Abstract of "Approximation Algorithms for Capacitated Facility-Placement and Vehicle-Routing Problems in Transportation Networks" by Amariah Becker, Ph.D., Brown University, .

Facility-placement and route-planning optimization problems aim to minimize the cost of providing a service to a set of clients. These problems arise frequently in a variety of contexts across computer science and operations research. Some of the most classic and well-studied of these including k-Center and the Traveling Salesperson Problem (TSP), though versatile, fail to adequately model the fact that actual service-providers have limited capacities. Facilities and vehicles have limited size (e.g. hospitals and buses can only hold so many people) as well as limited resources (e.g. staff, supplies, fuel, fleet size etc.), so are often unable to serve all clients at once. Most placement and route-planning algorithms cannot easily be extended to accommodate these considerations without drastic sacrifices to performance.

In this thesis, we present approximation algorithms that provide guaranteed near-optimal solutions to these types of planning problems. Moreover, our algorithms are specifically designed for metrics that model transportation networks: planar and bounded-genus graphs, graphs with bounded treewidth, and graphs with bounded highway dimension. We capitalize on the structure of these metrics to improve the performance of our algorithms.

We first consider Capacitated Domination problems, which aim to minimize the number of facilities required to serve all clients without exceeding facility capacities. For fixed capacities, we present an approximation scheme for planar graphs. Next, we address Capacitated Vehicle Routing, in which vehicles can only visit a limited number of clients in each trip. For arbitrary capacities, we improve the best-known approximation ratio for vehicle routing in trees. For fixed capacities, we design the first approximation schemes for vehicle routing in planar graphs and graphs of bounded highway dimension. Finally, we present a framework for designing vehicle-routing approximation schemes in tree metrics that can be customized to address many different vehicle-routing problems. In particular, the framework accommodates such constraints as vehicle distance (e.g. those imposed by fuel limits or driver fatigue), vehicle capacity, fleet size, and client regret.

Approximation Algorithms for Capacitated Facility-Placement and Vehicle-Routing Problems in Transportation Networks

by

Amariah Becker

B. A., Carleton College, 2011

Sc. M., Brown University, 2016

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

This dissertation by Amariah Becker 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 _____ _____
Philip Klein, Director

Recommended to the Graduate Council

Date _____ _____
Eli Upfal, Reader

Date _____ _____
Claire Mathieu, Reader

Date _____ _____
Renato Werneck, Reader

Approved by the Graduate Council

Date _____ _____
Andrew G. Campbell
Dean of the Graduate School

Acknowledgements

This dissertation would not have been possible without the help and support of many others:

My advisor, Philip Klein, for supporting my research and for always encouraging consideration of *how* ideas can be most effectively presented and clearly communicated.

My committee members, Claire Mathieu, Eli Upfal, and Renato Werneck, for making time for me amid their many commitments and for their thoughtful questions and feedback.

My coauthors and collaborators—especially Alice Paul, David Saulpic, and Aaron Schild, for allowing me to include material and figures from our joint work in this dissertation.

Peer-reviewers, friends, and colleagues for the (often unpaid and under-acknowledged) time they have spent giving valuable feedback on papers, talks, and research ideas.

The Brown CS Department and in particular Lauren Clarke and other staff members who do all the behind-the-scenes work that keeps the department running smoothly.

My grandfather for instilling the value of education and research.

My parents for nurturing my love of problem-solving.

Jenna and Elizah for never letting me take myself too seriously.

Cara and Tenley for providing balance and for their patience during the homestretch.

The many friends, family members, and former teachers who have been cheering me on.

The work in this dissertation was supported by the National Science Foundation grant CCF-1409520.

Contents

List of Figures

Chapter 1

Introduction

Facility placement and vehicle routing describe a category of optimization problems that address the very common goal of planning how a given service can best meet the demand imposed by a set of clients at locations. This generality allows many disparate objectives to be modeled, including provider-centric objectives to minimize service expenses (e.g. vehicle mileage, fuel consumed, vehicles needed, buildings constructed, staff hired, etc.), client-centric objectives to minimize inconvenience (e.g. travel distance, waiting time, missing deadlines, etc.), and objectives that combine these.

In most real-world planning problems, the vehicles and facilities have limited sizes, resources, and staff, so it is often infeasible for all clients to be served by a single facility, vehicle, or trip. Much of the research on these problems has not taken these real-world limits into consideration—and for good reason: even without considering capacities, these problems are already NP-hard, and are thus unlikely to be solved efficiently on even moderately large instances.

Although these problems are hard to efficiently solve exactly, a near-optimal solution often suffices. One approach to this relaxation is to design *heuristics* that in practice may provide good solutions, quickly, but have no guarantee on performance and often perform poorly on large instances. Another approach is to design *approximation algorithms*, which have provable performance guarantees that don't depend on instance size. Heuristics may outperform approximation algorithms on small instances, but approximation algorithms are often the only way to find near-optimal solutions for large instances.

Many facility-placement and vehicle-routing problems cannot even be efficiently *approximated*

well[1]. However, many actual transportation networks have a fairly predictable structure. Making assumptions about the structure of the network can greatly simplify vehicle-routing problems and often allows for better approximation guarantees. That is, the complexity of vehicle problems in general does not adequately capture the complexity of these problems on the networks for which these problems are relevant!

In this thesis, we show that some facility-placement and vehicle-routing problems that model the real-world constraints of vehicles, drivers, and clients can be near-optimally approximated in the networks that model transportation networks. Specifically, we address problems that can model real-world constraints, including Capacitated k-Domination, Capacitated Vehicle Routing, and Minimum Makespan Vehicle Routing for graphs that model transportation networks, including: graphs of bounded treewidth, graphs of bounded genus, and graphs of bounded highway dimension. In this thesis, we design approximation algorithms and prove performance guarantees for these problems in these metrics.

1.1 Motivation

In this section, we motivate the work of this thesis.

1.1.1 Problem Relevance

Building new facilities (e.g. schools, hospitals, fire stations, company branches, etc.) costs taxpayers and investors many millions of dollars, so it is important to plan their locations carefully to avoid unnecessary costs. Once facilities have been established, vehicle-routing costs continue to constitute a large portion of business and public expenses. The savings from even slight improvements can be substantial. Recognizing this opportunity for improvement, organizations are turning toward algorithmic solutions.

As an illustrative example, in 2016 the Boston Public School district spent $110 million dollars (11% of the district's entire budget) on transportation costs [90]. In 2017, they hosted the Boston Public School's Transportation Challenge, seeking public proposals for optimizing bus routes, in an attempt to reduce busing costs and redirect the savings to other education needs.

[1] Under the assumption that $P \neq NP$

1.1.2 Modeling Transportation Networks

We claim that vehicle-routing problems are more appropriately studied in the metrics for which they are relevant. In particular, we use graphs of bounded treewidth, graphs of bounded genus, and graphs of bounded highway-dimension to model transportation networks.

Although trees and graphs of bounded treewidth (or branchwidth) have simple structure that do not adequately model many road networks (for example, the grid street plans of cities like New York or Washington DC tend to have rather high treewidth), they remain a relevant class of graphs for studying vehicle-routing problems. Trees model many mining and logging networks, where the cost of adding redundancy is prohibitively expensive or destructive [74]. Similarly, shipping transportation along a shoreline with inland rivers is also modeled by a tree [69]. Furthermore, many buildings such as hotels, hospitals, warehouses, offices, and supermarkets are designed to have long branching corridors lined with rooms or goods [69]. Such layouts have very low treewidth. Commuter-rail networks for many cities (e.g. Boston or Washington DC) also have very low treewidth as they are primarily designed to transport people into and out of the city, rather than to connect distant suburbs to each other. Vehicle-routing problems arise naturally in all of these settings (e.g. restocking grocery shelves, coordinating room service for a large hotel, transporting ore out of mines, etc.).

Additionally, from a theoretical standpoint, understanding the complexity of the problem on trees bounds the complexity of the problem for other metrics. Designing algorithms for trees and bounded-treewidth graphs can also offer insight into designing algorithms for more complex graphs, and as we will see, some algorithms work by reducing the problem to bounded-treewidth instances.

Many vehicles (e.g. cars, trains, people, etc.) are physically restricted to move along the surface of the earth, which makes planar graphs natural candidates for modeling their transportation networks. This model is complicated by the use of bridges and tunnels which introduce *non-planarities*, in which edges may *cross* without actually intersecting each other. If there are few such non-planarities, the network can be modeled as a bounded-genus graph. Often these non-planarities can be localized in such a way that algorithms can 'work around' them [20].

Finally, highway dimension is a parameter that was specifically designed to capture a common trait of many transportation networks: shortest paths connecting points that are far from each other tend to pass through at least one of a small number of *hubs*. This has been experimentally shown to hold for US road networks [18, 19]. Therefore graphs of bounded highway dimension are relevant

to the study of vehicle-routing problems.

1.1.3 Performance Guarantees

We design approximation algorithms with provable performance guarantees. Specifically, our algorithms find *near-optimal* solutions, which we define to mean approximation schemes and small constant-factor approximations. Facility-location and vehicle-routing problems often arise in public works projects and in business planning. An important aspect of this type of planning is creating a realistic budget. This requires having at least a ballpark estimate of the associated costs. Approximation algorithms can provide these estimates.

In practice, exact algorithms can only be used to solve relatively small instances. Heuristics may be used for large instances, but their performance can be quite poor. Approximation algorithms often outperform heuristics for large instances. Provable approximations also provide lower bounds to solution costs which can be used to better understand the performance of other algorithms.

1.2 Thesis Overview

We give a brief overview of the techniques and results presented in this thesis.

1.2.1 Outline of Results

Capacitated ρ-Domination is the problem of opening the smallest number of facilities such that each client is assigned to a facility within a distance ρ and the number of clients assigned to each facility does not exceed its capacity. In Chapter 2 we present a polynomial-time approximation scheme (PTAS) for Capacitated ρ-Domination in unit-weight planar graphs when the distance ρ and the maximum capacity $\hat{q}$ are fixed. Our algorithm extends Baker's framework [13] for approximation schemes in planar graphs to the capacitated setting. Specifically, directly applying Baker's framework results in some facilities being *overcapacitated*. We allow this overcapacitation temporarily, and then perform a post-processing *smoothing* step that reassigns clients to achieve a feasible solution.

Capacitated Vehicle Routing is the problem of routing a set of vehicle tours from a depot so that they collectively cover all clients, the client demand covered by each vehicle does not exceed its capacity, and the sum of the tour lengths is minimized. In Chapter 3 we present some general

techniques for designing Capacitated Vehicle Routing approximation schemes when the vehicle capacity Q is bounded. Specifically we introduce an *error allowance* that can be afforded for every pair of consecutive clients u and v in a solution. This error is proportional not only to the distance between u and v, but also proportional to their distances to the depot. Furthermore, Capacitated Vehicle Routing with fixed capacities can be solved in polynomial time in graphs of bounded treewidth (or branchwidth), using a dynamic-programming algorithm. Putting these pieces together, we present a framework that reduces the goal of designing approximation schemes for Capacitated Vehicle Routing to that of designing *metric embeddings* into graphs with bounded treewidth such that distances are preserved up to the client-to-client error allowance. We show how such an embedding reduction can also be used for variants including Multi-Depot Capacitated Vehicle Routing and Capacitated Vehicle Routing with Penalties.

In Chapter 4 we present a quasi-polynomial-time approximation scheme (QPTAS) for Capacitated Vehicle Routing in planar and bounded-genus graphs when the vehicle capacity Q is fixed. Our algorithm recursively decomposes the input graph using separators composed of shortest paths. It then uses the client-to-client error allowance to designate portals along these separators such that a solution can afford to take *detours* in order to cross the separators at the portals. By polylogarithmically bounding the number of portals and depth of the recursion, the QPTAS follows from a dynamic-programming algorithm over the decomposition.

In Chapter 5 we improve this result to a polynomial-time approximation scheme (PTAS). We present a randomized metric embedding for planar graphs into bounded-treewidth graphs that preserves distances *in expectation* up the client-to-client error allowance. We then show how to derandomize the embedding so that the Capacitated Vehicle Routing solution cost is guaranteed to be near-optimal.

In Chapter 6 we give a PTAS for Capacitated Vehicle Routing with fixed capacities in graphs of bounded highway dimension. This result follows from designing a (deterministic) metric embedding of graphs of bounded highway dimension into graphs of bounded treewidth such that the distances are preserved up to the client-to-client error allowance. We extend this embedding to the multiple-depot setting, which gives an analogous PTAS for Multi-Depot Capacitated Vehicle Routing as well as fixed-parameter approximations for k-Center and k-Median in graphs of bounded highway dimension.

Capacitated Vehicle Routing with general (arbitrary) capacities is a considerably more

difficult problem. In Chapter 7 we present a 4/3 approximation algorithm for tree metrics, which is the best known approximation ratio and is in fact tight with respect to the best-known lower bound. The algorithm proceeds by iteratively finding sets of tours whose cost is within a 4/3 factor of their contribution to the lower bound. We show that if clients remain, such a set of tours can always be found. The union of these tour sets gives a 4/3-approximate solution.

In Chapter 8 we present a framework for designing polynomial-time approximation schemes for many vehicle-routing problems in trees, including Minimum Makespan Vehicle Routing and Min-Max Regret Routing, as well as *bicriteria* PTASs for problems including Distance-Constrained Vehicle Routing, Capacitated Vehicle Routing, and School Bus Routing. The framework works by partitioning the tree into clusters such that there exists a near-optimal solution that has simple structure with respect to the clusters. In particular, there exists a solution in which only a small number of vehicles serve clients in any given cluster. This clustering greatly restricts the set of potential solutions to consider without adding too much to the optimal solution cost. We present a dynamic-programming algorithm that then finds a near-optimal solution in polynomial time.

1.2.2 Techniques

One of the general techniques used throughout this thesis is finding ways to simplify the input metric without introducing too much error. If the metric is sufficiently simplified, the problem can then be solved optimally (or near-optimally) in the simpler instance. If the simplification does not introduce too much error, an optimal solution in the simplified metric can then be mapped back to a near-optimal solution in the input graph. This introduces a trade off between error and runtime: the simpler the metric, the faster the runtime of solving the problem, but the greater the error of the resulting solution.

We use this technique in the design of our approximation schemes, and our algorithms demonstrate this trade off. For Capacitated ε-Domination, our PTAS extends Baker's framework, which splits the input graph into subgraphs, solves the induced problem on each subgraph, and then combines the local solutions to form a global solution (see Chapter 2). Here, the branchwidth of the subgraphs is inversely proportional to the error introduced, and the runtime is exponential in the branchwidth. That is, to achieve smaller error, the subgraph branchwidth and thus runtime must be larger. For Capacitated Vehicle Routing, our QPTAS in planar graphs chooses portals

along separators through which to route the solution (see Chapter 4). The runtime is exponentially dependent on the number of portals per separator, and smaller error requires more portals. Our PTASs for Capacitated Vehicle Routing in planar graphs and bounded highway dimension graphs use metric embeddings into graphs of bounded treewidth that preserve distances up to the client-to-client error allowance (see Chapters 5 and 6). The treewidth of these embeddings depends inversely on the error: achieving smaller error requires a larger treewidth. Finally, in the framework we present for vehicle-routing problems in trees, the tree is partitioned into clusters (see Chapter 8). To achieve smaller error, the clusters must be smaller, which increases the complexity of the cluster structure and the runtime of the dynamic program.

