$\sum_{1\leq j\leq 1/\epsilon}(\mathrm{OPT}(I_l)+\mathrm{OPT}(R_m)+\mathrm{OPT}(I_R))\leq \sum_{1\leq j\leq 1/\epsilon}\left(\mathrm{cost}(S)+\sum_{t\in T_j}\mathrm{cost}(t)\right) \tag{6.16}$$

Note that for partitions P_i, P_j s.t $j \neq i$, T_i and T_j are disjoint. Consider a tour $t \in T_j$ that originates at θ_l and covers customers in R_l and R_m (an analogous argument holds if t originates at θ_r covering customers in R_m and R_r). As t covers in R_m, it must cover a customer at distance $> \epsilon^j L$ from θ_l, but as t also has small expanse, the closest customer from θ_l it covers is at distance $> \epsilon^{j+1}L$. Thus t does not cover any customers at positions $\leq \epsilon^{j+1}L$ which implies that $t \notin T_i$ for any $i > j$. On the other hand as t covers in R_l of partition P_j, it must cover a customer at distance $< \epsilon^j L$ from θ_l, but since t also has small expanse the farthest customer from θ_l it covers is at distance $< \epsilon^{j-1}L$. Thus t does not cover any customer at positions $\geq \epsilon^{j-1}L$ which implies that $t \notin T_i$ for any $i < j$.

Given that the T_j are disjoint, the right hand side of Equation 6.16 is at most $(1/\epsilon)\mathrm{cost}(S) + cost(S)$. Thus,

$$\sum_{1\leq j\leq 1/\epsilon}(\mathrm{OPT}(I_l)+\mathrm{OPT}(R_m)+\mathrm{OPT}(I_R))\leq (1/\epsilon+1)\mathrm{cost}(S)$$

As the sum on the left hand side has $1/\epsilon$ terms, there must exist a term j^* for which $\mathrm{OPT}(R_l) + \mathrm{OPT}(R_m) + \mathrm{OPT}(R_r) \leq (1+\epsilon)S$. As $\mathrm{cost}(S) \leq (1 + O(\epsilon)\mathrm{OPT}(I')$, the Lemma is proved using partition P_{j^*}.

6.6 Proof of Lemma 6.3.6

The bottleneck in the running time of Algorithm 15 is the time required to solve the linear program. The LP of Equation 6.2-6.5 has one variable per *augmented* tour configuration and a constraint for each demand type and each *pair* of positions in R_m. Below we show that the number of tour configurations, positions and demand types all remain the same as in the single depot case. Thus the running time of Algorithm 15 is still $(1/\epsilon)^{O(1/\epsilon)}$ which is constant for fixed ϵ.

We analyze the number of positions, big demand types and tour configurations. Region R_m can have twice as many positions as a region in the single depot case. R_m spans the positions between $(\epsilon^j L, (1-\epsilon^j)L]$ for some $j \in [1, 1/\epsilon]$ depending on the partition, and by Lemma 6.3.2, all positions in R_m are at distances $(\epsilon r_{\max}/n)(1+\epsilon)^i$ from one of the two depots. As there are at most $1/\epsilon^2 \log(1/\epsilon)$ powers of $(1+\epsilon)$ within the span of R_m, there are at most $c_{loc} = 2(1/\epsilon)^2 \log(1/\epsilon)$ positions.[2] As the big demands were rounded in the same manner as in the single depot case, each position has at most $1/\epsilon^2 + 1$ distinct of big demand sizes. Thus the number of big demand types in R_m is at most $c_{type} = c_{loc} \cdot (1 + 1/\epsilon^2) = 2(1/\epsilon)^2 \log(1/\epsilon)(1 + 1/\epsilon^2)$. As each tour can cover at most $1/\epsilon$ big

[2]The number of positions can be less than c_{loc} since we don't need to count positions with no demands.

demands, the number of *augmented* tour configurations is

$$\begin{aligned}&(\#\text{ depots})(\#\text{ farthest position from depot})(\#\text{ choices of at most } 1/\epsilon \text{ big demand types})\\ &= 2 \cdot c_{loc} \cdot \sum_{j \le 1/\epsilon} (c_{type})^j = (1/\epsilon)^{O(1/\epsilon)}\end{aligned}$$

Let T be the tours output by Algorithm 15 using the LP in Equations 6.2-6.5. As x^* satisfies constraint 6.4, tours T associated with $\bar{x_f}$, cover all the big demands in R_m.

Now we bound cost(T). We have that $\sum_{f \in \mathcal{F}} cost(f) x^*_f \le \text{OPT}(R)$, since x^* can cover small demands as fluid and the use fractional tour configurations. Using the values of c_{loc} and c_{type} derived above, the linear program has $c = c_{loc}^2 + c_{type} = (2/\epsilon^2)\log(1/\epsilon)[1 + 1/\epsilon^2 + (2/\epsilon^2)\log(1/\epsilon)]$ constraints other than the non-negativity constraints. Thus a basic optimal solution x^* has at most c fractional coordinates. By Claim 6.6.1 (below) each tour in OPT has cost at most $4r_{\max}$, thus $\text{cost}(f) \le 4r_{\max}$ for all $f \in \mathcal{F}$ and we have,

$$\sum_{f \in \mathcal{F}} \text{cost}(f)\bar{x}_f \le (\sum_{f \in \mathcal{F}} \text{cost}(f) x^*_f) + c \cdot \text{cost}(f) \le \text{OPT}(R) + c \cdot r_{\max}.$$

Claim 6.6.1. *Let I be an instance of the problem between two depots such that the radius of I is $r_{\max}$, then all tours in OPT(I) have cost at most $4r_{\max}$.*

Proof. For a contradiction assume there is a tour t in OPT(I) such that $\text{cost}(t) > 4r_{\max}$. We replace t with two tours t', t'' that together cover the same customers as t and such that $\text{cost}(t') + \text{cost}(t'') \le 4r_{\max}$, which implies that $t \notin \text{OPT}(I)$.

Define t' to be identical to t until it reaches distance $r_{\max}$ at which point t' returns to the originating depot. Thus $\text{cost}(t') = 2r_{\max}$. Let t'' start at the opposite depot of t and cover the customers of t appearing after distance $r_{\max}$ in t. The $\text{cost}(t'') \le 2r_{\max}$, as every customer covered by t after distance $r_{\max}$ is closer to the opposite depot of t, and the distance of these customers from the opposite depot is at most $r_{\max}$. Thus together t', t'' cover all the customers covered by t and have total cost at most $4r_{\max} \le \text{cost}(t)$. □

6.7 Proof of Lemma 6.3.7

Let A be the set of new tours added by Algorithm 20, and let $|A| = s$ denote the size of A. Each new tour is opened at some position p and connected to the closest depot to p. Let $(d_j)_{s \ge j \ge 1}$ be the distances between the opening position and the closest depot of each new tour in A sorted in increasing order, and define $d_0 = 0$ for convenience. Let A_j be the tours of A whose opening position is at distance at least d_j from its depot. We have:

$$\text{cost}(G) = \text{cost}(T) + 2 \sum_{s \ge j \ge 1} (d_j - d_{j-1})|A_j|. \tag{6.17}$$

Observe that $|A_s| = 1$, this represents the first new tour the algorithm opens. Now we bound $|A_j|$ for $j < s$. Let $(p_x)_{s \ge x \ge 1}$ be the the opening positions of the new tours in A ordered in increasing order (i.e from left to right). Let $p^l_{d_j}$ be the left most opening position which is at distance at least d_j from the left depot, i.e $p^l_{d_j} = min_x\{p_x : p_x \geq d_j\}$. Let $p^r_{d_j}$ denote the right most opening position which is at distance $\geq d_j$ from the right depot i.e $p^r_{d_j} = max_x\{p_x : L - p_x \geq d_j\}$. For each tour $t \in T$ let θ_t denote the depot of t and m_t denote the farthest position from θ_t vistied by t. Let T^l_j be the tours from the left depot that pass beyond position $p^l_{d_j}$ i.e $T^l_j = \{t \in T : \theta_t = \theta_l, m_t \geq p^l_{d_j}\}$ and similarly $T^r_j = \{t \in T : \theta_t = \theta_r, m_t \leq p^r_{d_j}\}$. Denote $T_j = T^l_j \cup T^r_j$ for convenience. Thus T_j is the set of tours that visit the interval between positions $p^l_{d_j}$ and $p^r_{d_j}$ and each tour in T_j has length at least d_j.

Let $cap(j)$ denote the total remaining capacity (before any small demands are covered) of tours in T_j, thus $cap(j) = \sum_{t \in T_j} c_t$. Recall that $S = (w_i, p_i)_i$ is the set of small demands and let $small(j)$ denote the total small demand in positions between $p^l_{d_j}$ and $p^r_{d_j}$,

$$small(j) = \sum_{\substack{(w_i,p_i) \in S \\ p^l_{d_j} \geq p_i \geq p^r_{d_j}}} w_i$$

Algorithm 20 opens a new tour at position p only if no tour passing p has enough remaining capacity to cover a small demand at p. As new tours are opened at positions $p^l_{d_j}$ and $p^r_{d_j}$ and each tour in T_j passes by at least one of these positions, all tours of T_j must have remaining capacity $< \epsilon k$ (i.e less than the maximum size of any small demand). Thus the total amount of small demand assigned by the algorithm to T_j is,

$$\text{(small demand added to } T_j) \;\geq\; cap(j) - |T_j|\epsilon k,$$

and the total amount of small demand remaining between positions $p^l_{d_j}$ and $p^r_{d_j}$ is at most:

$$\text{(small demand remaining between } p^l_{d_j}, p^r_{d_j}) \;\leq\; small(j) - cap(j) + |T_j|\epsilon k \;\leq\; |T_j|\epsilon k. \tag{6.18}$$

where the second inequality in Equation 6.18, follows as constraint (6.4) of the linear program implies that the total small demand located in the interval between any pairs of positions is equal to the remaining capacity of the tours visting the interval. As the algorithm opens a new tour only when no other new tour has enough remaining capacity to cover some small demand, all but at most two tours, one from the right depot and one from the left depot, are almost full and we have:

$$|A_j| \leq \frac{|T_j|\epsilon k}{(1-\epsilon)k} + 2 = \frac{\epsilon |T_j|}{(1-\epsilon)} + 2. \tag{6.19}$$

As $|A_s| = 1$, Equation 6.19 holds for all A_j for $1 \leq j \leq s$. Substituting Equation 6.19 into Equation 6.17 we have,

$$\text{cost}(G) = \text{cost}(T) + \frac{\epsilon}{(1-\epsilon)} \sum_{j \geq 1} 2(d_j - d_{j-1})|T_j| + 4 \max_j d_j.$$

As $\text{cost}(T) \geq \sum_{j \geq 1} 2(d_j - d_{j-1})|T_j|$ and $\max_j d_j \leq r_{\max}$ the maximum radii, we obtain for $\epsilon \leq 1/2$,

$$\text{cost}(G) \leq \frac{\text{cost}(T)}{1-\epsilon} + 2r_{\max} \leq (1+2\epsilon)\text{cost}(T) + 2r_{\max}.$$

Chapter 7

Many Customers and Restricted Capacity

This chapter's results are joint work with Shay Mozes and Claire Mathieu and have appeared in [20].

7.1 Introduction

This chapter presents a PTAS for 1-dimensional Unsplit VRP with a single depot and restricted vehicle capacity.

Recall the problem: Given a positive integer $k \leq n$ denoting the vehicle capacity, a set $C = \{(p_i, w_i)\}_{i \leq n}$ of n customers each with a position p_i on the line and a demand $w_i \leq k$, and a single depot also with a position on the line, find a collection of tours of minimum total length covering the demands of all customers in C, such that each tour starts and ends at the depot, delivers at most k demand and such that no customer's demand is split up among multiple tours.

Please refer to the introduction of Chapter 5 for the background of this problem and to see how it is equivalently viewed as a scheduling problem of minimizing the makespan on a single batch machine with non-identical job sizes and as a generalization of bin-packing.

To the best of our knowledge the restricted capacity setting has not been specifically studied in prior work. However the constraint certainly makes sense in any setting where the total number of items to deliver is large compared to the delivery vehicle, *e.g.* a helicopter delivering goods to land from a large ship containing many items.

As mentioned in the introduction of Chapter 5, bin packing reduces to the Unsplit problem, in any dimensions and any number of depots. Bin packing does not to have a polynomial time approximation algorithm with approximation ratio better than 3/2 unless P=NP, and hence does not admit a PTAS. Thus it is surprising that the Unsplit problem of this chapter can have a PTAS. The explanation for this is the following: the hardness proof that bin packing admits no PTAS is via a reduction to the *Partition* problem [1]. Partition is known to be NP-Hard in general, but can be solved in polynomial time when each integer is at most n (or when input is supplied as unary). Carrying out the reduction from Unsplit with the restricted capacity constraint, that $k \leq n$, results in the polynomially solvable version of Partition, hence the hardness result no longer applies.

Nevertheless, the Unsplit problem with restricted capacity is an NP-Hard problem, as Bin packing with restricted bin size at most n and sizes $w_i \geq 1$ for all i, (or in other words bin packing with unary inputs) reduces to it. As Bin packing is strongly NP-Hard, it is NP-Hard even when the input is supplied in unary [27].

We give a PTAS for the 1-dimensional Unsplit problem with single depot and restricted capacity. Our algorithm runs in polynomial time as long as $k = poly(n)$ (or if k is specified in unary).