Another general technique we use is bounding the problem parameters. For example, several of our approximation schemes for Capacitated Vehicle Routing require fixed or bounded capacity Q, our Multi-Depot Capacitated Vehicle Routing results requires a bounded number of depots ρ, and our Capacitated $\mathscr{k}$-Domination PTAS requires bounded $\mathscr{k}$. We also give results for instances with bounded graph parameters such as treewidth ω, genus g, and highway dimension η. We then give algorithms whose performance is measured in terms of these parameters. This in some sense isolates what is difficult about these problems for arbitrary constraints and general metrics. For example, in our approximation scheme for Capacitated Vehicle Routing in graphs of bounded highway dimension, given a fixed $\epsilon > 0$, $Q > 0$, and $\eta > 0$, there is a polynomial-time algorithm that finds a $(1+\epsilon)$-approximate solution for Capacitated Vehicle Routing in instances with capacity Q and highway dimension η. Though this algorithm can be used on arbitrary instances, as η becomes arbitrarily large, so does the degree of the polynomial (as is to be expected since Capacitated Vehicle Routing is APX-hard in general metrics, even for $Q = 3$ [11]).

1.3 Definitions and Notation

In this section we introduce the general definitions and notation that are used throughout this thesis. We defer defining problem-specific notation and terms until the relevant chapters.

1.3.1 Basic Graph Notation

We use $G = (V, E)$ to denote an undirected graph G with vertex set V and edge set E. We use ℓ_G to denote the edge lengths of G, and unless otherwise noted we assume that $\ell_G(e) \geq 0$. When G is

unambiguous, we often remove it from the subscript. For a subgraph or edge set F, we define $\ell(F)$ to be the sum $\sum_{e \in F} \ell(e)$.

We use $[k]$ to denote the set $\{1, 2, ..., k\}$.

A *path* P is a sequence of vertices $P = v_1, v_2, ..., v_{|P|}$ such that for every $i \in [|P| - 1]$, (v_i, v_{i+1}) is an edge in E. The length $\ell(P)$ of path P is the sum of the lengths of these edges. Vertices v_1 and $v_{|P|}$ are called the *endpoints* of P, and P is a v_1-to-$v_{|P|}$ path.

The edge lengths define a metric on the vertex set V. That is, for every $u, v \in V$, $d_G(u, v) = \ell_G(P[u, v])$ where $P[u, v]$ denotes the shortest path from u to v. The shortest-path metric satisfies the triangle inequality, and we assume that if $e = (u, v)$ is an edge in E, then $\ell_G(e) = d_G(u, v)$. Again, when G is unambiguous, we omit it from the subscript. For a set $X \subseteq V$, we say that the distance $d(v, X)$ between vertex v and X is the distance $\min_{x \in X} d(v, x)$ to the closest member of X. We define the distance $d(X, Y)$ between two sets similarly.

The *diameter* of a graph G, denoted $diam(G)$, is the maximum distance $max_{u,v} d_G(u, v)$ over all vertices u and v. The *aspect ratio* of a graph G is the ratio of the maximum to minimum distances in the graph, namely $\frac{diam(G)}{min_{u \neq v} d_G(u,v)}$.

All-pairs shortest paths can be easily computed in polynomial time, so we assume throughout this thesis that we have access to all (precomputed) distances.

A *tour* π is a *closed* path. That is, for some $v \in V$, π is a v-to-v path. We say that tour π *starts* and *ends* at v. A *cycle* is a non-empty tour in which no edge is traversed more than once.

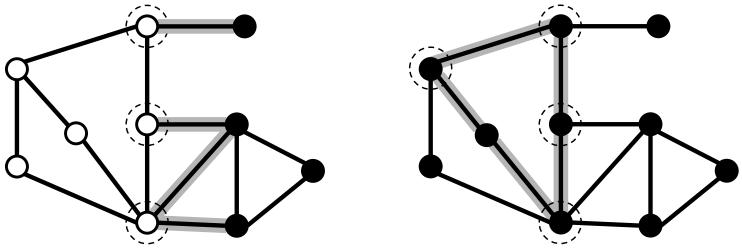
For an undirected graph $G = (V, E)$ with subset of edges, $F \subseteq E$, we define the *boundary* of F, denoted $\partial_G(F)$, to be the set of vertices in V incident both to edges in F and $E \setminus F$. For a subset of vertices, $S \subseteq V$, we define the *cut* of S, denoted $\delta_G(S)$, to be the set of edges in E incident both to vertices in S and $V \setminus S$, and we define the *frontier* $\partial_G(S)$ to be the set of vertices in S incident to edges in $\delta_G(S)$ (refer to Figure 1.1). Again, when G in unambiguous, we omit the subscript.

1.3.2 Graph Classes

We now formally define the specific graph classes that we address in this thesis.

Trees

A tree is a graph T that contains no cycles. Often the tree is *rooted* at some root vertex r. If v is a non-root vertex in a rooted tree T, the *parent* of v, denoted $p(v)$, is the vertex adjacent to v in $P[v, r]$



(a) cut and frontier (b) boundary

Figure 1.1: (a) Vertices in $S \subseteq V$ are shown in white; edges of cut $\delta_G(S)$ are highlighted gray; vertices of frontier $\partial_G(S)$ are circled. (b) Edges in $F \subseteq E$ are highlighted gray; vertices of boundary $\partial_G(F)$ are circled.

(the shortest v-to-r path). The parent of r is not defined. Similarly, u is a *child* of v if $v = p(u)$. Vertices with no children are called *leaves*. The *ancestors* of v are all vertices in $P[v, r]$, including v and r, and the *descendants* of v are all vertices u such that v is an ancestor of u, including v itself. The *subtree rooted at* v, denoted T_v is the subgraph of T induced by the descendants of v.

A *binary* tree is a rooted tree in which every vertex has exactly two or zero children.

Embeddings and Drawings

A *metric embedding* of a *guest* graph G into a *host* graph H is a mapping $\phi(\cdot)$ of the vertices of G to the vertices of H.

A *drawing* (which is also called an embedding) of a graph $G = (V, E)$ on a surface M is a mapping $\phi(\cdot)$ of each v in V to a distinct point $\phi(v)$ on M and each edge $(u, v) \in E$ to a continuous arc on M between $\phi(u)$ and $\phi(v)$ that does not contain $\phi(w)$ for any $w \in V \setminus \{u, v\}$. An *edge crossing* in a drawing is a point $p \notin \phi(V)$ on M such that two distinct arcs contain p.

To avoid confusion, in this thesis, we use the word *embedding* only to describe metric embeddings, and use *drawing* to describe the mapping of a graph onto a surface.

Planar and Bounded Genus Graphs

The *genus* of a graph G is the smallest integer k such that G can be drawn on a genus-k surface without edge crossings. A genus-zero graph G is called *planar* since it can be drawn on the plane (or equivalently on the surface of a sphere) with no edge crossings.

A planar graph with n vertices has $O(n)$ edges.

Branchwidth and Treewidth

There are various notions of the *width* of a graph.

A *tree decomposition* of a graph $G = (V, E)$ is a tree T in which each node of T (called a *bag*) consists of a set of vertices in G, such that the following hold:

- Every vertex $v \in V$ is in some bag in T,
- for every edge $e = (u, v) \in E$ there is some bag b in T such that u and v are both in b, and
- for every $v \in V$, the subgraph of T induced by the bags that contain v is connected.

The *width* of a tree decomposition is the size of the largest bag minus one. The *treewidth* of a graph G is the minimum width among all tree decompositions of G. The treewidth is a measure of how *treelike* the graph is. It is not hard to see that the treewidth of a tree is one, and the treewidth of a complete graph is $|V| - 1$.

It is often useful to transform a tree decomposition into a *nice* tree decomposition. In a nice tree decomposition, each leaf bag contains a single vertex and each non-leaf bag of the decomposition is one of three types:

- An *introduce* bag b has one child b', such that $b = b' \cup \{v\}$ for some vertex $v \notin b'$. The vertex v is introduced at b.
- A *forget* bag b has one child b', such that $b = b' \setminus \{v\}$ for some vertex $v \in b'$. The vertex v is forgotten at b.
- A *join* bag b has two children b_1 and b_2 such that $b = b_1 = b_2$.

A tree decomposition can be converted into a nice tree decomposition with the same width in polynomial time [34].

A collection of sets $\mathcal{X}$ is called a *laminar family* if for all $X_1, X_2 \in \mathcal{X}$, either $X_1 \subseteq X_2$, $X_2 \subseteq X_1$ or $X_1 \cap X_2 = \emptyset$.

A *recursive partition* [40] (also called a *recursive decomposition*) of a set X is a rooted binary tree T in which each node corresponds to a *cluster* $\mathcal{C} \subseteq X$, such that:

- The root node corresponds to cluster $\mathcal{C} = X$, and
- for every node corresponding to a cluster $\mathcal{C}_0$, the clusters $\mathcal{C}_1$ and $\mathcal{C}_2$ corresponding to its child nodes form a partition of $\mathcal{C}_0$.

The clusters in a recursive partition form a laminar family.

A *branch decomposition* of a graph $G = (V, E)$ is a recursive partition of E in which the leaf clusters are singleton edge sets. The *boundary* of a cluster $\mathcal{C}$ in a branch decomposition is the set of vertices incident to edges both in $\mathcal{C}$ and not in $\mathcal{C}$. The *width* of a branch decomposition is the size of the largest cluster boundary. The branchwidth of a graph G is the minimum width over all branch-decompositions of G.

Branchwidth and treewidth are closely related and have been proved, for any graph, to be within a small constant multiplicative factor of each other [88].

The tree structure of branch and tree decompositions make them very well suited for dynamic-programming algorithms. In particular, it often suffices to consider the way a solution interacts with vertices in each bag (for tree decompositions) or cluster boundary (for branch decompositions). If the width is small, the possible interactions can be enumerated efficiently.

Highway Dimension

Given an $x \in \mathbb{R}^+$ and a vertex $v \in V$ we use $B_x(v) = \{u \in V | d(u, v) \leq x\}$ to denote the *ball* with radius x centered at v. We use the definition of highway dimension from Feldmann et al. [47]: The *highway dimension* of a graph G is the smallest integer η such that for every $x \in \mathbb{R}^+$ and $v \in V$, there is a set of at most η vertices in $B_{cx}(v)$ such that every shortest path in $B_{cx}(v)$ of length at least x intersects this set. Here, c is a small fixed constant greater than four. Note that the highway dimension of a graph depends on this choice of c (see Section 6.1.1).

The *doubling dimension* of a graph G is the smallest integer θ such that for every vertex v and radius x, a ball $B_{2x}(v)$ of radius $2x$ centered at v can be covered by at most 2^θ balls of radius x. That is, there is a set of vertices $\{v_1, v_2, ..., v_{2^\theta}\}$ such that $B_{2x}(v) \subseteq \bigcup_{1 \leq i \leq 2^\theta} B_x(v_i)$.

1.3.3 Approximation Algorithms

We use OPT to denote an optimum solution for an optimization problem and $cost(OPT)$ to denote its cost. For a minimization problem, an *α-approximation algorithm* is an algorithm that finds a solution with cost at most $\alpha \cdot cost(OPT)$ (also called an α-approximate solution). We say that an algorithm is polynomial-time if the running time is $O(n^c)$ for some constant c, where n is the size of the input.

A *polynomial-time approximation scheme* (PTAS) is a family of algorithms, indexed by ϵ, such that for every $\epsilon > 0$, a $(1+\epsilon)$-approximate solution is found in $O(n^c)$ time, where c can depend on ϵ (but not on n). An *efficient polynomial-time approximation scheme* (EPTAS) is a PTAS that runs in time $O(f(\epsilon)n^c)$ where c does not depend on ϵ. A *fully-polynomial-time approximation scheme* (FPTAS) is a PTAS that runs in time polynomial both in n and $\frac{1}{\epsilon}$. A *quasi polynomial-time approximation scheme* (QPTAS) is an approximation scheme that runs in time $O(n^{\log^c(n)})$ where the degree of n may depend polylogarithmically on n.

An $(\alpha, \beta)-$*bicriteria* approximation algorithm for a minimization problem with some constraint value k and optimum solution OPT is an algorithm that finds a solution of cost at most $\alpha \cdot cost(OPT)$ that is feasible for constraint value βk. For example, in Capacitated Vehicle Routing, such an algorithm would find a solution of cost at most $\alpha \cdot cost(OPT)$ in which each tour covers at most βQ clients.

A *heuristic* is a technique for finding a feasible solution in a shorter amount of time than what is required to solve the problem exactly. Though such solutions can often be found in a reasonable amount of time, the quality of these solutions is not guaranteed to be near-optimal. While heuristics are usually problem-specific, *metaheuristics* are problem-independent approaches to finding feasible solutions. Metaheuristics include local-search strategies, ant colony optimization, and evolutionary algorithms.

We will not define complexity classes in detail in this thesis, but we remind readers that under the assumption that $P \neq NP$, NP-hard problems have no polynomial-time exact algorithms and APX-hard problems have no polynomial-time approximation schemes. A problem that is parameterized by k is *fixed-parameter tractable* (FPT) if there exists an exact algorithm that runs in time $f(k)n^{O(1)}$, where n is the size of the input and f is some computable function. Under the exponential-time hypothesis, a problem that is $W[1]$-hard when parameterized by k is not in FPT, and no such algorithm exists. An approximation algorithm for a problem parameterized by k is a *fixed-parameter approximation* (FPA) if its runtime is $f(k)n^{O(1)}$ for some computable function f that does not depend on n.

Chapter 2

A PTAS for Capacitated ℓ-Domination in Planar Graphs

2.1 Introduction

In this chapter, we address the Capacitated ℓ-Domination problem, which is a generalization of the classic Dominating Set problem that accommodates integral demands, capacities, and distances. We present a PTAS for instances on planar graphs in which the domination distance and the capacities are bounded.

Dominating Set is a classic NP-complete optimization problem that takes as input an undirected graph G with vertex set V and finds a *dominating set* $S_{dom} \subseteq V$ such that every vertex $v \in V$ is either in S_{dom} or adjacent to a vertex in S_{dom} and such that $|S_{dom}|$ is minimized. Beyond facility location, Dominating Set has applications in resource allocation [51] and social network theory [95].

The ℓ-Domination problem generalizes Dominating Set. For a positive integer ℓ, the ℓ-Domination problem is to find a dominating subset $S_{dom} \subseteq V$ such that every vertex $v \in V$ is within a hop-distance ℓ of some vertex in S_{dom} and $|S_{dom}|$ is minimized. The ℓ-Domination problem is an optimization version of the (k, ℓ)-Center decision problem of determining for a given k and ℓ whether there is a set of k vertices (*centers*) such that every vertex is at most a distance ℓ from the nearest center. It is straightforward to see that Dominating Set is a special case of

ℓ-Domination in which $\ell = 1$.

We consider *capacitated* versions of Dominating Set problems, in which the *capacity* of vertices (i.e. the number of neighbors that a vertex in the dominating set can *cover*) is restricted. The capacitated Dominating Set problem, Capacitated Domination, takes as input *demand* and *capacity* values for every vertex, and finds a dominating set S_{dom} of minimum size such that every vertex in V has its demand *covered* by vertices in S_{dom}, the total amount of demand covered by any vertex in S_{dom} does not exceed its capacity, and any vertex can only cover itself or its neighbors (i.e. the *closed neighborhood*)[1]. We similarly define Capacitated ℓ-Domination with the difference being that any vertex can only cover vertices within a hop-distance ℓ.

In this chapter we present a PTAS for Capacitated ℓ-Domination in planar graphs, when both ℓ and the maximum capacity are bounded. We formalize this in the following theorem.

Theorem 1. *For any $\epsilon > 0$ and positive integers ℓ and $\hat{q}$, there exists a polynomial-time algorithm that, given an instance of* Capacitated ℓ-Domination *on a planar graph with maximum capacity $\hat{q}$, finds a solution of cost at most $1 + \epsilon$ times the optimum cost.*

Note that by setting $\ell = 1$, an analogous PTAS for Capacitated Domination follows as a corollary.

We use Baker's framework for planar graphs (see Section 2.3) to achieve this result. Baker's framework depends on splicing together local solutions to form a global solution. The challenge in extending this framework to capacitated problems is that splicing feasible (*capacity-respecting*) local solutions may generate an infeasible (*capacity-exceeding*) global solution. To address this, we show how to *smooth* a capacity-exceeding global solution into a capacity-respecting global solution. Additionally, we modify the traditional Baker framework to bound the cost of this smoothing process. Finally, Baker's framework also depends on being able to efficiently solve instances on graphs of bounded branchwidth. We introduce the first known dynamic programming algorithm to solve Capacitated ℓ-Domination in planar graphs of bounded branchwidth.

2.1.1 Related Work

For Dominating Set in general graphs, an $O(\ln n)$-approximation is known [64]. It has also been shown that unless NP $\subset$ TIME($n^{\log \log n}$), Dominating Set cannot be approximated within a factor

[1] In one common variant of the problem, a vertex can cover its own demand 'for free' and the capacity only limits coverage of neighbors.

of $(1-\epsilon)\ln n$ [45], thus a PTAS is unlikely to exist. For restricted graph classes such as unit disk graphs [85], bounded-degree graphs [29], and planar graphs [81], improvements on this bound have been found. In particular, the planar case is known to admit a PTAS using the framework established by Baker [13] and described below.

For $\mathscr{r}$-DOMINATION in general graphs, a $\ln n$-approximation is known [16]. Planar graphs and map graphs are known to admit a PTAS, again using Baker's approach [37].

CAPACITATED DOMINATION was shown to be W[1]-hard for general graphs when parameterized by both treewidth and solution size [38]. Moreover, this hardness result extends to planar graphs, and the bidimensionality framework does not apply to this problem [22].

Kao et al. characterize CAPACITATED DOMINATION by several attributes [66]. The *weighted* version has real, non-negative vertex weights whereas the *unweighted* version has uniform vertex weights. The *unsplittable*-demand version requires that the entire demand of any vertex is covered by a single vertex in the dominating set, while the *splittable*-demand version allows demand to be covered by multiple vertices in the dominating set. The *soft*-capacity version allows multiple copies of vertices to be included in the dominating set (effectively scaling both the capacity and the weight of vertex by any integral amount) whereas the *hard*-capacity version limits the number of copies of each vertex (usually to one). Lastly, the capacities and demands can be required to be (non-negative) integral.

For the soft-capacity version, several results are known. For general graphs, Kao et al. present a $(\ln n)$-approximation for the weighted unsplittable-demand case, a $(4\ln n+2)$-approximation for the weighted splittable-demand case, a $(2\ln n+1)$-approximation for the unweighted splittable-demand case, and a (Δ)-approximation for the weighted splittable-demand case, where Δ is the maximum degree in the graph [66, 67]. As these are generalizations of the classic DOMINATING SET problem, there is little room for improvement for these approximations on general graphs.

CAPACITATED DOMINATION with soft capacities in trees is NP-hard but admits a PTAS [67]. For soft capacities in planar graphs, Kao and Lee developed a constant-factor approximation algorithm [65]. Additionally, Kao and Chen present a pseudo-polynomial-time approximation scheme for the weighted splittable-demand case by first showing how to solve the problem on graphs of bounded treewidth and then applying Baker's framework [67].

Substantially fewer results have been published on CAPACITATED DOMINATION with *hard* capacities. In this work we consider the **unweighted**, **splittable-demand**, **hard-capacity** version

of CAPACITATED $\imath$-DOMINATION with integral capacities and demands. When $\imath = 1$, our PTAS (stated in Theorem 1) is the hard-capacity complement of the PTAS presented by Kao and Chen for soft-capacities [67].

Very few results have been published for CAPACITATED $\imath$-DOMINATION. For points on a unit sphere, a heuristic using network flow was developed [79]. In the closely related CAPACITATED k-CENTER problem, the number of centers, k, is given and the objective is to minimize the maximum distance of a vertex to a center, subject to each center having a limited number of clients (vertices) assigned to it. For uniform capacities and unit demand, a 6-approximation is known [73] for edge-weighted graphs. Note that k-CENTER and $\imath$-DOMINATION are two different optimization perspectives of $(k, \imath)$-CENTER.

2.2 Preliminaries

Recall, that for an undirected graph $G = (V, E)$ and subset of edges $F \subseteq E$, the *boundary* of F, denoted $\partial_G(F)$, is the set of vertices in V incident both to edges in F and $E \setminus F$. For a subset of vertices, $S \subseteq V$, the *cut* of S, denoted $\delta_G(S)$, is the set of edges in E incident both to vertices in S and $V \setminus S$, and the *frontier* $\partial_G(S)$ is the set of vertices in S incident to edges in $\delta_G(S)$.

Our problems define demand functions $dem : V \to \mathbb{Z}^*$ and capacity functions $q : V \to \mathbb{Z}^*$ on the vertex set V. The capacity (resp. demand) of a set of vertices is the sum of the constituent vertex capacities (resp. demands). Namely, for $S \subseteq V$, $q(S) = \sum_{v \in S} q(v)$ and $dem(S) = \sum_{v \in S} dem(v)$.

Let $d_{hop}(u, v)$ denote the hop-distance (number of edges in shortest path) from u to v. If $d_{hop}(u, v) \leq \imath$ we say that u and v are *$\imath$-neighbors* and $N_\imath(v) = \{u \mid d_{hop}(u, v) \leq \imath\}$ is the (closed) *$\imath$-neighborhood* of v.

For a given demand function $dem(\cdot)$, capacity function $q(\cdot)$, and hop-distance $\imath$, an *assignment* $\mathcal{A}$ is a multiset of *facility-client* pairs (u, v) with $u, v \in V$ such that $d_{hop}(u, v) \leq \imath$ and $\forall v \in V,\ |\{(x, v) \in \mathcal{A} \ : \ x \in V\}| \leq dem(v)$ (i.e. v appears as a client at most $dem(v)$ times). It is convenient for our algorithm to define assignments in such a way that client demand is never exceeded. Note that in this chapter we denote edge (u, v) as e_{uv} to distinguish edge notation from facility-client pair notation.

An assignment is *proper* or *capacity-respecting* if $\forall u \in V,\ |\{(u, x) \in \mathcal{A} \ : \ x \in V\}| \leq q(u)$ (i.e. u appears as a facility at most $q(u)$ times). An assignment is *covering* or *demand-meeting* if

$\forall v \in V,\ |\{(x,v) \in \mathcal{A} \ :\ x \in V\}| = dem(v)$. The total unmet demand for an assignment is $t(\mathcal{A}) = \sum_{v \in V} (dem(v) - |\{(x,v) \in \mathcal{A} \ :\ x \in V\}|)$. Observe that if $\mathcal{A}$ is covering then $t(\mathcal{A}) = 0$. A given assignment $\mathcal{A}$ has *dominating set* (or *facility set*) $S_{dom}(\mathcal{A}) = \{u \in V : (u,v) \in \mathcal{A}\}$ and *cost* $cost(\mathcal{A}) = |S_{dom}(\mathcal{A})|$. The objective of Capacitated $\varkappa$-Domination is to find a proper, covering assignment $\mathcal{A}$ such that $cost(\mathcal{A})$ is minimized.

Conceptually, in a solution to Capacitated $\varkappa$-Domination, $\mathcal{A}$ describes an assignment of vertices to $\varkappa$-neighbors such that the demand of every vertex is met and the capacity of every vertex is not exceeded. This generalizes Capacitated Domination in which $\varkappa = 1$ which itself generalizes the classic Dominating Set problem in which every vertex v has $dem(v) = 1$ and $q(v) = \infty$.

We use superscripts to denote the multiplicity of a facility-client pair in an assignment. For example $\{(u,v)^4, (w,w)\}$ assigns four units of demand from the vertex v to the vertex u and one unit of demand from w to itself.

We use $\mathcal{A}_1 \uplus \mathcal{A}_2$ to denote the multiset union operation in which multiplicities are additive. For example $\{(u,v)^2, (u,w)\} \uplus \{(u,v)^3, (w,v)\} = \{(u,v)^5, (u,w), (w,v)\}$.

2.3 Baker's Framework

Our PTAS (presented in Section 2.4.1) is based on Baker's framework for designing PTASs in planar graphs [13]. For a given problem, the framework decomposes the input graph into subgraphs of bounded branchwidth. In each of these subgraphs, the induced subproblem can be solved efficiently. A global solution is then formed by combining the local subgraph solutions. The error incurred by splicing local solutions together can be charged to the interface between these subgraphs, and a *shifting* argument is used to bound this error.

We now present Baker's framework in more detail by demonstrating how it has been applied to Dominating Set (and soft-Capacitated Domination) on planar graphs [65, 81]. The input graph is partitioned into *levels* (vertex subsets) defined by minimum hop-distance (breadth-first search) from an arbitrary root v_0. The resulting subgraph induced by k consecutive levels has $O(k)$ branchwidth. The graph is decomposed into subgraphs of k consecutive levels such that every two adjacent subgraphs overlap in two *boundary* levels (see Figure 2.1a). There are at most k different options (called *shifts*) for dividing the graph into k-level subgraphs in this way. For each shift, Dominating Set is solved for each (bounded-branchwidth) subgraph with the slight modification

that the demand from the vertices of the topmost and bottommost levels of each subgraph is ignored. Note that these vertices can still be included in the local subgraph solutions. These local solutions can be found in $2^{O(k)}n$ time and a global solution is found by taking the union of the local solutions.

Ignoring the demand of the outermost levels prevents over-counting the demand requirements of the vertices that appear in multiple subgraphs (i.e. the levels in which adjacent subgraphs overlap), while still allowing these vertices to cover demand from inner levels in the subgraph. By construction, adjacent subgraphs overlap by exactly two levels, so a vertex appears on the outermost level of at most one subgraph. In particular, any demand ignored by one subgraph will be covered by an adjacent subgraph. Taking the union of subgraph solutions ensures then that all demand is covered.

We can bound the error incurred by this algorithm by observing the following. Because demand of the outermost subgraph levels is ignored, an optimum global solution, OPT, induces local solutions OPT_i on each subgraph G_i. However the algorithm finds an *optimum* solution SOL_i for each such subgraph, so the cost of SOL_i is at most the cost of OPT_i. Consider the global solution SOL found by the algorithm by taking the union of the SOL_i. Observe that,

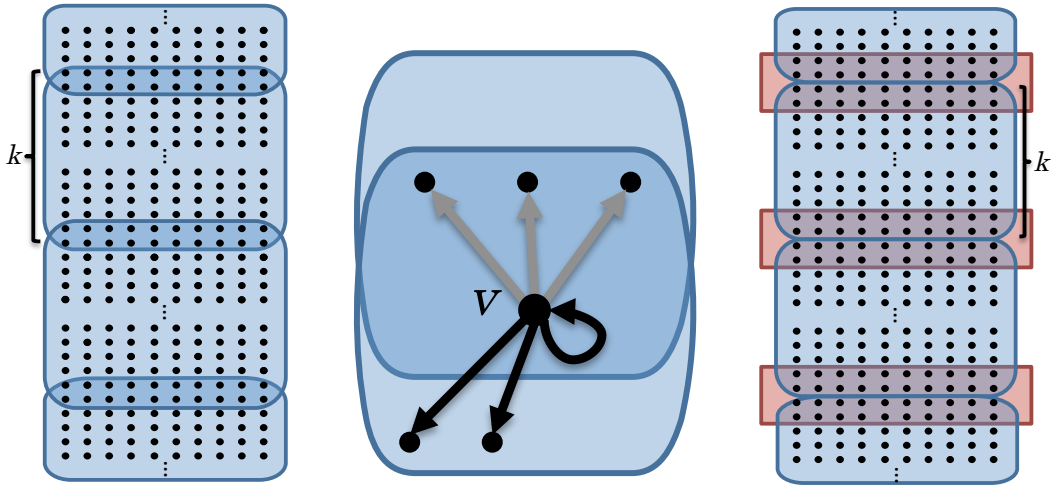
$$cost(SOL) \leq \sum_i cost(SOL_i) \leq \sum_i cost(OPT_i)$$

The sum of the costs of these local subgraph solutions, however, is bounded by the cost of OPT plus the cost of the subset of OPT that intersects the boundary levels, since these members of the solution are counted twice. By a simple averaging argument, for some *shift* this extra cost must be a small fraction ($O(1/k)$) of the total cost of OPT.

The problem with extending this approach to hard capacities emerges when a vertex in a boundary level is included in the solutions for two different subgraphs. For the classic Dominating Set problem, these vertices have unbounded capacities so taking the union of subgraph solutions is still feasible. Similarly, for soft-Capacitated (ℓ)-Domination, vertex capacities are allowed to be doubled. Since the cost of these doubled vertices is small, the algorithm can afford to pay double for them. However, for hard-Capacitated (ℓ)-Domination these vertices pose a problem. Taking the union of feasible subgraph solutions may *overload* these boundary vertices (i.e. exceed their capacities), and the hard-capacity setting does not allow simply paying extra to use these vertices multiple times (see Figure 2.1b).