Theorem 7.1.1. *Given an instance of the* 1*-dimensional Unsplit problem with a single depot and restricted capacity* $k = poly(n)$*, Algorithm 21 outputs a solution of cost* $(1 + O(\epsilon))OPT$*, in time* $k^{e^{O(1/\epsilon)}} + O(n \log(n)) + \log(n)(1/\epsilon)^{O(1/\epsilon)}$.

We note that, unless P=NP, we cannot hope to achieve a PTAS when the conditions of Theorem 7.1.1 do not hold. A PTAS for the case when $k > poly(n)$ would give us a polynomial time algorithm, rather than a pseudo polynomial time algorithm, for deciding the Partition problem.

Overview of our approach. We partition the instance into two instances *close* and *far* and solve them independently. A similar technique was used for the UnitDem problem on the plane by [29, 7]. The *far* instance is small enough so that it can be solved exactly by dynamic programming. The *close* instance is solved by Algorithm 13 of Chapter 5. The final output is the union of the solutions for *far* an *close* . Figure 7.1 shows a solution computed by our algorithm.

The far instance is solved in polynomial time when $k \leq poly(n)$, using a generalization of the dynamic program of subset sum.

The crux of the analysis is a structural lemma which proves that the instance can be split into *close* and an instance *far* with the right properties. It is crucial that, on the one hand, far does not contain too much demand, so it can be solved efficiently and on the other hand, that far contains enough demand so that the asymptotic condition of Theorem 5.1.1 holds for *close*. The lemma is an extension of the technique of [29] to the unsplittable case.

7.2 Algorithm and Proof of Main Theorem

We present the PTAS in Algorithm 21 and prove Theorem 7.1.1. We assume that the instance is preprocessed as given by Definition 5.2.1. Figure 7.1 shows a solution computed by our algorithm.

[1]Partition: Given a set of integers $S = w_1, \ldots, w_n$, decide if S can be partitioned into two sets S_1 and S_2 such that the sum of the numbers in S_1 and S_2 are equal.

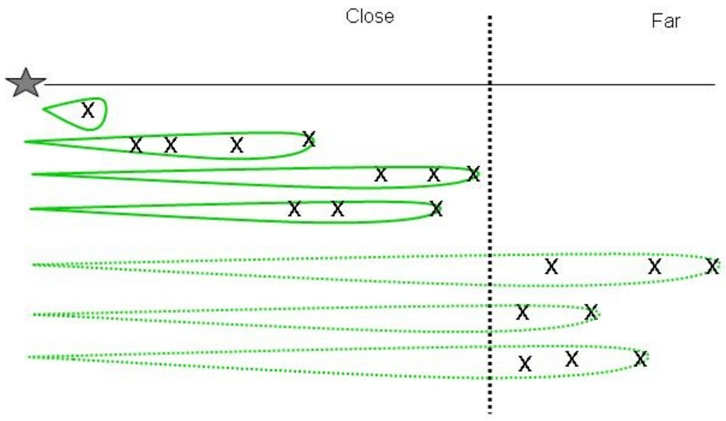

Figure 7.1: A solution computed by Algorithm 21. The star is the depot, the dashed tours cover only customers from far (marked"x"'). The solid tours cover only customers from *close*.

Algorithm 21 PTAS: Unsplit with single depot, resticted capacity, one dimension

```
Input: A Preprocessed instance with vehicle capacity $k$, customers $(p_i, w_i)_{i \leq n}$
Precondition: $k \leq poly(n)$.
1: Partition the instance into close and far using Algorithm 22
2: Find OPT($far$) using Algorithm 23.
3: Find Best(close) using Algorithm 13 of Chapter 5
Output: Best(close) $\cup$ OPT($far$), as the solution for the whole instance.
```

7.2.1 Partitioning into *close* and far Instances

To partition the instance into *close* and far, Algorithm 22 considers the customers in decreasing order of their positions and identifies an index i^*, such that, by condition (i) of Line 4, the total demand appearing at positions greater than the position of i^* is large, but by condition (ii), still bounded. The far instance is defined to contain all the customers with indices less than or equal to i^* and *close* to contain the customers with indices greater than i^*.

Lemma 7.2.1 shows the optimal solution of I can be transformed into separate solutions for *close* and far at small additional cost.

Lemma 7.2.1. *Given an instance I, Algorithm 22 returns two instances* far *and* close *s.t.*

$$OPT(close) + OPT(far) \leq (1 + O(\epsilon))OPT$$

Proof. Assume that *close* is not empty i.e $i^* \neq n$, as otherwise the Lemma holds trivially. Let T be the set of tours of OPT that cover customers in both instances *close* and far. We show how to modify T so each tour covers customers in only one instance. For each tour $t \in T$ cut t at position p_{i^*} to obtain three pieces: the first piece goes from the depot to position p_{i^*} and covers only customers in *close*, the second piece goes further than p_{i^*} and covers customers only in far and the third piece goes from p_{i^*} back to the depot covering only customers in *close*. Concatenate the first and third pieces of together at p_{i^*} to get a tour that covers only customers in the *close* instance. Doing the above for each $t \in T$ yields a set of tours T_1 that cover only customers in *close*.

Algorithm 22 Partition into *close* and far

```
Input: vehicle capacity $k$, customers $(p_i, w_i)_{i \le n}$ s.t. $p_1 \ge \cdots \ge p_n$
1: if  the total demand i.e $\sum_{i \le n} w_i = e^{O(1/\epsilon)} k$  then
2:     Let $i^* = n$.
3: else
4:     Let $i^*$ be the minimum index such that:
           i.   $\sum_{j \le i^*} w_j \ge k/\epsilon^6$ and
           ii.  $p_{i^*} \sum_{j \le i^*} w_j \le \epsilon \sum_{j=1}^{n} w_j p_j$.
5: end if
6: Far: Let the $far$ instance consist of the customers indexed by $1, \dots, i^*$.
7: Close: Let the close consist of the remaining customers $i^* + 1, \dots n$.
Output: Instances $far$ and close
```

Let T_2 be the set of second pieces of each tour in T. While there exists at least two pieces in T_2 each covering at most $k/2$ demand, concatenate the pieces together at p_{i^*} into a new piece covering at most k demand. After all concatenations are done, all but at most one piece in T_2 covers at least $k/2$ demand. Add a single round trip connection from p_{i^*} to the depot for each piece in T_2 to obtain tours covering that cover only customers in far.