We address this issue by redirecting the assignment of overloaded vertices to *underloaded* vertices

elsewhere in the graph. The algorithm may have to do this for *every* solution vertex that appears on the boundary levels. Unfortunately, the above bound on the sum of costs of subgraph solutions SOL_i does not bound the cost of the solution vertices of SOL that appear on boundary levels. We address this problem by modifying the way the graph is decomposed into subgraphs: instead of setting a single value k such that *all* subgraphs are uniformly defined by k consecutive levels, as is typical in Baker's framework, we set two values k_1, k_2 and alternate between two *types* of subgraphs in the decomposition: subgraphs defined by k_1 levels and subgraphs defined by k_2 levels. By charging all overloaded vertices to only one type of subgraph, we can bound the cost of the reassignment of these vertices by an appropriate choice of k_1 and k_2.



(a) Baker Subgraphs (b) Overloaded Vertex (c) Slabs and Patches

Figure 2.1: (a) Subgraphs consist of k consecutive BFS levels. (b) An arc from u to v indicates that (u, v) is a (facility, client) pair in the assignment. If $q(v) = 3$ the upper-subgraph assignment (gray arcs) and lower-subgraph assignment (black arcs) are each proper, but the union of solutions is not proper since vertex v is overloaded. (c) Patches are depicted by rectangles and slabs are depicted by rounded shapes.

2.4 PTAS

In this section we present our PTAS result for Capacitated ℓ-Domination in planar graphs, given by Theorem 1.

2.4.1 Description of PTAS

Our algorithm extends the approach introduced by Baker [13] and described in Section 2.3. A breadth-first search from an arbitrary root vertex, v_0, partitions the graph into m levels (defined by hop-distance from v_0) such that the subgraph induced by any k consecutive levels has branchwidth at most $2k$.

We set up the following notation for decomposing the graph into subgraphs. For all i, j, k such that $0 \leq i < k$, and $0 \leq j \leq \lfloor \frac{m}{k} \rfloor$, let G_j^i denote the subgraph induced by levels $jk + i$ through $(j+1)k+i-1$ and H_j^i denote the subgraph induced by levels $(j+1)k+i-2r$ through $(j+1)k+i+2r-1$. For a fixed k and i we call G_j^i the j_{th} *slab* and H_j^i the j_{th} *patch* (see Figure 2.1c). We call k the slab *height* and i the slab *shift*.

Consecutive slabs do not overlap each other and each slab has branchwidth at most $2k$. Each patch occurs at the interface between two consecutive slabs[2], overlapping with $2\,\text{ɾ}$ levels from each adjacent slab. For fixed ɾ, each patch has constant branchwidth (at most $8\,\text{ɾ}$).

The algorithm proceeds as follows. Independently solve Capacitated ɾ-Domination restricted to each slab and each patch (with demands modified as described below). Then take the union of solutions over all slabs and patches. Since patches overlap the slabs, if a dominating vertex from a proper patch solution is also a dominating vertex from a proper slab solution, the union of solutions may no longer be proper. In fact the capacities required for some patch vertices have been potentially doubled. The algorithm alleviates this excess strain on patch vertices by reassigning the demand to *underfull* vertices elsewhere in the graph. We refer to this process as *smoothing*.

Analogously to the algorithm for the uncapacitated version described above, for each slab G_j^i and each patch H_l^i, modify the problem so that the vertices of the ɾ-topmost and ɾ-bottommost levels have no demand (excluding the top of the first slab and the bottom of the last slab). Since patches overlap the $2\,\text{ɾ}$-bottommost and $2\,\text{ɾ}$-topmost levels of slabs, every vertex still has its demand covered by at least one subgraph containing it. The branchwidth of the slab is at most $2k$, so for any fixed k the dynamic-programming algorithm described in Section 2.5 can be used to generate a solution to the modified problem on the slab in polynomial time.

Let $\mathcal{A}_{G_j^i}$ denote the optimal assignment for the modified Capacitated ɾ-Domination for slab

[2]Note that in order to keep the patches themselves from overlapping, we must ensure that $k \geq 4\,\text{ɾ}$.

G^i_j and let $\mathcal{A}_{H^i_j}$ denote the optimal assignment for patch H^i_j. Combining slab and patch assignments gives $\tilde{\mathcal{A}}_i = \left(\biguplus_j \mathcal{A}_{G^i_j}\right) \uplus \left(\biguplus_j \mathcal{A}_{H^i_j}\right)$. Since $\tilde{\mathcal{A}}_i$ may no longer be a proper assignment, the algorithm uses a *smoothing* procedure, SMOOTH, to generate a final solution $\mathcal{A}'_i$. SMOOTH first removes facility-client pairs from an input assignment $\mathcal{A}$ until it is proper. It does so by repeatedly finding an *overloaded* vertex v and removing a pair (v, u) from $\mathcal{A}$ until no overloaded vertex can be found. SMOOTH then reassigns any remaining unmet demand to *underfull* vertices in the graph with excess capacity by repeatedly finding alternating paths in the induced assignment graph (described in Lemma 2). This reassignment process is described in detail in the following lemmas. Finally, the algorithm returns the assignment $\mathcal{A}'_i$ with the minimum cost among all choices of shift i.

2.4.2 Lemmas for Theorem 1

Working toward proving Theorem 1, we analyze the algorithm proposed above and prove the following lemmas.

Lemma 1. *Given a graph G with demand function $dem(\cdot)$, capacity function $q(\cdot)$, and vertex subset $S \subseteq V$, if a proper, covering assignment exists for* CAPACITATED 1-DOMINATION, *then $q(S) + dem(\partial(S)) \geq dem(S)$.*

Proof. Assume that such a proper, covering assignment exists. For any subset of vertices $S \subseteq V$, the demand from non-frontier vertices of S must be met by vertices in S. Since a solution exists, S must have sufficient capacity. That is, $dem(S \setminus \partial(S)) \leq q(S)$. Furthermore, $dem(S \setminus \partial(S)) = dem(S) - dem(\partial(S))$. This gives, $dem(S) - dem(\partial(S)) \leq q(S)$, and the result follows. □

Lemma 2. *Given a proper assignment $\mathcal{A}'$ with unmet demand $t(\mathcal{A}') > 0$, if a proper, covering assignment exists for* CAPACITATED 1-DOMINATION, *then a new proper assignment $\mathcal{A}''$ can be found in linear time such that $cost(\mathcal{A}'') \leq cost(\mathcal{A}') + 1$ and $t(\mathcal{A}'') < t(\mathcal{A}')$.*

Proof. Let U be the set of vertices with unmet demand. We assume that these vertices are at full capacity, otherwise simply assign such an underfull vertex, v, to itself and satisfy the lemma by letting $\mathcal{A}'' = \mathcal{A}' \uplus \{(v, v)\}$. We also assume that a vertex $v \in U$ spends its entire capacity toward meeting its own demand, so (v, v) occurs $q(v)$ times in $\mathcal{A}'$. We can always reassign v's capacity in this way without increasing overall unmet demand or the size of $\mathcal{A}'$.

Use assignment $\mathcal{A}'$ to define an arc set on V such that $(u, v) \in \mathcal{A}'$ indicates an arc from facility u to client v, and for every vertex u, the outdegree $\Delta^-(u)$ is at most $q(u)$. The dominating set

$S_{dom}(\mathcal{A}')$ is $\{v \ : \ \Delta^-(v) > 0\}$, and a covering assignment is one in which, for every vertex u, the indegree $\Delta^+(u)$ equals $dem(u)$.

Consider the directed *assignment graph* $\mathscr{A}$ defined by this arc set. We can assume that $\mathscr{A}$ has no directed cycles (except for self-loops), otherwise every arc in such a cycle can be changed to a self-loop without changing the total unmet demand or the size of $\mathcal{A}'$.

Since a proper, covering assignment exists, there must be a set W of vertices that are not at full capacity.

Let $G' = (V', E')$ denote $G = (V, E)$ augmented with self-loops at every vertex. Formally, $V' = V$ and $E' = E \cup \{vv \ : \ v \in V\}$. We define a *semi-alternating* directed path $P = p_0, p_1, ..., p_l$ to be a path in G' such that $\forall i \ p_i p_{i+1} \in E'$ and $(p_i, p_{i+1}) \notin \mathcal{A}' \Leftrightarrow (p_{i+1}, p_{i+2}) \in \mathcal{A}'$. That is, P is a path through G' that alternates between *forward* arcs in the assignment graph $\mathscr{A}$ and edges in G' that are not forward arcs in $\mathscr{A}$ (see Figure 2.2).

We show that there exists such a *semi-alternating* directed path $P = p_0, p_1, ..., p_l$ from a vertex $p_0 \in U$ to a vertex $p_l \in W$. Such a path can be found in linear time using breadth-first search from vertices with insufficient demand met.

Assume to the contrary that no such path exists. Let U^+ be the set of all vertices $v \in V$ such that there exists a semi-alternating directed path from u to v for some $u \in U$. Clearly $U \subseteq U^+$, and by assumption $U^+ \cap W = \emptyset$, so all vertices of U^+ are at full capacity. Since each vertex in U spends all of its capacity on meeting its own demand, it has no outgoing arcs in $\mathscr{A}$, so the first edge of every semi-alternating directed path originating at some $u \in U$ is *not* a forward arc in $\mathscr{A}$. (If the first edge of the path is a self loop in $\mathscr{A}$, it must have a subpath with the same endpoints that is also semi-alternating and whose first edge is not in $\mathscr{A}$). Therefore all of u's neighbors are in U^+ so u cannot appear on the frontier $\partial_{G'}(U^+)$ and $U \cap \partial_{G'}(U^+) = \emptyset$. Similarly, any arcs of $\mathscr{A}$ *into* vertices in $\partial_{G'}(U^+)$ must start in $V' \setminus U^+$ otherwise U^+ could be extended to include neighbors of such vertices.

Since all of the demand of every vertex in $\partial_{G'}(U^+)$ is met, and all arcs of $\mathscr{A}$ into $\partial_{G'}(U^+)$ start in $V' \setminus U^+$, we infer that $V' \setminus U^+$ contributes a total of $dem(\partial_{G'}(U^+))$ toward meeting the demand of U^+. Furthermore all vertices in U^+ are at full capacity and there are no outgoing arcs of $\mathscr{A}$ in $\delta_{G'}(U^+)$ so U^+ contributes a total of $q(U^+)$ toward meeting its own total demand. Yet $U \subseteq U^+$ so U^+ contains unmet demand. Therefore $q(U^+) + dem(\partial_{G'}(U^+)) < dem(U^+)$ which by Lemma 1 contradicts our feasibility assumption.

Given a *semi-alternating* directed path $P = p_0, p_1, ..., p_l$ from a vertex $p_0 \in U$ to a vertex $p_l \in W$, we know that (p_0, p_1) is not in $\mathcal{A}'$ since p_0 is in U and does not contribute to meeting any neighbor's demand. We also assume that (p_{l-1}, p_l) is not in $\mathcal{A}'$ since if the path ends in an assignment arc we can append a self-loop. Reassign the arcs along this path as follows:

$$\mathcal{A}'' = (\mathcal{A}' \setminus \{(p_1, p_2), (p_3, p_4), ..., (p_{l-2}, p_{l-1})\}) \uplus \{(p_1, p_0), (p_3, p_2), ..., (p_{l-2}, p_{l-3}), (p_l, p_{l-1})\}$$

This gives $q(\mathcal{A}'') \leq q(\mathcal{A}') + 1$. All of the demand previously met along path P is still met although potentially by a different source, but $\mathcal{A}''$ also meets one additional unit of previously unmet demand at vertex p_0 (see Figure 2.2). Therefore the resulting proper assignment $\mathcal{A}''$ satisfies $t(A'') < t(A')$. □

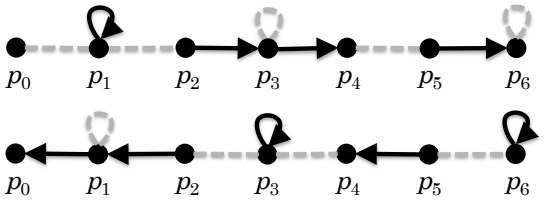


Figure 2.2: Black arrows depict assignment arcs and dotted gray lines depict edges of G'. The upper path shows a semi-alternating directed path $P = p_0, p_1, p_1, p_2, p_3, p_3, p_4, p_5, p_6, p_6$. The lower path shows the result of arc reassignment.

We now prove that the lemma can be extended to Capacitated ℓ-Domination.

Lemma 3. *Given a graph $G = (V, E)$ and a proper assignment $\mathcal{A}'$ with unmet demand $t(\mathcal{A}') > 0$, if a proper, covering assignment exists for* Capacitated ℓ-Domination *then a new proper assignment $\mathcal{A}''$ can be found in polynomial time such that $cost(\mathcal{A}'') \leq cost(\mathcal{A}') + 1$ and $t(\mathcal{A}'') < t(\mathcal{A}')$.*

Proof. Construct $G^* = (V, E^*)$ from G by adding an edge between every pair of ℓ-neighbors. That is, $E^* = E \cup \{e_{uv} \mid d_{hop}(u, v) \leq \ell\}$ where $d_{hop}(\cdot)$ is hop-distance in G. Any assignment $\mathcal{A}$ for Capacitated ℓ-Domination in G is also an assignment for Capacitated 1-Domination in G^*, and vice versa. Therefore applying Lemma 2 to G^* (using the same demand and capacity functions) results in a new assignment with the desired properties. □

The final requirement before proving Theorem 1 is an algorithm for solving Capacitated ℓ-Domination optimally in instances of bounded branchwidth. We state this result now and defer the proof until Section 2.5.

Theorem 2. *For any positive integers $\wr$ and ω,* Capacitated $\wr$-Domination *on instances with branchwidth ω can be solved exactly in $n^{O(\wr\omega)}$ time.*

2.4.3 Proof of Theorem 1

We now prove Theorem 1 which we restate here for convenience.

Theorem 1. *For any $\epsilon > 0$ and positive integers $\wr$ and $\hat{q}$, there exists a polynomial-time algorithm that, given an instance of* Capacitated $\wr$-Domination *on a planar graph with maximum capacity $\hat{q}$, finds a solution of cost at most $1+\epsilon$ times the optimum cost.*

Proof. Let OPT denote an optimal assignment for Capacitated $\wr$-Domination. For a fixed shift i let $OPT_{G^i_j} = OPT \cap G^i_j$ be the intersection of the optimal assignment with the j^{th} slab and $SOL_{G^i_j}$ denote the optimal assignment for the modified problem on the j^{th} slab. Similarly, let $OPT_{H^i_j} = OPT \cap H^i_j$ be the intersection of the optimal assignment with the j^{th} patch and $SOL_{H^i_j}$ denote the optimal assignment for the modified problem on the j^{th} patch.

Because we modify the problem to ignore demand from the $\wr$ outermost levels of a slab (resp. patch) $OPT_{G^i_j}$ (resp. $OPT_{H^i_j}$) is a covering assignment for the modified problem on G^i_j (resp. H^i_j) since all of the demand on the non-outermost levels will be covered by $OPT_{G^i_j}$ (resp. $OPT_{H^i_j}$). Therefore $cost(SOL_{G^i_j}) \leq cost(OPT_{G^i_j})$ and $cost(SOL_{H^i_l}) \leq cost(OPT_{H^i_l})$.

Our algorithm first determines the union, $\tilde{\mathcal{A}}$, of assignments on slabs and patches. The size of this union is at most the sum of the sizes of the assignments for each slab and each patch. The algorithm then removes and reassigns at most $\hat{q}$ assignment pairs for each duplicated vertex in the union (where $\hat{q}$ is the value of the maximum capacity). Duplicated vertices can only occur within the levels of the patches. By Lemma 3 each reassignment adds at most one vertex to the dominating set. Let $\mathcal{A}'$ be the final assignment output by the algorithm. Therefore

$$\begin{aligned} cost(\mathcal{A}') &\leq \sum_j cost(SOL_{G^i_j}) + \hat{q}\sum_j cost(SOL_{H^i_j}) \\ &\leq \sum_j cost(OPT_{G^i_j}) + \hat{q}\sum_j cost(OPT_{H^i_j}) \\ &\leq cost(OPT) + \hat{q}\sum_j cost(OPT_{H^i_j}) \end{aligned}$$

The final inequality comes from the slabs, and therefore also their intersections with OPT, being

disjoint. Let i^* be the shift that minimizes the sum, $\sum_j cost(OPT_{H_j^i})$, of the intersection of patches with the optimal dominating set $S_{dom}(OPT)$.

If $\sum_j cost(OPT_{H_j^{i^*}})$ were to exceed $(4\ell/k)cost(OPT)$ then the sum of patch intersections over all shifts, $\sum_i \sum_j cost(OPT_{H_j^i})$ would exceed $4\ell cost(OPT)$ since i^* was chosen from k possible shifts to give the minimum such sum. But each vertex in $S_{dom}(OPT)$ occurs in a patch for at most 4ℓ different shifts so can contribute at most 4ℓ to the sum, giving $\sum_i \sum_j cost(OPT_{H_j^i}) \leq 4\ell cost(OPT)$. Therefore, $\sum_j cost(OPT_{H_j^{i^*}}) \leq (4\ell/k)cost(OPT)$, giving

$$cost(\mathcal{A}') \leq cost(OPT) + (4\ell\hat{q}/k)cost(OPT)$$

Setting k to $4\ell\hat{q}/\epsilon$ gives a $(1+\epsilon)$-approximation.

The slabs can be solved in $n^{O(k\ell)}$ time and patches can be solved in $n^{O(\ell^2)}$ time using the dynamic program described in Section 2.5. The algorithm then performs at most $\epsilon\hat{q}n$ reassignments, each of which can be computed in linear time. This process is repeated k times, giving an overall runtime dominated by $n^{O(\ell^2\hat{q}/\epsilon)}$, which, for fixed ℓ, $\hat{q}$, and ϵ, is polynomial. □

2.4.4 Corollaries and Extensions

In fact, the PTAS of Theorem 1 can be improved for Capacitated Domination (the special case of $\ell = 1$). If there is also a bound $\widehat{dem}$ on the maximum demand, a much simpler dynamic programming algorithm can be used to solve instances of bounded branchwidth (see Section 2.5.2).

Theorem 3. *For any positive integers ω, $\hat{q}$, and $\widehat{dem}$,* Capacitated Domination *on instances with branchwidth ω, maximum capacity $\hat{q}$, and maximum demand $\widehat{dem}$ can be solved exactly in $(\hat{q} \cdot \widehat{dem})^{O(\omega)}n$ time.*

Note in particular that such an algorithm is FPT when parameterized by ω, $\hat{q}$, and $\widehat{dem}$, as the degree of the polynomial no longer depends on any of these. This immediately yields the following EPTAS as a corollary.

Theorem 4. *For any positive integers $\hat{q}$ and $\widehat{dem}$, there exists an EPTAS for* Capacitated Domination *on instances with maximum capacity $\hat{q}$ and maximum demand $\widehat{dem}$.*

We observe that the capacity of a vertex never need exceed the sum of the demands in its closed neighborhood and the demand of a vertex can never exceed the sum of the capacities in its closed

neighborhood. Therefore to achieve the EPTAS of Theorem 4 it is sufficient for any two of the following to be bounded: maximum capacity, maximum demand, maximum vertex degree.

Capacitated Vertex Cover

In Capacitated Vertex Cover there is a capacity function $q : V \to \mathbb{Z}^*$ on the vertices of G and a demand function $dem : E \to \mathbb{Z}^*$ on the edges of G. Under the constraint that edge-demand can only be met by its own endpoints, the objective is to find a set of vertices S_{dom} of minimum size such that every edge has its demand covered by vertices in S_{dom} and each vertex in S_{dom} covers a total amount of demand that does not exceed its capacity. Capacitated Vertex Cover generalizes the traditional Vertex Cover problem in which each vertex has unlimited capacity and each edge has a demand of one. Furthermore, Capacitated Vertex Cover is identical to Capacitated Domination with the demand on edges instead of on vertices. In the following corollary, we show that we can reduce Capacitated Vertex Cover to Capacitated Domination to achieve a PTAS for the former.

Corollary 1. *For any positive integer $\hat{q}$ there exists an EPTAS for* Capacitated Vertex Cover *with maximum capacity $\hat{q}$ in planar graphs.*

Proof. Consider the following reduction from Capacitated Vertex Cover to Capacitated Domination. Let G' be the graph formed by bisecting every edge of G. Specifically, we define a new graph G' with edge set E', vertex set V', demand function $dem : V' \to \mathbb{Z}^*$, and capacity function $q : V' \to \mathbb{Z}^*$, as follows:

For every edge $e_{uv} \in E$, add edges e_{uw} and e_{wv} to E' and add vertices u, v, w to V'. Additionally, set capacities to $q'(u) = q(u)$, $q'(v) = q(v)$, and $q(w) = 0$, and set demands to $dem'(w) = dem(e)$ and $dem'(u) = dem'(v) = 0$. This provides the necessary conditions to solve Capacitated Domination on G'.

It is easy to see that a dominating set of G' exactly corresponds to a vertex cover of G. Setting the bisecting vertex to have zero capacity guarantees that these vertices will not be chosen in the minimum dominating set. When the maximum capacity is bounded by $\hat{q}$, an EPTAS for Capacitated Vertex Cover follows as a corollary to Theorem 4, by noting that edge demand cannot exceed the sum of the capacity of its endpoints, so maximum demand is bounded by $2\hat{q}$.

□

2.5 Solving Capacitated ℓ-Domination for Bounded Branchwidth

In this section we describe a dynamic-programming algorithm for solving Capacitated ℓ-Domination in graphs with bounded branchwidth and maximum capacity $\hat{q}$. Even though an algorithm for Capacitated Domination follows from this algorithm, we additionally present a much simpler and more efficient dynamic program for this problem when the maximum demand is also bounded by some $\widehat{dem}$.

2.5.1 Dynamic Program for Capacitated ℓ-Domination

In any proper, covering assignment $\mathcal{A}$, each facility-client pair $(u, v) \in \mathcal{A}$ indicates one unit of demand at vertex v is met by one unit of capacity at vertex u within a distance ℓ. We call the shortest v-to-u path in G the (directed) *resolution*[3] path for this unit of demand. We design an algorithm that finds the resolution paths corresponding to an optimal assignment.

Although designing a dynamic program for Capacitated Domination is fairly straightforward, there are two main challenges with designing one for Capacitated ℓ-Domination. First (as described in Demaine et al. for *uncapacitated* ℓ-Domination) the resolution paths can wind up and down the branch decomposition tree, so the algorithm needs a way to convey information across levels [37]. The second challenge is that, even though the capacity of a vertex v on the boundary of a cluster may be bounded, the number of resolution paths that cross the boundary at v may be quite large, and unlike the uncapacitated version, we cannot merge these paths. We address these challenges by extending the dynamic program used for the uncapacitated version [37].

Consider an assignment $\mathcal{A}$ and the corresponding resolution paths. For each cluster in the branch decomposition, some number of these resolution paths cross the cluster boundary at each boundary vertex. Each such path has a *direction* (crossing *into* or *out of* the cluster) and a *magnitude* (path length remaining until facility is reached). Our dynamic-programming table, DP, is indexed by *configurations* $(\mathcal{C}, h_{\mathcal{C}})$ where $\mathcal{C}$ is a cluster and $h_{\mathcal{C}}$ is a function that specifies how many paths of each direction and magnitude cross at each vertex of a cluster boundary.

Specifically, the algorithm enumerates all functions of the form $h_{\mathcal{C}} : \partial(\mathcal{C}) \times [-\ell, \ell] \to [0, \hat{q}n]$, where $h_{\mathcal{C}}(v, i)$ indicates the number of resolution paths that cross $\mathcal{C}$ at v with magnitude $|i|$ and

[3] We adopt this word from Demaine et al. [37]

direction $sign(i)$. If $i < 0$ the path crosses *into* $\mathcal{C}$ and if $i > 0$ the path crosses *out of* $\mathcal{C}$. Note that the sign only indicates the direction of the next vertex in the resolution path and not necessarily the direction of the endpoint of the path (a path may enter and/or leave the cluster again elsewhere). If $i = 0$ the resolution path ends at v, so $h_{\mathcal{C}}(v, 0) > 0$ indicates that v itself is in the dominating set. Therefore $h_{\mathcal{C}}(v, 0)$ equals $|\{(v, u) \in \mathcal{A} : u \in V\}|$ and must not exceed $q(v)$.

We say that assignment $\mathcal{A}$ is *consistent* with $h_{\mathcal{C}}$, if $h_{\mathcal{C}}$ exactly describes how the resolution paths corresponding to $\mathcal{A}$ cross the boundary of cluster $\mathcal{C}$. The DP entry at index $(\mathcal{C}, h_{\mathcal{C}})$ is the set $\mathcal{C} \cap S_{dom}(\mathcal{A})$ with minimum cardinality over all proper, covering assignments $\mathcal{A}$ that are consistent with $h_{\mathcal{C}}$. The algorithm proceeds level-by-level through the decomposition, starting at the leaves and continuing rootward.

The leaves of the decomposition correspond to single edges in the graph. For the leaf cluster $\mathcal{C}$ of edge e_{uv}, a configuration is *valid* if it meets the following criteria.

- Any resolution path crossing *into* $\mathcal{C}$ at u (resp v) with i hops remaining must cross *out of* $\mathcal{C}$ at v (resp u) with $i - 1$ hops remaining. That is, for $i < 0$, $h_{\mathcal{C}}(u, i) \leq h_{\mathcal{C}}(v, -i - 1)$ and $h_{\mathcal{C}}(v, i) \leq h_{\mathcal{C}}(u, -i - 1)$.

- The resolution paths crossing $\mathcal{C}$ at u (resp v) must include a path for each unit of demand at u (resp v). That is, $\Sigma_i h_{\mathcal{C}}(u, i) \geq dem(u)$ and $\Sigma_i h_{\mathcal{C}}(v, i) \geq dem(v)$.

- The number of paths ending at u (resp v) cannot exceed the capacity of u (resp v). That is, $h_{\mathcal{C}}(u, 0) \leq q(u)$ and $h_{\mathcal{C}}(v, 0) \leq q(v)$.

Moving rootward, we show how to determine the table entry for a parent cluster given the table entries for the child clusters. Consider a parent cluster $\mathcal{C}_0$ with child clusters $\mathcal{C}_1$ and $\mathcal{C}_2$. We divide the cluster boundaries into four regions and handle each region separately.

Let $\partial_{1,2} = (\partial(\mathcal{C}_1) \cap \partial(\mathcal{C}_2)) \setminus \partial(\mathcal{C}_0)$; $\partial_{0,1} = (\partial(\mathcal{C}_0) \cap \partial(\mathcal{C}_1)) \setminus \partial(\mathcal{C}_2)$; $\partial_{0,2} = (\partial(\mathcal{C}_0) \cap \partial(\mathcal{C}_2)) \setminus \partial(\mathcal{C}_1)$; and $\partial_{0,1,2} = \partial(\mathcal{C}_0) \cap \partial(\mathcal{C}_1) \cap \partial(\mathcal{C}_2)$ (refer to Figure 2.3).

Indices $(\mathcal{C}_1, h_{\mathcal{C}_1})$ and $(\mathcal{C}_2, h_{\mathcal{C}_2})$ are *compatible* with $(\mathcal{C}_0, h_{\mathcal{C}_0})$ if the following hold:

- Resolution paths crossing $\partial_{1,2}$ cross *into* one cluster and *out of* the other, so $h_{\mathcal{C}_1}$ and $h_{\mathcal{C}_2}$ agree on magnitude and disagree on sign. That is, $h_{\mathcal{C}_1}(v, i) = h_{\mathcal{C}_2}(v, -i)$, $\forall i, v \in \partial_{1,2}$.

- Resolution paths crossing $\partial_{0,1}$ cross the $\mathcal{C}_1$ boundary the same way they cross the $\mathcal{C}_0$ boundary, so $h_{\mathcal{C}_0}$ must agree with $h_{\mathcal{C}_1}$. That is $h_{\mathcal{C}_0}(v, i) = h_{\mathcal{C}_1}(v, i)$, $\forall i, v \in \partial_{0,1}$. The same holds for $\partial_{0,2}$.

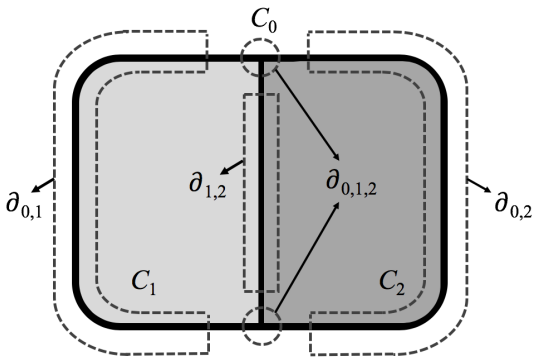


Figure 2.3: Depiction of parent ($\mathcal{C}_0$) and children ($\mathcal{C}_1$ and $\mathcal{C}_2$) boundary partitioning.