The total cost of $T_1 \cup T_2$ is the cost of T plus the cost of the round trips to the depot. Let $\text{dem}(i^*) = \sum_{j \le i^*} w_j$. Since at most one tour in T_2 covers $< k/2$ demand, the number of tours in T_2 is at most $\frac{\text{dem}(i^*)}{(k/2)+1} + 1 < \frac{2\text{dem}(i^*)}{k}$, which follows as $\text{dem}(i^*) \ge k$ by condition (i) of Line 4. Thus the total cost of the round trips required for T_2, is at most $2p_{i^*}(2\text{dem}(i^*)/k)$. Thus

$$\text{OPT}(close) + \text{OPT}(far) \le \text{OPT} + \frac{4(p_{i^*})\text{dem}(i^*)}{k}$$

By Line 4 condition (ii), $(p_{i^*})\text{dem}(i^*) \le \epsilon \sum_{j=1}^{n} w_j p_j$, so applying the Rad bound (Lemma 1.3.2) completes the proof as,

$$\frac{4p_{i^*}\text{dem}(i^*)}{k} \le 4\epsilon \sum_j \frac{w_j p_j}{k} \le 2\epsilon \text{OPT}.$$

□

7.2.2 Solving the far instance

Algorithm 23 is used to solve far and its correctness is proved by Lemma 7.2.4. The Algorithm is based on Lemma 7.2.2, that choosing far as in Algorithm 21 implies that OPT(far) uses only a constant number of tours. The proofs of Lemma 7.2.2 appears in Section 7.3 and proof of Lemma 7.2.4 in Section 7.4.

Lemma 7.2.2. *For any instance, let far be selected as in Algorithm 21. Then OPT(far) uses at most $c_{far} = O(1)$ tours.*

Given that c_{far} is a constant, Algorithm 23 solves far by enumerating all configurations of solutions for far, checking feasibility, and returning the minimum cost feasible configuration. Lemma 7.2.4 shows that Algorithm 23 computes the optimal far solution.

Configurations of far. We define configurations to describe solutions of far concisely. The configuration considers the c_{far} tours of OPT(far), in decreasing order of their maximum positions $m_1 \geq m_2 \geq \ldots \geq m_{c_{far}}$. The positions naturally define c_{far} intervals where interval I_i consists of customers between positions $(m_{i+1}, m_i]$, where we define $m_{c_{far}+1} = p^*$ for convenience. The customers in I_i can only be covered by tours $j = 1, \ldots, i$, i.e those with maximum position at least m_i. The list D^i specifies how the demands in I_i is partitioned among these tours.

Definition 7.2.3. *Let OPT(far) use c_{far} tours. A* configuration *of far consists of:*

- *An ordered list of positions $m_1 \geq m_2 \ldots \geq m_{c_{far}}$ s.t. m_j is the maximum position of tour j.*
- *For each $i \in [1, c_{far}]$, a list D^i of i numbers $D^i = \{d_1^i, \ldots d_i^i\}$, such that d_j^i is the amount of demand picked up by the j-th tour in interval $I_i = (m_{i+1}, m_i]$, where for convenience we use $m_{c_{far}+1} = p_{i^*}$.*

The cost *of the configuration is $\sum_{j \leq c_{far}} 2r_j$.*

Note that D^i does not directly describe how to partition the demands among the tours in I_i. Determining whether there exists a partition that is consistent with a configuration, is done in $k^{e^{O(1/\epsilon)}}$ time (i.e., polynomial in n assuming $k \leq poly(n)$) using a trivial extension of the dynamic program for the subset sum problem. See proof of Lemma 7.2.4 for more details.

Algorithm 23 Solving the far instance

Input: far customers $(p_s, w_s)_s$ with $\sum_s w_s = ke^{O(1/\epsilon)}$

```
1: for each configuration $f$ of $far$ as given in Definition 7.2.3 do
2:     for each tour $j \leq c_{far}$ do
3:         if (Load on tour $j$) $= \sum_{i \leq c_{far}} d_j^i$, is $> k$ then
4:             Mark $f$ infeasible, as the capacity of tour is exceeded.
5:         end if
6:     end for
7:     for each interval $i \leq c_{far}$, with demand dem($I_i$) $= \{w_s : (w_s, p_s) \in I_i\}$ do
8:         if Extended DP of subset sum cannot partition dem($I_i$) into $d_1^i, \ldots, d_i^i$ then
9:             Mark $f$ infeasible, as demands in $I_i$ cannot be partitioned as required by $f$.
10:        end if
11:    end for
12: end for
```

Output: Solution represented by the minimum cost configuration, not marked infeasible.

Lemma 7.2.4. *Given an instance of far with demand $ke^{O(1/\epsilon)}$, Algorithm 23 finds the optimal solution of far in time $k^{e^{O(1/\epsilon)}}$.*

7.2.3 Proof of Main Theorem 7.1.1

Correctness of Algorithm 21. By Lemma 7.2.1 OPT(far) plus OPT(*close*) is a near optimal solution of the original instance. It remains to show that Algorithm 21 computes near optimal solutions for both far and *close*. Lemma 7.2.4 proves that Algorithm 23 computes the optimal solution of the far instance.

We focus on cost of solution for *close*. Using the notation from the proof of Theorem 5.1.1, let P^* be the partition of *near* for which Lemma 5.3.3 holds and let $R_1^*, R_2^*, \ldots, R_r^*$ be the regions of P^*. It remains to show that Algorithm 13 finds a near optimal solution for each region of partition P^*. Applying the same argument as in the proof of Theorem 5.1.1 we can show that Algorithm 13 finds a solution of cost at most,

$$\sum_{i\le r}(1+2\epsilon)\mathrm{OPT}(R_i^*) + 2p_{R_i^*} = (1+O(\epsilon))\mathrm{OPT}(\mathit{close}) + \ell(\epsilon)\sum_{i\ge 0} p_{R_i^*} \tag{7.1}$$

where $\ell(\epsilon) = O(1/\epsilon^5)$ as defined in the proof of Theorem 5.1.1. We analyze the last term of Equation 7.1. The farthest customer in *close* is at position $\le p_{i^*}$, and by definition of regions (Definition 5.3.2) the farthest position of a region R_i^* is $p_{R_i^*} = p_{i^*}\epsilon^{i/\epsilon}$. Thus,

$$\sum_{i\ge 0} p_{R_i^*} \le \sum_{i\ge 0} p_{i^*}\epsilon^{i/\epsilon} \le 2p_{i^*} \le \epsilon^6\mathrm{OPT} \tag{7.2}$$

where the last inequality follows by Lemma 7.2.5 (given below). Thus combining Equations 7.2 and 7.1 and value of $\ell(\epsilon)$ we have that our solution of *close* has cost at most

$$\begin{aligned}(1+O(\epsilon))\mathrm{OPT}(\mathit{close}) + \ell(\epsilon)\sum_{i\ge 0} p_{R_i^*} &\le (1+O(\epsilon))\mathrm{OPT} + O(1/\epsilon^5)\cdot\epsilon^6\mathrm{OPT} \\ &= (1+O(\epsilon))\mathrm{OPT}(\mathit{close}) + O(\epsilon)\mathrm{OPT} \qquad (7.3)\end{aligned}$$

Thus the cost of the solution output by Algorithm 21 is at most

$$\mathrm{OPT}(far) + (1+O(\epsilon))\mathrm{OPT}(\mathit{close}) + O(\epsilon)\mathrm{OPT},$$

which is $(1+O(\epsilon))$OPT by Lemma 7.2.1.