- Resolution paths crossing vertices on $\partial_{0,1,2}$ are more difficult to address.
  - All three clusters must agree on the number of resolution paths ending on $\partial_{0,1,2}$, so $h_{\mathcal{C}_1}(v,0) = h_{\mathcal{C}_2}(v,0) = h_{\mathcal{C}_0}(v,0)$, $\forall i, v \in \partial_{0,1,2}$.
  - Each resolution path that crosses out of (resp into) $\mathcal{C}_0$ must correspond to a path crossing out of (resp into) either $\mathcal{C}_1$ or $\mathcal{C}_2$, so $h_{\mathcal{C}_0}(v,i) \leq h_{\mathcal{C}_1}(v,i) + h_{\mathcal{C}_2}(v,i)$.
  - Similarly, each path that crosses $\mathcal{C}_1$ (resp $\mathcal{C}_2$) must correspond to a path crossing either $\mathcal{C}_0$ in the same direction or $\mathcal{C}_2$ (resp $\mathcal{C}_1$) in the opposite direction, so $h_{\mathcal{C}_1}(v,i) \leq h_{\mathcal{C}_0}(v,i) + h_{\mathcal{C}_2}(v,-i)$ and $h_{\mathcal{C}_2}(v,i) \leq h_{\mathcal{C}_0}(v,i) + h_{\mathcal{C}_1}(v,-i)$.
  - Finally, the net total of crossings in and out must agree. That is $h_{\mathcal{C}_0}(v,i) - h_{\mathcal{C}_0}(v,-i) = h_{\mathcal{C}_1}(v,i) + h_{\mathcal{C}_2}(v,i) - h_{\mathcal{C}_1}(v,-i) - h_{\mathcal{C}_2}(v,-i)$.

The entry for each valid configuration is the set of vertices in the cluster that are resolution path endpoints. For leaf cluster $\mathcal{C}$ corresponding to edge (u,v), this is $\{w \in \{u,v\} \mid h_{\mathcal{C}}(w,0) > 0\}$. To determine the entry for configuration $(\mathcal{C}_0, h_{\mathcal{C}_0})$ the algorithm searches over all compatible pairs $(\mathcal{C}_1, h_{\mathcal{C}_1})$ and $(\mathcal{C}_2, h_{\mathcal{C}_2})$ and finds $DP(\mathcal{C}_1, h_{\mathcal{C}_1}) \cup DP(\mathcal{C}_2, h_{\mathcal{C}_2})$ such that the cardinality is minimized.

Let $\mathcal{C}_{G_0}$ denote the cluster containing all edges of input graph G_0 and let $h_\emptyset$ be the trivial empty function. An optimal proper, covering dominating set is stored in $DP(\mathcal{C}_{G_0}, h_\emptyset)$.

The assignment itself can then be found by setting the capacity of all vertices not in the dominating set to zero and then using the smoothing technique described in Section 2.4.1 to assign demand to the vertices in the dominating set.

We now prove how the dynamic-programming algorithm described above satisfies Theorem 2, which we restate here.

Theorem 2. *For any positive integers $\mathscr{r}$ and ω,* Capacitated $\mathscr{r}$-Domination *on instances with branchwidth ω can be solved exactly in $n^{O(\mathscr{r}\omega)}$ time.*

Proof. We proceed by induction on the branch decomposition.

Our base cases are the leaves of the decomposition. Let edge (u, v) be such a leaf cluster, $\mathcal{C}$. The three criteria we impose on the base case configurations are rationalized as follows:

- We can assume all resolution paths are simple since they are shortest paths, so we can ignore resolution paths that enter and exit a cluster at the same vertex. Any resolution path that enters at u must exit (or end) at v, shortening the remaining path length by one. For $i < 0$, each of the $h_{\mathcal{C}}(u, i)$ paths entering $\mathcal{C}$ at u must be accounted for in the $h_{\mathcal{C}}(v, -i-1)$ paths exiting (or ending) at v. So $h_{\mathcal{C}}(v, -i-1) \geq h_{\mathcal{C}}(u, i)$ and a symmetric argument shows $h_{\mathcal{C}}(u, -i-1) \geq h_{\mathcal{C}}(v, i)$.

- We require that for each unit of demand at a given vertex there is a resolution path originating from that vertex. We account for these paths in the base case, so the number of all paths passing through a vertex is at least the size of the demand, giving $\Sigma_i h_{\mathcal{C}}(u, i) \geq dem(v)$ and $\Sigma_i h_{\mathcal{C}}(u, i) \geq dem(u)$.

- Finally, we impose the obvious restriction that the number of resolution paths that end at a vertex cannot exceed the capacity of that vertex, giving $h_{\mathcal{C}}(u, 0) \leq q(v)$ and $h_{\mathcal{C}}(u, 0) \leq q(u)$.

The cost of any configuration for $\mathcal{C}$ is the number of vertices w in $\{u, v\}$ for which $h_{\mathcal{C}}(w, 0) > 0$, which can be determined in constant time.

To show the inductive step, we consider a cluster $\mathcal{C}_0$ and assume that an optimal dominating set for a resolution-path-respecting assignment has been found by the dynamic program for its child clusters $\mathcal{C}_1$ and $\mathcal{C}_2$. To determine the entry in the table for index $(\mathcal{C}_0, h_{\mathcal{C}_0})$ the algorithm searches over all compatible pairs $(\mathcal{C}_1, h_{\mathcal{C}_1})$ and $(\mathcal{C}_2, h_{\mathcal{C}_2})$ and finds $DP(\mathcal{C}_1, h_{\mathcal{C}_1}) \cup DP(\mathcal{C}_2, h_{\mathcal{C}_2})$ such that the cardinality is minimized. This cardinality is the sum of the cost of the child clusters minus the cost (number of vertices in the dominating set) of the intersection of the child clusters (to avoid double-counting).

To verify our notion of configuration compatibility, consider the following cases:

- If $v \in (\partial(\mathcal{C}_1) \cap \partial(\mathcal{C}_2)) \setminus \partial(\mathcal{C}_0)$ then any resolution path crossing out of $\mathcal{C}_1$ at v must correspond to a path of the same length crossing *into* $\mathcal{C}_2$ at v, and vice versa. Therefore any compatible

configurations must have $h_{\mathcal{C}_1}(v,i) = h_{\mathcal{C}_2}(v,-i)$, $\forall i$.

- If $v \in (\partial(\mathcal{C}_0) \cap \partial(\mathcal{C}_1)) \setminus \partial(\mathcal{C}_2)$, then any resolution path crossing out of (resp. into) $\mathcal{C}_0$ at v must also cross out of (resp. into) $\mathcal{C}_1$ at v. Therefore any compatible configurations must have $h_{\mathcal{C}_0}(v,i) = h_{\mathcal{C}_1}(v,i)$, $\forall i$.
- Similarly, if $v \in (\partial(\mathcal{C}_0) \cap \partial(\mathcal{C}_2)) \setminus \partial(\mathcal{C}_1)$, $h_{\mathcal{C}_0}(v,i) = h_{\mathcal{C}_2}(v,i)$, $\forall i$.
- Finally, if $v \in \partial(\mathcal{C}_0) \cap \partial(\mathcal{C}_1) \cap \partial(\mathcal{C}_2)$, then we consider the four criteria:
  - If v is in the dominating set, then all three configurations must agree on this. Moreover, the configurations must all agree on *how many* resolutions path terminate at v, so $h_{\mathcal{C}_1}(v,0) = h_{\mathcal{C}_2}(v,0) = h_{\mathcal{C}_0}(v,0)$.
  - If $i > 0$ each resolution path with remaining length i crossing out of $\mathcal{C}_0$ at v must correspond to a path crossing out of $\mathcal{C}_1$ or $\mathcal{C}_2$ and if $i < 0$ each resolution path with remaining length i crossing into $\mathcal{C}_0$ at v must correspond to a path crossing into $\mathcal{C}_1$ or $\mathcal{C}_2$. So $h_{\mathcal{C}_0}(v,i) \leq h_{\mathcal{C}_1}(v,i) + h_{\mathcal{C}_2}(v,i)$.
  - Similarly, if $i > 0$ each resolution path with remaining length i crossing out of $\mathcal{C}_1$ at v must correspond to a path crossing out of $\mathcal{C}_0$ or into $\mathcal{C}_2$ and if $i < 0$ each resolution path with remaining length i crossing into $\mathcal{C}_1$ at v must correspond to a path crossing into $\mathcal{C}_0$ or out of $\mathcal{C}_2$. So $h_{\mathcal{C}_1}(v,i) \leq h_{\mathcal{C}_0}(v,i) + h_{\mathcal{C}_2}(v,-i)$. Symmetrically, $h_{\mathcal{C}_2}(v,i) \leq h_{\mathcal{C}_0}(v,i) + h_{\mathcal{C}_1}(v,-i)$.
  - The difference between the number of resolution paths with remaining length i crossing into and out of $\mathcal{C}_0$ at v must be the same as the difference between the number of resolution paths with remaining length i crossing into $\mathcal{C}_1$ and $\mathcal{C}_2$ and out of $\mathcal{C}_1$ and $\mathcal{C}_2$. Therefore $h_{\mathcal{C}_0}(v,i) - h_{\mathcal{C}_0}(v,-i) = h_{\mathcal{C}_1}(v,i) + h_{\mathcal{C}_2}(v,i) - h_{\mathcal{C}_1}(v,-i) - h_{\mathcal{C}_2}(v,-i)$.

Since the branch decomposition gives a binary tree with $O(n)$ leaves, there are $O(n)$ clusters. For each cluster, $\mathcal{C}$, the boundary $\partial(\mathcal{C})$ has at most ω vertices. For each vertex $v \in \partial(\mathcal{C})$ there are $(\hat{q}n)^{2\hat{q}}\hat{q}$ options for the value of $h_{\mathcal{C}}$. There are thus at most $((\hat{q}n)^{2\hat{q}}\hat{q})^{\omega}$ configurations for each cluster boundary. This gives $O((\hat{q}n)^{2\hat{q}\omega}\hat{q}^{\omega} \cdot n)$ indices of the DP table. Each entry requires comparing unions of entries from all compatible child configuration pairs. There are $O((\hat{q}n)^{2\hat{q}\omega}\hat{q}^{\omega} \cdot (\hat{q}n)^{2\hat{q}})$ such pairs since knowing the configurations for one child and the parent almost completely determines the configuration for the other child. The only choice for the second child is for the function at

the (at most two[4]) vertices at the intersection of all three boundaries, and since for each i and v the difference between $h_{\mathcal{C}}(v,i)$ and $h_{\mathcal{C}}(v,-i)$ is already determined, there are at most $\hat{q}n$ options for each of $\wr$ values of $i > 0$ for each of at most two vertices. Therefore there are at most $(\hat{q}n)^{2\wr}$ choices for the configuration of the second child cluster. Checking for compatibility requires $O(\omega)$ time. This gives the algorithm a total runtime of $O(\omega(\hat{q}^{2\wr\omega+\wr+\omega})^2 \cdot n^{4\wr\omega+2\wr+1})$.

□

2.5.2 Dynamic Program for Capacitated Domination

Although an algorithm for solving Capacitated Domination exactly on graphs of bounded branch-width follows from Theorem 2, we present a more efficient dynamic program for instances in which the maximum demand $\widehat{dem}$ is bounded.

We use the branch decomposition of G to guide the dynamic program. For each cluster $\mathcal{C}$ in the decomposition, the algorithm enumerates all functions $\mathcal{Q}, \mathcal{D}em : \partial(\mathcal{C}) \to \mathbb{Z}$ such that $\forall v \in \partial(\mathcal{C})$, $0 \leq \mathcal{Q}(v) \leq q(v)$ and $0 \leq \mathcal{D}em(v) \leq dem(v)$. Our dynamic-programming table, DP, is indexed by triplets $(\mathcal{C}, \mathcal{Q}, \mathcal{D}em)$ over all such functions and clusters.

The entry at index $(\mathcal{C}, \mathcal{Q}, \mathcal{D}em)$ is an optimal assignment $\mathcal{A}$ for cluster $\mathcal{C}$ such that for vertices on the boundary $\partial(\mathcal{C})$, $\mathcal{Q}(v)$ indicates the *exact* amount of capacity that v contributes to vertices in cluster $\mathcal{C}$ and $\mathcal{D}em(v)$ indicates the *exact* amount of demand from v that is met by vertices in cluster $\mathcal{C}$. We call this a *restricted assignment*. The algorithm proceeds level-by-level through the decomposition, starting at the leaves and continuing rootward.

The algorithm first handles the leaves of the decomposition, which correspond to single edges in the graph. For the leaf cluster of edge (u, v), there are three cases to consider:

- If $\mathcal{Q}(v) + \mathcal{Q}(u) \neq \mathcal{D}em(u) + \mathcal{D}em(v)$, there is no solution because the cluster's contributed capacity does not equal its covered demand. The algorithm returns $\{(u,v)^{\infty}\}$.

- If v has sufficient capacity to cover its own demand ($\mathcal{Q}(v) \geq \mathcal{D}em(v)$) it does so and then uses any residual capacity to cover some of the demand from u. Any remaining uncovered demand at u is covered by u itself. Since $\mathcal{Q}(v) + \mathcal{Q}(u) = \mathcal{D}em(u) + \mathcal{D}em(v)$ the algorithm returns $\{(v,v)^{\mathcal{D}em(v)}, (v,u)^{\mathcal{Q}(v)-\mathcal{D}em(v)}, (u,u)^{\mathcal{Q}(u)}\}$.

[4]By planarity, we can assume that the branch decomposition is a *sphere-cut decomposition* and that there are at most two vertices at the intersection of all three boundaries [39].

- Otherwise, u has sufficient capacity to cover its own demand ($\mathcal{Q}(u) > \mathcal{D}em(u)$) and, symmetrically, the algorithm returns $\{(u,u)^{\mathcal{D}em(u)}, (u,v)^{\mathcal{Q}(u)-\mathcal{D}em(u)}, (v,v)^{\mathcal{Q}(v)}\}$.

Moving rootward, the restricted assignment for a parent cluster is computed using the table entries for the child clusters. Consider a cluster $\mathcal{C}_0$ with child clusters $\mathcal{C}_1$ and $\mathcal{C}_2$. Boundaries $\partial_{1,2}$, $\partial_{0,1}$, $\partial_{0,2}$, and $\partial_{0,1,2}$ are as defined in Section 2.5.1 (see Figure 2.3).

We say $(\mathcal{C}_1, \mathcal{Q}_1, \mathcal{D}em_1)$ and $(\mathcal{C}_2, \mathcal{Q}_2, \mathcal{D}em_2)$ are *compatible* with $(\mathcal{C}_0, \mathcal{Q}_0, \mathcal{D}em_0)$ if the following hold:

- For all v on $\partial_{1,2}$: $\mathcal{Q}_1(v) + \mathcal{Q}_2(v) \leq q(v)$ and $\mathcal{D}em_1(v) + \mathcal{D}em_2(v) = dem(v)$.
- For all v on $\partial_{0,1}$: $\mathcal{Q}_0(v) = \mathcal{Q}_1(v)$ and $\mathcal{D}em_0(v) = \mathcal{D}em_1(v)$. (Similarly for all v on $\partial_{0,2}$: $\mathcal{Q}_0(v) = \mathcal{Q}_2(v)$ and $\mathcal{D}em_0(v) = \mathcal{D}em_2(v)$.)
- For all v on $\partial_{0,1,2}$: $\mathcal{Q}_1(v) + \mathcal{Q}_2(v) = \mathcal{Q}_0(v)$ and $\mathcal{D}em_1(v) + \mathcal{D}em_2(v) = \mathcal{D}em_0(v)$.