Lemma 7.2.5. *OPT* $> 2p_{i^*}/\epsilon^6$.

Proof. By definition of i^*, $\sum_{j\le i^*} w_j \ge k/\epsilon^6$. Using Lemma 1.3.2 we have

$$\mathrm{OPT} \ge 2\sum_{j\le i^*}\frac{w_j p_j}{k} \ge 2\sum_{j\le i^*}\frac{w_j p_{i^*}}{k} \ge 2\cdot\frac{k}{\epsilon^6}\frac{p_{i^*}}{k}.$$

□

The running time. By Lemma 7.2.4 far can be computed by Algorithm 23 in time $k^{e^{O(1/\epsilon)}}$. By Theorem 5.1.1 Algorithm 13 finds a solution for the *close* instance in time $O(n\log n/\epsilon) + O(\log n\cdot(1/\epsilon)^{O(1/\epsilon)})$. Thus the running time of Algorithm 21 is $k^{e^{O(1/\epsilon)}} + O(n\log n/\epsilon) + O(\log n\cdot(1/\epsilon)^{O(1/\epsilon)})$.

7.3 Proof of Lemma 7.2.2

The proof follows from Lemmas 7.3.1 and 7.3.2. Lemmas 7.3.1 shows that the total demand in far is $O(k)$ and Lemma 7.3.2 shows that as we can assume that all but one tour in OPT(far) covers at least $k/2$ demand, $c_{far} = 0(1)$.

Lemma 7.3.1 is an extension of a technique used by Haimovich and Rinnooy Kan's [29] for the UnitDem problem where the demands are splittable. It bounds the total demand in the far instance by bounding the number of demands that violate the requirement $p_{i^*} \sum_{j \le i^*} w_j \le \epsilon \sum_{j=1}^{n} w_j p_j$ (Line 4, cond(ii) of Algorithm21).

Lemma 7.3.1. *For an instance I, let i^* be as chosen as in Algorithm 22. Then*

$$\sum_{j \le i^*} w_j = e^{O(1/\epsilon)} k = O(k)$$

Proof. Let the customers be sorted in decreasing order of their positions as in the input of Algorithm 21. Assume that the total demand in the instance is more than $e^{O(1/\epsilon)}k$, as otherwise the Lemma holds even if far contains the entire instance.

Algorithm 21 searches for the minimum index in the sorted list of customers where the conditions of Line 4 are satisfied. Let i_0 be minimum index satisfying condition (i) of Line 4, i.e $\sum_{j \le i_0} w_i \ge k/\epsilon^6$. Either the algorithm choose $i^* = i_0$ or not. If $i^* = i_0$ then as each demand has size at most k the total demand in far is at most $k/\epsilon^6 + k$. Thus the lemma holds.

Now suppose $i^* \ne i_0$. As condition (i) is satisfied at i_0 it continues to be satisfied for all indices $i > i_0$. Thus while i^* has not been reached, i.e for every index $i \in [i_0, i^*)$, it must be that condition (ii) of Line 4 is unsatisfied. Thus we have, $(w_1 + \cdots + w_i)p_i > \epsilon \sum_j w_j p_j$. Equivalently by rearranging the equation and multiplying both sides by w_i we have,

$$\forall i \in [i_0, i^*), \quad \frac{1}{\epsilon} \frac{w_i p_i}{\sum_j w_j p_j} > \frac{w_i}{w_1 + \cdots + w_i}.$$

Summing over all $i \in [i_0, i^*)$ implies

$$\frac{1}{\epsilon} > \sum_{i \in [i_0, i^*)} \frac{w_i}{w_1 + \cdots + w_i}. \tag{7.4}$$

Go through the sequence $(w_i)_{i_0 \le i < i^*}$ in order of increasing i, and greedily partition the w_i's into groups $g_1, g_2, \ldots$ such that for every group g, except perhaps the last one, the weight of demands in the group is between k/ϵ and $k(1/\epsilon + 1)$, i.e $k/\epsilon \le \sum_{i \in g} w_i < k(1/\epsilon + 1)$. Letting $W_0 = w_1 + \cdots + w_{i_0 - 1}$ and, for group g_ℓ, $W_\ell = \sum_{i \in g_\ell} w_i$, we can rewrite the right hand side of Equation 7.4 as

$$\begin{aligned} \sum_{\ell \ge 1} \sum_{i \in g_\ell} \frac{w_i}{W_0 + \cdots + W_{\ell-1} + \sum_{i' \in g_\ell, i' \le i} w_{i'}} &\ge \sum_{\ell \ge 1} \frac{1}{W_0 + \cdots + k_{\ell-1} + W_\ell} \sum_{i \in g_\ell} w_i \\ &= \sum_{\ell \ge 1} \frac{W_\ell}{W_0 + \cdots + W_\ell}. \end{aligned} \tag{7.5}$$

Since all W_ℓ's (except possibly the last one) are within a $(1+\epsilon)$ factor of each other, Equation 7.5 is at least

$$\frac{1}{1+\epsilon}\sum_{\ell\geq 1}\frac{1}{\ell+1} \;\geq\; \frac{1}{2}\log(\#(\text{groups})). \tag{7.6}$$

Replacing the right hand side of Equation 7.4 with Equation 7.6, we have $1/\epsilon > \log(\#(\text{groups})/2$, which implies that the number of groups is at most $\exp(2/\epsilon)$ (plus possibly one more to account for the last group).

Thus after going through all the indices of demands in groups $\{W_\ell\}_{\ell\geq 0}$ Equation 7.4 cannot be satisfied, hence condition (ii) of Line 4 must hold. Since group W_0 has demand at most $k(1/\epsilon^6+1)$ and the remaining $\exp(2/\epsilon)$ groups have total demand at most $k(1/\epsilon+1)$, the total demand in positions greater than i^* is $\exp(O(1/\epsilon))k$, as desired. $\square$

Lemma 7.3.2. *Let I be an instance of the problem such that the sum of all the demands in I is D. Then OPT uses at most $\lceil 2D/k\rceil$ tours.*

Proof. While there are two tours in OPT that each cover $\leq k/2$ demand, merge them together at the depot. Thus without loss of generality we can assume that all but at one tour in OPT covers at least $k/2+1$ demand. $\square$

7.4 Proof of Lemma 7.2.4

Correctness of Algorithm 23 The algorithm iterates through all possible configurations, checks the feasibility of each, and returns the feasible configuration with minimum cost. Fix a configuration f. In each interval I_i, d^i_j specifies the total demand assigned to tour j from interval I_i. Thus the total load for tour j over all intervals is given by $\sum_{i\leq c_{far}} d^i_j$. Line 3 computes the load of each tour in the configuration and verifies that is at most k.