To determine the entry in the table for index $(\mathcal{C}_0, \mathcal{Q}_0, \mathcal{D}em_0)$ the algorithm searches over all compatible pairs $(\mathcal{C}_1, \mathcal{Q}_1, \mathcal{D}em_1)$ and $(\mathcal{C}_2, \mathcal{Q}_2, \mathcal{D}em_2)$ with respective entries $\mathcal{A}_1$ and $\mathcal{A}_2$, and finds the $\mathcal{A}_1 \uplus \mathcal{A}_2$ such that $cost(\mathcal{A}_1 \uplus \mathcal{A}_2)$ is minimized.

Letting $\mathcal{C}_{G_0}$ denote the cluster containing all edges of G and $\mathcal{Q}_\emptyset$ and $\mathcal{D}em_\emptyset$ denote the trivial empty functions, the algorithm returns the assignment stored in $DP(\mathcal{C}_G, \mathcal{Q}_\emptyset, \mathcal{D}em_\emptyset)$.

Theorem 3. *For any positive integers ω, $\hat{q}$, and $\widehat{dem}$,* Capacitated Domination *on instances with branchwidth ω, maximum capacity $\hat{q}$, and maximum demand $\widehat{dem}$ can be solved exactly in $(\hat{q} \cdot \widehat{dem})^{O(\omega)} n$ time.*

Proof. We claim that the dynamic program described above is such an algorithm. We prove correctness using induction on the branch decomposition.

Our base cases are the leaves of the decomposition. Let edge e_{uv} be such a leaf. We consider the same three cases addressed in the algorithm description.

- If $\mathcal{Q}(v) + \mathcal{Q}(u) \neq \mathcal{D}em(u) + \mathcal{D}em(v)$ the algorithm returns $\{(u,v)^\infty\}$ because the total capacity used must equal the total demand covered for the cluster to have a feasible solution.
- Otherwise if $\mathcal{Q}(v) \geq \mathcal{D}em(v)$ the algorithm returns $\{(v,v)^{\mathcal{D}em(v)}, (v,u)^{\mathcal{Q}(v)-\mathcal{D}em(v)}, (u,u)^{\mathcal{Q}(u)}\}$. Since $\mathcal{Q}(v) + \mathcal{Q}(u) = \mathcal{D}em(u) + \mathcal{D}em(v)$, the demand from v and u is covered and the exact capacity is used. If both $\mathcal{Q}(v)$ and $\mathcal{Q}(u)$ equal zero, the empty assignment is returned. If

exactly one of $\mathcal{Q}(v)$ and $\mathcal{Q}(u)$ equals zero, the size of the dominating set for the returned assignment is one. Otherwise the size is two. Such an assignment is therefore a minimum.

- Otherwise $\mathcal{Q}(v) < \mathcal{D}em(v)$, and the algorithm returns $\{(u,u)^{\mathcal{D}em(u)}, (u,v)^{\mathcal{Q}(u)-\mathcal{D}em(u)}, (v,v)^{\mathcal{Q}(v)}\}$. This case is symmetric to the previous one.

To show the inductive step, we consider a cluster $\mathcal{C}_0$ and assume that the restricted assignment for Capacitated Domination has been solved optimally by the dynamic program for its child clusters $\mathcal{C}_1$ and $\mathcal{C}_2$. For index $(\mathcal{C}_0, \mathcal{Q}_0, \mathcal{D}em_0)$ the algorithm searches over all compatible pairs $(\mathcal{C}_1, \mathcal{Q}_1, \mathcal{D}em_1)$ and $(\mathcal{C}_2, \mathcal{Q}_2, \mathcal{D}em_2)$ with respective entries $\mathcal{A}_1$ and $\mathcal{A}_2$ and returns the $\mathcal{A}_1 \uplus \mathcal{A}_2$ such that $cost(\mathcal{A}_1 \uplus \mathcal{A}_2)$ is minimized. Let $\mathcal{A}_1^* \uplus \mathcal{A}_2^*$ be such a minimum where $\mathcal{A}_1^*$ is the entry for $(\mathcal{C}_1^*, \mathcal{Q}_1^*, \mathcal{D}em_1^*)$ and $\mathcal{A}_2^*$ is the entry for $(\mathcal{C}_2^*, \mathcal{Q}_2^*, \mathcal{D}em_2^*)$.

To show that $\mathcal{A}_1^* \uplus \mathcal{A}_2^*$ is an optimal restricted assignment for $(\mathcal{C}_0, \mathcal{Q}_0, \mathcal{D}em_0)$, we must show that boundary restrictions are met and that it is proper, covering, and minimum.

By definition of compatible, boundary restrictions are automatically satisfied because every vertex on the boundary of $\mathcal{C}_0$ must appear on the boundary of at least one of its children and compatibility forces agreement between the child (or children for vertices on all three boundaries) and the parent.

Since $\mathcal{A}_1^*$ and $\mathcal{A}_2^*$ are each proper, and compatibility requires that $\mathcal{Q}_1^*(v) + \mathcal{Q}_2^*(v)$ is at most $q(v)$ for each vertex v on the shared boundary, $\mathcal{A}_1^* \uplus \mathcal{A}_2^*$ must be proper. Similarly, since $\mathcal{A}_1^*$ and $\mathcal{A}_2^*$ are each covering, and compatibility requires that $\mathcal{D}em_1^*(v) + \mathcal{D}em_2^*(v)$ equals $dem(v)$ for each vertex v on the shared boundary, $\mathcal{A}_1^* \uplus \mathcal{A}_2^*$ must be covering as well.

Let $\mathcal{A}_0^*$ be a minimum restricted assignment for $(\mathcal{C}_0, \mathcal{Q}_0, \mathcal{D}em_0)$. Partition $\mathcal{A}_0^*$ into X and Y as follows:

- For $(v,v) \in \mathcal{A}_0^*$, with $v \in \partial(\mathcal{C}_1) \cap \partial(\mathcal{C}_2)$, put (v,v) into either X or Y arbitrarily.
- Otherwise for $(v,v) \in \mathcal{A}_0^*$, put (v,v) in X if v is the endpoint of an edge in $\mathcal{C}_1$ and in Y if v is the endpoint of an edge in $\mathcal{C}_2$.
- Otherwise for $(u,v) \in \mathcal{A}_0^*$, put (u,v) in X if edge e_{uv} is in $\mathcal{C}_1$ and in Y if edge e_{uv} is in $\mathcal{C}_2$.

Clearly, every pair in assignment $\mathcal{A}_0^*$ must be in exactly one of X and Y so $\mathcal{A}_0^* = X \uplus Y$.

Let $\mathcal{Q}_x : \partial(\mathcal{C}_1) \to \mathbb{Z}$ be the function that maps boundary vertices of $\mathcal{C}_1$ to the capacity utilized by these vertices in X, and $\mathcal{D}em_x : \partial(\mathcal{C}_1) \to \mathbb{Z}$ be the function that maps boundary vertices of $\mathcal{C}_1$

to the demand from these vertices covered in X. By construction, X is now a restricted assignment for $(\mathcal{C}_1, \mathcal{Q}_x, \mathcal{D}em_x)$. Our inductive hypothesis gives $q(DP(\mathcal{C}_1, \mathcal{Q}_x, \mathcal{D}em_x)) \leq q(X)$. Symmetrically, defining $\mathcal{Q}_y$ and $\mathcal{D}em_y$ similarly for $\partial(\mathcal{C}_2)$ gives $q(DP(\mathcal{C}_2, \mathcal{Q}_y, \mathcal{D}em_y)) \leq q(Y)$.

$DP(\mathcal{C}_1, \mathcal{Q}_x, \mathcal{D}em_x)$ and X agree on the dominating vertices on the boundaries, as do $DP(\mathcal{C}_2, \mathcal{Q}_y, \mathcal{D}em_y)$ and Y, so the above inequalities ensure:

$$q(DP(\mathcal{C}_1, \mathcal{Q}_x, \mathcal{D}em_x) \uplus DP(\mathcal{C}_2, \mathcal{Q}_y, \mathcal{D}em_y)) \leq q(X \uplus Y) = q(\mathcal{A}_0^*)$$

Since $(\mathcal{C}_1, \mathcal{Q}_x, \mathcal{D}em_x)$ and $(\mathcal{C}_2, \mathcal{Q}_y, \mathcal{D}em_y)$ are compatible with $(\mathcal{C}_0, \mathcal{Q}_0, \mathcal{D}em_0)$, $DP(\mathcal{C}_1, \mathcal{Q}_x, \mathcal{D}em_x) \uplus DP(\mathcal{C}_2, \mathcal{Q}_y, \mathcal{D}em_y)$ will be considered by the algorithm when finding the minimum such union. Therefore $q(\mathcal{A}_1^* \uplus \mathcal{A}_2^*) \leq q(DP(\mathcal{C}_1, \mathcal{Q}_x, \mathcal{D}em_x) \uplus DP(\mathcal{C}_2, \mathcal{Q}_y, \mathcal{D}em_y)) \leq q(\mathcal{A}_0^*)$ and is thus minimum.

Since the branch decomposition gives a binary tree with $O(n)$ leaves, there are $O(n)$ clusters. For each cluster, $\mathcal{C}$, the boundary $\partial(\mathcal{C})$ has at most ω vertices. For each vertex $v \in \partial(\mathcal{C})$ there are $q(v)$ options for the value of $\mathcal{Q}$ and $dem(v)$ options for the value of $\mathcal{D}em$. There are thus at most $\hat{q}^\omega$ functions $\mathcal{Q}$ and $\widehat{dem}^{\,\omega}$ functions $\mathcal{D}em$ for each cluster boundary. This gives $O((\hat{q}\widehat{dem})^\omega n)$ indices of the DP table. Each entry requires comparing unions of entries from all compatible child configuration pairs. There are $O(\hat{q}^{2\omega}\widehat{dem}^{\,\omega})$ such pairs since knowing $\mathcal{D}em$ for one child and the parent completely determines $\mathcal{D}em$ for the other child. Checking for compatibility requires $O(\omega)$ time. This gives the algorithm a total runtime of $O(\omega\hat{q}^{3\omega}\widehat{dem}^{\,2\omega} n) = (\hat{q}\widehat{dem})^{O(\omega)} n$. □

Chapter 3

Capacitated Vehicle Routing Overview

3.1 Introduction

In this chapter, we give an in-depth introduction to the Capacitated Vehicle Routing problem. We first formally define the problem.

Capacitated Vehicle Routing

Input: edge-weighted graph $G = (V, E)$, depot vertex r, client set $S \subseteq V$, demand function $dem : S \to \mathbb{Z}^+$, and capacity $Q \in \mathbb{Z}^+$

Output: set of tours $\Pi = \{\pi_1, \pi_2, ..., \pi_k\}$ such that for all $v \in S$, there is some $j \in \{1, 2, ..., k\}$ such that π_j visits v, and for all i, π_i contains r and $\sum_{v \in \pi_i} dem(v) \leq Q$

Objective: minimize sum of tour lengths $\sum_{i=1}^{k} \ell(\pi_i)$

Note that here, $s \in \pi$ indicates that s is a client covered by tour π. There may be other clients $s' \notin \pi$ that tour π passes through without covering.

The client demand in Capacitated Vehicle Routing can be either *splittable* or *unsplittable*. In our problem formulation we require unsplittable demands because splittable-demand instances can be reduced to unsplittable-demand instances: a client s with splittable demand $dem(s)$ can be replaced by $dem(s)$ co-located clients with unsplittable unit demand. These new clients can be

added to the graph and connected to s with zero-length edges.

Two generalizations of Capacitated Vehicle Routing that we address are: Multiple-Depot Capacitated Vehicle Routing in which, rather than a single depot, a set $R \subset V$ of depots is specified and a solution consists of a set of paths whose endpoints are in R, and Capacitated Vehicle Routing with Penalties in which a solution can ignore clients by paying penalties for them.

A solution SOL to a vehicle-routing problem can be completely defined by the order in which clients and the depot are visited, since the vehicle can be assumed to use shortest paths between consecutive clients. For ease of notation, we say that $(u, v) \in SOL$ if u and v are clients visited consecutively by SOL, and similarly use $(u, r) \in SOL$ and $(r, v) \in SOL$ to denote clients visited immediately before and after visiting the depot.

3.2 Related Work

3.2.1 Traveling Salesperson Problem

One of the most well-studied vehicle-routing problems is Traveling Salesperson Problem (TSP), which is defined as follows:

Traveling Salesperson Problem (TSP)

Input: edge-weighted graph $G = (V, E)$

Output: a tour π that visits every vertex $v \in V$

Objective: minimize tour length $\ell(\pi)$

In fact, TSP is the special case of Capacitated Vehicle Routing where the client set S equals the entire vertex set V, and the vehicle capacity Q is large enough to accommodate all client demand.

TSP is a classic NP-hard problem and in general metrics TSP is APX-hard [86] and known to be inapproximable to within a factor of 123/122 [68]. For Euclidean metrics and planar graphs, TSP remains NP-hard, but is known to admit polynomial-time approximation schemes [6, 7, 54]. Solving TSP on trees is trivial, as the tour formed by traversing the tree in a depth-first search is an optimal TSP solution. For graphs with constant treewidth (or branchwidth), TSP is polynomial-time solvable using dynamic programming on the tree (or branch) decomposition [33].

For general metrics, a trivial 2-approximation can easily be found using depth-first search on the minimum spanning tree, whose cost is a lower bound for TSP. The best known approximation guarantee for general metrics is Christofides' 1.5 approximation [30].

Already for TSP, we see that the hardness of routing problems in general metrics does not capture the approximability of routing problems in metrics for which routing problems are relevant. We now consider the more general problem.

3.2.2 Capacitated Vehicle Routing

The Capacitated Vehicle Routing problem was first proposed in 1959 as the Truck Dispatching Problem [35]. Since then, a large majority of the work on this problem has focused on finding exact solutions and on designing heuristics. These two branches of literature are too vast to describe in detail here, so we give a brief summary.

Initially, much of the work on Capacitated Vehicle Routing focused on finding exact solutions. The two most common techniques for exact solutions have been branch-and-cut algorithms and algorithms that use a set-partitioning formulation of Capacitated Vehicle Routing. We refer the reader to surveys on these techniques [14, 15, 25]. We note that the standard benchmarks for computationally analyzing the performance of these techniques use instances with between 10 and 200 clients and Euclidean distances, and even for these, the state-of-the-art solutions for larger instances take about a day to find [15].

Recently, much of the research on Capacitated Vehicle Routing has shifted toward designing heuristics. We refer the reader to surveys on these techniques [75, 76]. Metaheuristics such as Evolutionary Algorithms and Tabu Search have further improved the state-of-the-art for heuristic approaches [25]. Noting the limitations of the classic benchmarks used for exact instances, a set of new benchmark instances was recently proposed [94]. These attempt to provide less artificial and more challenging instances on which to compare the performance of various algorithms. While the state-of-the-art heuristics can run on even the larger instances in a few hours, the quality of the returned solutions is unclear. Furthermore, these new instances still use Euclidean distances and the largest instance has only 1000 clients.