Let $\text{dem}(i)=\{w_1,\dots w_m\}$ be the demands in interval I_i. Line 8 can use the following generalization of the dynamic program of subset-sum to check that the demands in I_i can be partitioned into $D^i=\{d^i_1,d^i_2,\dots,d^i_i\}$. The dynamic program populates a table Q. Table element $Q[s,d_1,\dots d_i]$ specifies whether the first s demands from I_i, $w_1\dots w_s$, can be partitioned into i sets whose sums are $d_1\dots d_i$. The table is populated using the following recurrence: $Q[s,d_1,\dots d_i]$ is true if any of the following are true: $Q[s-1,d_1-w_s,d_2,\dots d_i]$, $Q[s-1,d_1,d_2-w_s,d_3,\dots d_i]$, $Q[s-1,d_1,d_2,d_3-w_s\dots d_i]$, $\dots$, $Q[s-1,d_1,d_2,\dots,d_i-w_s]$. For the base case: $Q[1,d_1,\dots,d_i]$ is true if $w_1=d_t$ for some $t\leq i$ and all the other $d_{t'}=0$ for all $t'\neq t$. Otherwise $Q[1,d_1,\dots d_i]$ is false. We are interested in the entry $Q[m,d_1,\dots,d_i]$ which specifies whether all the demands in $\text{dem}(i)$ can be partitioned according to D^i.

Running Time. Algorithm 23 iterates over all configurations of far, so we start by analyzing the number of configurations there are. By Lemma 7.3.1 the total demand in far is $ke^{O(1/\epsilon)}$, thus by Lemma 7.3.2, the number of tours $c_{far} = e^{O(1/\epsilon)}$. There are at most $ke^{O(1/\epsilon)}$ positions with customers, thus the number of possible lists of maximum positions $m_1, m_2, \ldots, m_{c_{far}}$ is (# position)$^{c_{far}} = k^{e^{O(1/\epsilon)}}$. Each interval $i \leq c_{far}$ contains at list D^i of i numbers, thus overall there are at most c_{far}^2 numbers in these lists. As each number is less than or equal to k the number of possible lists $D^1, \ldots, D^{c_{far}}$ is $k^{c_{far}^2} = k^{e^{O(1/\epsilon)}}$. Therefore, the total number of configurations is $k^{e^{O(1/\epsilon)}}$, which is a polynomial in n when $k \leq poly(n)$.

Next we analyze the time required to verify the feasibility of a configuration. Line 3 takes c_{far} time as it involves summing at c_{far} values. For Line 8 we analyze the runtime of the generalization of the dynamic program for subset sum. Table Q has size (# possible s)(# possible $d_1, \ldots, d_i$) $= ke^{O(1/\epsilon)} \cdot k^{c_{far}}$, as there are at most $We^{O(1/\epsilon)}$ demands and c_{far} tours in interval each I_i, and as each $d_i \leq k$. Each entry in Q can be computed in constant time by looking up at most $c_{far}+1$ entries. Thus the dynamic program for Line 8 is computed in time $O(c_{far} \cdot ke^{O(1/\epsilon)} \cdot k^{c_{far}}) = k^{e^{O(1/\epsilon)}}$. Lines 3 and 8 are each executed c_{far} times per configuration, thus the total time to verify a configuration is $c_{far}(c_{far} + k^{e^{O(1/\epsilon)}}) = k^{e^{O(1/\epsilon)}}$.

To verify all configurations the total running time of Algorithm 23 is

$$k^{e^{O(1/\epsilon)}} \cdot k^{e^{O(1/\epsilon)}} = k^{e^{O(1/\epsilon)}}.$$

Chapter 8

Conclusion and Open Questions

We designed quasipolynomial time approximation schemes for the Euclidean UnitDem problem and asymptotic polynomial time approximation schemes for the 1-dimensional Unsplit problem. Our algorithms work for the setting with single or multiple depots. Thus we provided support for the thesis statement: *Arbitrarily good approximation guarantees are possible by exploiting geometric properties in vehicle routing problems.*

Our UnitDem algorithm has quasipolynomial and seriously super-polynomial running time, so it is unlikely to lead to much in the way of practical improvements. However our result does make progress towards and provides stronger evidence for the long standing conjecture that Euclidean UnitDem has a PTAS for all k. Recently Adamaszek et al. designed a PTAS for a larger range of k using our QPTAS as a black box [1].

Our approximation schemes for the Unsplit problem are reasonably efficient and simple and we believe these can be useful in practice, especially the single depot algorithm which has applications to batch scheduling.

Open questions for UnitDem The major open question still remaining is: *Is there a polynomial time approximation scheme for the Euclidean UnitDem problem for all values of k?* As noted previously, PTAS already exist when k is either large or small compared to n [1, 7, 29, 1, 2], and several authors including [1, 7] have also noted that $k = \Theta(\sqrt{n})$ seems to be the difficult case to getting a PTAS for all k.

There are several obstacles to reducing the running time of our QPTAS. We describe a tour segment by its pair of portals and the threshold number of points it covers. As there are $O(\log n)$ portals and $O(\log^2 n)$ thresholds possible we get a polylogarithmic number of tour profiles. The quasipolynomial running time of our method follows as there can be $O(n)$ tour segments of each profile. To reduce the number of tour profiles we seem to require a result showing the existence of a near optimal solution that uses a small number of portals in each box. We also need to be able to reduce the number of thresholds while still maintaining that each tour covers no more than $(1+\epsilon)k$ points. A few factors of our quasipolynomial running time also comes from accumulating cost over the $O(\log n)$ levels of the randomized dissection so a more global accounting of the cost may also be useful to reducing the running time.

Despite these challenges obtaining a PTAS for the UnitDem problem would be an exciting result. A related setting that also may provide some insights for this is the average case setting of the problem where the locations of customers and depots are independently and identically distributed. An open question is: *Is there a polynomial time approximation scheme for the average case setting of the Euclidean UnitDem problem*? Haimovich and Rinnooy Kan showed that on the Euclidean plane, when either $k = o(\sqrt{n})$ or $k = \Theta(n)$, their tour partitioning algorithm (Alg. 8) is asymptotically optimal [29]. Recently, Bompadre, Dror and Orlin showed that in average case the tour partitioning algorithm is a $(2+\epsilon) - c$ approximation [11] for $c \geq 0.015$ which is a slight improvement over the worst case setting where the approximation factor is known to be $2+\epsilon$. Thus the average case setting seems to be an easier problem to design a PTAS for and could also provide new ideas that may extend to the worst case setting. Note that Karp's work on the average case Euclidean TSP [47] inspired some of key ideas in the PTAS for the worst case setting of the problem. Finding a PTAS for average case UnitDem would have important applications as well since the average case model applies in many practical settings.

The *fixed fleet* problem is a common variant of UnitDem where the input also includes the fleet size f and the customers are required to be covered using at most f routes. This models scenarios where the tours must occur simultaneously; for example bus routes for school children. We conjecture that it should be possible to extend our QPTAS for UnitDem when the fixed fleet constraint is *soft*: if OPT is the value of the optimal solution that is constrained to use at most f tours, then our approach can be extended to construct a solution that uses at most $f(1+O(\epsilon))$ tours of total length $(1+O(\epsilon))$OPT. This would require modifying the dynamic program in our algorithm to find the optimal black solution that uses at most f tours. As we only drop an $O(\epsilon)$ fraction of the total points from the black tours (i.e. the red points) we should be able to cover them with at most $O(\epsilon)f$ tours using the tour partitioning algorithm. An interesting open question is: *Is there a polynomial time approximation scheme for the fixed fleet version of the Euclidean UnitDem problem*? Arora's Euclidean TSP algorithm yields a PTAS for the problem whenever the optimal solution consists of a constant number of tours i.e if $k = \Omega(n)$ or if $f = O(1)$.

Open questions for Unsplit We designed asymptotic polynomial time approximation schemes for the 1-dimensional Unsplit problem with running time exponential in $1/\epsilon$. We conjecture that it is possible to get a fully polynomial time by using the column generation technique of Karp and Karmarkar [33]. This technique was also employed by Epstein and Levin to achieve fully polynomial time for a different variant of the bin packing problem [25].

It would also be wonderful to extend our algorithms to the Euclidean setting. The main open question is: *Is there an asymptotic polynomial time approximation scheme for the Euclidean Unsplit problem*? We note that in 2-dimensions it is still possible to partition the instance into regions and use a shifting argument to show that a near optimal solution uses tours that cover customers from only one region. In fact a similar technique was used in [1] for the UnitDem problem on the plane. We also note that Bramel et al. give a probabilistic analysis for the Euclidean plane for the average case setting where customer demands are drawn i.i.d from any distribution [13].

Bibliography

[1] A. Adamaszek, A. Czumaj, and A. Lingas. PTAS for k-tour cover problem on the plane for moderately large values of k. In *ISAAC*, Berlin, Heidelberg, 2009. Springer-Verlag. [6, 25, 47, 59, 95, 97]

[2] Sanjeev Arora. Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. *J. ACM*, 45(5):753–782, 1998. [v, 3, 6, 10, 11, 13, 20, 24, 25, 95]

[3] Sanjeev Arora. Approximation schemes for NP-hard geometric optimization problems: A survey. *Mathematical Programming*, 97(1-2):43–69, 2003. [10, 11, 25, 26]

[4] Sanjeev Arora and George Karakostas. Approximation schemes for minimum latency problems. *SIAM J. Comput.*, 32(5):1317–1337, 2003. [10]

[5] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and the hardness of approximation problems. *J. ACM*, 45(3):501–555, 1998. [10]

[6] Tetsuo Asano, Naoki Katoh, Hisao Tamaki, and Takeshi Tokuyama. Covering points in the plane by k-tours: a polynomial approximation scheme for fixed k. Research Report RT0162, IBM Tokyo Research Laboratory, 1996. [25, 47]

[7] Tetsuo Asano, Naoki Katoh, Hisao Tamaki, and Takeshi Tokuyama. Covering points in the plane by k-tours: towards a polynomial time approximation scheme for general k. In *STOC '97: Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 275–283, New York, NY, USA, 1997. ACM. [6, 25, 32, 43, 47, 87, 95]

[8] Meral Azizoglu and Scott Webster. Scheduling a batch processing machine with non-identical job sizes. *International Journal of Production Research*, 38:2173–2184, June 2000. [58]

[9] B. S. Baker. Approximation algorithms for NP-complete problems on planar graphs. *J. ACM*, 41(1):153–180, 1994. [59, 61]

[10] N. Bansal, J. R. Correa, C. Kenyon, and M. Sviridenko. Bin packing in multiple dimensions: Inapproximability results and approximation schemes. *Math. Oper. Res.*, 31(1):31–49, 2006. [58]

[11] A. Bompadre, M. Dror, and J.B. Orlin. Probabilistic analysis of unit-demand vehicle routing problems. *Journal of Applied Probability*, 44(1):259–278, 2007. [96]

[12] Glencora Borradaile, Philip N. Klein, and Claire Mathieu. A polynomial-time approximation scheme for Euclidean steiner forest. In *FOCS '08: Proceedings of the 2008 49th Annual IEEE Symposium on Foundations of Computer Science*, pages 115–124, Washington, DC, USA, 2008. IEEE Computer Society. [10, 50]

[13] J. Bramel, Jr. Coffman, Edward G., P. W. Shor, and D. Simchi-Levi. Probabilistic analysis of the capacitated vehicle routing problem with unsplit demands. *Oper. Res.*, 40:1095–1106, November 1992. [57, 97]

[14] S. Cardon, S. Dommers, C. Eksin, R. Sitters, A. Stougie, and L. Stougie. A PTAS for the multiple depot vehicle routing problem. *SPOR Reports*, January 2008. available at www.win.tue.nl/bs/spor/. [6, 47]

[15] N. Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. *In J. F. Traub, editor, Algorithms and Complexity: New Directions and Recent Results*, page 441, 1976. [10]

[16] E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson. *Approximation algorithms for bin packing: a survey*, pages 46–93. PWS Publishing Co., Boston, MA, USA, 1997. [58]

[17] Janos Csirik, David S. Johnson, and Claire Kenyon. Better approximation algorithms for bin covering. In *SODA '01: Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, pages 557–566, Philadelphia, PA, USA, 2001. Society for Industrial and Applied Mathematics. [58]

[18] Artur Czumaj and Andrzej Lingas. A polynomial time approximation scheme for Euclidean minimum cost k-connectivity. In *ICALP '98: Proceedings of the 25th International Colloquium on Automata, Languages and Programming*, pages 682–694, London, UK, 1998. Springer-Verlag. [46]

[19] G. B. Dantzig and J. H. Ramser. The truck dispatching problem. *Management Science*, 6(1):80–91, 1959. [1]

[20] A. Das, C. Mathieu, and S. Mozes. The train delivery problem-vehicle routing meets bin packing. In *In Proc. WAOA'08, th Workshop on Approximation and Online Algorithms, September 2010.* Springer, 2010. [56, 71, 86]

[21] Aparna Das and Claire Mathieu. A quasi-polynomial time approximation scheme for Euclidean capacitated vehicle routing. In *SODA '10: Proceedings of the twenty first annual ACM-SIAM symposium on Discrete algorithms*, Philadelphia, PA, USA, 2009. Society for Industrial and Applied Mathematics. [25, 47]

[22] Adrian Dumitrescu and Joseph S. B. Mitchell. Approximation algorithms for tsp with neighborhoods in the plane. *J. Algorithms*, 48(1):135–159, 2003. [10]

[23] L. Dupont and F. Jolai Ghazvini. Minimizing makespan on a single batch processing machine with non-identical job sizes. *European Journal of Automation Systems*, 32:431–440, 1998. [58]

[24] Lionel Dupont and Clarisse Dhaenens-Flipo. Minimizing the makespan on a batch machine with non-identical job sizes: an exact procedure. *Computers and Operations Research*, 29(7):807 – 819, 2002. [58]

[25] L. Epstein and A. Levin. An APTAS for generalized cost variable-sized bin packing. *SIAM J. Comput.*, 38:411–428, April 2008. [58, 96]

[26] W. Fernandez de la Vega and Lueker G. S. Bin packing can be solved within $1+\epsilon$ in linear time. *Combinatorica*, 1(4):349–355, 1981. [7, 58, 60, 65, 72, 75]

[27] M. R. Garey and D. S. Johnson. " strong " NP-completeness results: Motivation, examples, and implications. *J. ACM*, 25(3):499–508, 1978. [3, 57, 87]

[28] B. Golden, S. Raghavan, and E. Wasil. *In The vehicle routing problem: Latest Advances and New Challenges.*, volume 43 of *Operations Research/Computer Science Interfaces Series.* Springer, USA, 2008. [1]

[29] M. Haimovich and A. H. G. Rinnooy Kan. Bounds and heuristics for capacitated routing problems. *Mathematics of Operations Research*, 10(4):527–542, Nov., 1985. [v, 3, 6, 7, 25, 26, 27, 30, 32, 36, 40, 43, 47, 87, 92, 95, 96]

[30] M. Haimovich, A.H.G. Rinnooy Kan, and L. Stougie. Analysis of heuristics for vehicle routing problems. In *Vehicle Routing: Methods and Studies. Management Sci. Systems.*, volume 16, pages 47–61, North Holland, Amsterdam, 1988. Elsevier Science B.V. [2, 6, 7, 57]

[31] D. S. Hochbaum and W. Maass. Approximation schemes for covering and packing problems in image processing and vlsi. *J. ACM*, 32(1):130–136, 1985. [61]

[32] Howard Karloff. How long can a Euclidean traveling salesman tour be? *SIAM J. Discrete. Math.*, 2(1):91–99, 1989. [46]

[33] N. Karmarkar and R. M. Karp. An efficient approximation scheme for the one-dimensional bin-packing problem. In *FOCS*, pages 312–320, Washington, DC, USA, 1982. IEEE Computer Society. [58, 96]

[34] Philip N. Klein. A linear-time approximation scheme for tsp in undirected planar graphs with edge-weights. *SIAM Journal on Computing*, 37(6):1926–1952, 2008. [3]

[35] Stavros G. Kolliopoulos and Satish Rao. A nearly linear-time approximation scheme for the Euclidean k-median problem. *SIAM J. Comput.*, 37(3):757–782, 2007. [10]

[36] Martine Labbé, Gilbert Laporte, and Hlne Mercure. Capacitated vehicle routing on trees. *Operations Research*, 39(4):616–622, 1991. [57]

[37] Gilbert Laporte. What you should know about the vehicle routing problem. *Naval Research Logistics*, 54(8):811–819, 2007. [1]

[38] C.L. Li and D. Simchi-Levi. Worst-Case Analysis of Heuristics for Multidepot Capacitated Vehicle Routing Problems. *INFORMS*, 2(1):64–73, 1990. [2, 6, 47, 48, 52, 53]

[39] J. Malkevitch. Keep on trucking. *Monly Essays on Mathematical Topics*, 2010. [1]

[40] J.S.B. Mitchell. Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric tsp, k-mst, and related problems. *SIAM J. Comput.*, 28(4):1298–1309, 1999. [3, 10, 25]

[41] F.D. Murgolo. An efficient approximation scheme for variable-sized bin packing. *SIAM J. Comput.*, 16(1):149–161, 1987. [58]

[42] Christos H. Papadimitriou and Kenneth Steiglitz. Some complexity results for the traveling salesman problem. In *STOC '76: Proceedings of the eighth annual ACM symposium on Theory of computing*, pages 1–9, New York, NY, USA, 1976. ACM. [10]

[43] N. Rafiee Parsa, B. Karimi, and A. Husseinzadeh Kashan. A branch and price algorithm to minimize makespan on a single batch processing machine with non-identical job sizes. *Computers and Operations Research*, 37(10):1720 – 1730, 2010. [58]

[44] Janice Partyka and Randolph. Hall. Vehicle routing software survey- on the road to connectivity. *OR/MS Today*, 37, February 2010. [2]

[45] Satish B. Rao and Warren D. Smith. Approximating geometrical graphs via "spanners" and "banyans". In *STOC '98: Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 540–550, New York, NY, USA, 1998. ACM. [11, 46]

[46] Jan Remy and Angelika Steger. A quasi-polynomial time approximation scheme for minimum weight triangulation. In *STOC '06: Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, pages 316–325, New York, NY, USA, 2006. ACM. [10]

[47] Karp R.M. *Probabilistic analysis of partitioning algorithms for the TSP in the plane*, volume 2 of *Math. Oper. Res.*, pages 209–224. 1977. [96]

[48] Daniel J. Rosenkrantz, Richard Edwin Stearns, and Philip M. Lewis II. An analysis of several heuristics for the traveling salesman problem. *SIAM J. Comput.*, 6(3):563–581, 1977. [9]

[49] Sartaj Sahni and Teofilo Gonzalez. P-complete approximation problems. *J. ACM*, 23(3):555–565, 1976. [3]

[50] Paolo Toth and Daniele Vigo, editors. *The vehicle routing problem.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2001. [1, 2]

[51] R. Uzsoy. Scheduling a single batch processing machine with non-identical job sizes. *International Journal of Production Research*, 32:1615–1635, July 1994. [58]

[52] Vijay V. Vazirani. *Approximation algorithms.* Springer-Verlag New York, Inc., New York, NY, USA, 2001. [45]

[53] WEBSITE. http://neo.lcc.uma.es/radi-aeb/webvrp/, 2010. [3]

[54] WEBSITE. http://researchandpractise.com/vrp/surveys.html, 2010. [3]

[55] Guochuan Zhang, Xiaoqiang Cai, C.-Y Lee, and C.K Wong. Minimizing makespan on a single batch processing machine with nonidentical job sizes. *Naval Research Logistics*, 48:226–240, 2001. [6, 58]

[56] Yuzhong Zhang and Zhigang Cao. An asymptotic PTAS for batch scheduling with nonidentical job sizes to minimize makespan. In *COCOA'07: Proceedings of the 1st international conference on Combinatorial optimization and applications*, pages 44–51, Berlin, Heidelberg, 2007. Springer-Verlag. [58]