In this thesis, we seek approximation algorithms with *provable* performance guarantees. We now

outline the current state of such results. To better understand the complexity of Capacitated Vehicle Routing, recall the distinction between the two variants of the problem: *splittable* demands can be covered jointly by multiple tours and *unsplittable* demands must be covered by a single tour. As described in the previous section, splittable demands are equivalent to unsplittable unit demands.

As Capacitated Vehicle Routing generalizes TSP, it inherits TSP's hardness. In fact, Capacitated Vehicle Routing is substantially harder to solve than TSP. To illustrate this, consider the case of trees and arbitrary capacity Q. As noted above, TSP is trivial in trees, whereas splittable Capacitated Vehicle Routing is NP-hard in trees by a reduction from bin packing [74] and the unsplittable case is hard to approximate to better than a 1.5-factor in trees [52]. For unsplittable Capacitated Vehicle Routing in trees, Labbe et al. gave a 2-approximation [74]. For splittable Capacitated Vehicle Routing in trees, Hamaguchi and Katoh [59] define a simple lower bound on $cost(OPT)$ and give a 1.5-approximation, which Asano, Katoh, and Kawashima [10] improve to a $\frac{\sqrt{41}-1}{4} \approx 1.351$-approximation using the same lower bound.

They show that there are instances in which the ratio of $cost(OPT)$ to this lower bound is $4/3$, proving that a $4/3$-approximation would be best possible for this lower bound, and leave as an open question as to whether or not such an algorithm exists. We answer this question in the affirmative by providing such a $4/3$-approximation algorithm (see Chapter 7). It remains unknown as to whether this approximation ratio can be improved, using a different lower bound. Notably, in Chapter 8 we give a $(1, 1+\epsilon)$ *bicriteria* PTAS for Capacitated Vehicle Routing in trees, in which capacities are allowed to be exceeded by a $(1+\epsilon)$ factor, but it is not known if a PTAS exists without this allowed stretch in capacity.

For general metrics and arbitrary capacities, Haimovich and Rinnoy Kan [58] introduced one of the biggest advances for approximating splittable Capacitated Vehicle Routing. They use the following lower bound for Capacitated Vehicle Routing [58].

$$cost(OPT) \geq \frac{2}{Q} \sum_{s \in S} d(s, r) \tag{3.1}$$

The intuition of Lower Bound (3.1) is as follows. The length of any tour in a feasible solution must be at least twice the distance to the client on the tour that's farthest from the depot, as the vehicle has to get to that client and back. The tour length is therefore also at least twice the *average*

distance from the depot to a client on the tour. Noting that each tour visits at most Q clients and summing over all tours gives the lower bound.

Haimovich and Rinnoy Kan use this lower bound in a technique called *Iterated Tour Partitioning* [58], which starts with a TSP tour of the clients and partitions the tours into $|S|/Q$ segments each containing Q clients. Each tour of the solution consists of one of these segments and paths from the depot to each of the segment endpoints. A simple averaging argument shows that for at least one of the Q options for partitioning the tour in this way, the sum of the lengths of these paths from the depot is at most the value of Lower Bound 3.1. The length of an optimum TSP tour is also a clear lower bound on the cost of an optimum vehicle-routing solution, so the total cost an iterated tour partitioning solution is at most $(1 + \alpha\frac{Q-1}{Q})cost(OPT)$ where α is the approximation ratio for the TSP tour [58].

Lower Bound (3.1) was designed for the splittable-demand setting, but it can straightforwardly be extended to the unsplittable-demand setting:

$$cost(OPT) \geq \frac{2}{Q}\sum_{s\in S} dem(s) \cdot d(s, r) \tag{3.2}$$

A similar iterated tour partitioning approach can be applied to the unsplittable-demand setting, resulting in a $(2 + \alpha\frac{Q-2}{Q})$ approximation [5].

Using Christofides' 1.5-approximation for TSP [30], this strategy results in a $(2.5-\frac{1.5}{Q})$-approximation for splittable Capacitated Vehicle Routing and a $(3.5 - \frac{3}{Q})$-approximation for unsplittable Capacitated Vehicle Routing. There has been little improvement over Iterated Tour Partitioning in approximating Capacitated Vehicle Routing in general metrics for arbitrary capacities. Recently, Bomparde [23] showed how to improve the approximation ratio of each case by $\frac{1}{3Q^3}$, although even for moderate values of Q, this improvement is negligible.

A reasonable, and common, relaxation for Capacitated Vehicle Routing is to consider the case of bounded capacity. For $Q = 1$, the problem is trivial, and for $Q = 2$, the problem can be solved exactly in polynomial time using matchings [11]. But for any fixed $Q \geq 3$, Capacitated Vehicle Routing is APX-hard in general metrics [11]. On the other hand, for fixed Q, Capacitated Vehicle Routing is polynomial-time solvable in trees and in graphs of bounded treewidth.

One line of research has focused on Capacitated Vehicle Routing in Euclidean metrics. For

the Euclidean plane ($\mathbb{R}^2$), PTASs have been found for constant Q [58], $Q = O(\log n / \log\log n)$ [11], $Q \leq 2^{\log^\delta n}$ (where $\delta > 0$ depends on ϵ) [4], and $Q = \Omega(n)$ [11], as well as for the multiple-depot variant when both Q and the number of depots is constant [26]. For $\mathbb{R}^3$, a PTAS is known for $Q = O(\log n)$ [72], and for higher-dimensional Euclidean spaces $\mathbb{R}^d$, a PTAS is known for $Q = O(\log^{1/d} n)$ [71]. For arbitrary Q, Mathieu and Das gave a QPTAS for Capacitated Vehicle Routing in the plane [36]. No PTAS for arbitrary Q is known for Euclidean (or *any* non-trivial) metrics.

For Capacitated Vehicle Routing with fixed Q, the space between solving exactly in trees and being inapproximable in general metrics is poorly understood for non-Euclidean metrics. This thesis works toward filling this gap by answering the following question: are there approximation schemes or small constant-factor approximations for Capacitated Vehicle Routing in graphs that model transportation networks? Toward this goal, in Chapter 4 we present a QPTAS for planar and bounded-genus graphs, in Chapter 5 we improve this to a PTAS for planar graphs, and in Chapter 6 we present a PTAS for bounded-highway-dimension graphs.

3.3 Methods for Approximation Schemes for Capacitated Vehicle Routing with Fixed Capacities

In Chapters 4, 5, and 6, we present approximation schemes for Capacitated Vehicle Routing, for specific graph classes, when the vehicle capacity Q is a fixed constant. In this section we introduce some of the techniques that we develop for these results.

At a high level, the approximation schemes work by reducing the input instance to a simpler instance (namely one with bounded treewidth or branchwidth) that is easy to solve. If this reduction does not incur too much error, then an optimal solution found in the simpler instance can be mapped to a near-optimal solution in the input instance. We now describe this process in greater detail.

We present these methods for the *unsplittable-demand* setting, since this generalizes the splittable-demand setting.

3.3.1 Error Allowance

A standard technique in approximation schemes is to charge the introduced error to distances between vertices in an optimal solution. Applying this approach to vehicle-routing problems, for every $(u, v) \in OPT$ (recall this notation means that u and v are consecutive clients visited in an optimal solution) a near-optimal solution can *charge* $\epsilon \cdot d(u, v)$ error to this pair. The argument is that if a solution can be constructed so that all error is charged to some such consecutive-client pair and each such pair is charged at most once, the cost of such a solution would be at most $(1 + \epsilon)cost(OPT)$.

One of our major insights toward designing approximation schemes for Capacitated Vehicle Routing is that clients farther from the depot can afford to be charged a higher error than those close to the depot. That is, $(u, v) \in OPT$ can be charged error not only proportional to $d(u, v)$ but also proportional to $d(u, r)$ and $d(v, r)$.

In particular, we establish the following *error allowance* between consecutive clients u and v:

$$\textbf{Error Allowance} : \frac{\epsilon}{Q}(d(u, r) + d(v, r)) \tag{3.3}$$

Specifically, pairs of consecutive clients in a given solution can afford to be charged this amount. We use Lower Bound 3.2 to justify this error allowance.

3.3.2 Dynamic Programming

In Section 3.2 we stated that for fixed capacity Q, Capacitated Vehicle Routing in graphs of bounded treewidth can be solved exactly in polynomial time. In this section, we give such an algorithm. In particular, we present a dynamic programming algorithm that achieves the following.

Theorem 5. *For any $Q, \omega > 0$, there is an algorithm with $n^{O(\omega Q)}$ runtime that solves* Capacitated Vehicle Routing *on instances with capacity Q and treewidth ω.*

Though previous dynamic programming solutions with $n^{O(\omega Q^2)}$ were known, we believe we were the first to improve the runtime to $n^{O(\omega Q)}$. We note that an analogous dynamic program can be designed using branchwidth. See Section 1.3 for a definitions of treewidth and branchwidth.

Proof. Let G be a graph with treewidth ω and let D be a tree decomposition of G. Choose an arbitrary bag to be the root, and for each bag b of the decomposition let *cluster* $\mathcal{C}_b$ be the union

of the bags descending from b in the tree decomposition, minus the elements of b itself. The bag b forms a *boundary* between cluster $\mathcal{C}_b$ and $V \setminus \mathcal{C}_b$.

A configuration in the dynamic program describes how a solution *interacts with* a cluster: for each vertex v in the boundary b of the cluster, and for each possible capacity $q \leq Q$, the configuration specifies $I_{v,q}$ and $O_{v,q}$ which are respectively the number of tours that enter and exit $\mathcal{C}_b$ using vertex v and that have visited exactly q clients at the moment they reach v. We refer to this as the *flow* in and out of $\mathcal{C}_b$ at v. These values are sufficient to recover the intersection of the solution with the cluster: connecting each entering tour with an exiting one, at minimal cost, gives the optimal solution.

To simplify the dynamic program, we first convert D into a *nice* tree decomposition with $O(\omega n)$ bags (see Section 1.3). Recall that in a nice tree decomposition, each leaf bag contains a single vertex, and each non-leaf bag is one of three types: an *introduce* bag b has exactly one child b' and $b = b' \cup \{v\}$ for some vertex $v \notin b'$, a *forget* bag b has one child b', such that $b = b' \setminus \{v\}$ for some vertex $v \in b'$, and a *join* bag b has two children b_1 and b_2 such that $b = b_1 = b_2$.

One property of a nice tree decomposition is that each vertex is *forgotten* at most once. We assume the forget bag for the depot occurs at the root of the decomposition. If not, r can be added to every bag in the tree, and the leaves extended. This results in a nice tree decomposition with at most twice as many bags while adding at most one to the width.

A tour can be *uncrossed* to avoid crossing the same vertex in the same direction twice. As there are at most n different tours, $I_{v,q} \leq n$ and $O_{v,q} \leq n$, so there are $n^{O(\omega Q)}$ possible configurations per bag. Since there are $O(\omega n)$ bags in the nice tree decomposition, there are a total of $n^{O(\omega Q)}$ different configurations.

The algorithm runs bottom-up: given a configuration for each child node, it finds all possible *compatible* configurations for the parent node. We address the treatment of each type of bag as follows.

For a leaf bag, b, containing vertex u, $\mathcal{C}_b$ is empty, so trivially there are no tours entering or exiting $\mathcal{C}_b$. For configurations with $I_{u,q} = O_{u,q} = 0$ for all q, the algorithm stores the cost zero.

For a forget bag, the parent bag b is equal to its child bag b' minus some vertex u. For each child bag configuration, the algorithm considers all ways to transform it into a compatible parent bag configuration by rerouting u's flow and, if u is a client, covering its demand, $dem(u)$. For each resulting parent bag configuration, the dynamic program stores the cost only if it is less than the

current value stored for that configuration. After considering all child bag configurations and ways of forming a parent bag configuration, the values stored in the table are guaranteed to be optimal. Consider some configuration for the child bag. The algorithm transforms this child configuration into a parent configuration by first constructing several intermediate configurations. First, if u is a client, one tour is selected to visit it. There are three cases. If the tour crosses into $\mathcal{C}_{b'}$ after visiting u, the algorithm chooses a capacity $q \geq dem(u)$ and makes the following changes to the flow at u:

$$I_{u,q} \to I_{u,q} - 1, \quad I_{u,q-dem(u)} \to I_{u,q-dem(u)} + 1$$

There are at most Q such choices.

If the tour crosses out of $\mathcal{C}_{b'}$ after visiting u, the algorithm chooses a capacity $q \leq Q - dem(u)$ and makes the following changes to the flow at u:

$$O_{u,q} \to O_{u,q} - 1, \quad O_{u,q+dem(u)} \to O_{u,q+dem(u)} + 1$$

There are at most Q such choices.

Otherwise, the tour segment that visits u does not cross into $\mathcal{C}_{b'}$. The algorithm chooses $v_1, v_2 \in b$ and $q \leq Q - dem(u)$, makes the following changes to the flow at v_1 and v_2:

$$I_{v_1,q} \to I_{v_1,q} + 1, \quad O_{v_2,q+dem(u)} \to O_{v_2,q+dem(u)} + 1,$$

and adds $d(v_1, u) + d(u, v_2)$ to the intermediate configuration cost. There are $\omega^2 Q$ such choices. The algorithm then reroutes all flow through u to some vertex in the parent bag, b. The algorithm chooses, for each vertex v of b and each capacity, the number of the tours that enter (resp. exit) $\mathcal{C}_{b'}$ through u directly from (resp. to) v. Each such tour adds a cost of $d(u, v)$ to the intermediate configuration cost. There are $O(n^{2\omega Q})$ such choices. Thus, for each child configuration there are $O(\omega^2 Q n^{2\omega Q})$ choices, giving an $n^{O(\omega Q)}$ overall runtime for each forget bag.

For an introduce bag, the parent bag is equal to its child bag plus some vertex u. Since the child bag forms a boundary between the inside and outside of the cluster, no tour can cross directly into the cluster via u, as it must first cross some vertex of the child bag. Therefore the only compatible parent configurations are those that have no tours crossing at u. So for every parent configuration, if $I_{u,q} = O_{u,q} = 0$ for all q, the algorithm stores the cost of the corresponding child configuration,

namely the configuration that results by removing u. Otherwise the cost is ∞.

For a join bag, the parent bag has two child bags identical to itself. Lemma 4 presents an oracle that tells, in constant time, the minimal cost needed to form parent configuration (I^0, O^0) given child configurations (I^1, O^1) and (I^2, O^2), with an infinite cost if the configurations are not compatible. The algorithm tries all combinations of configurations: the complexity of this step is $n^{O(\omega Q)}$.

Since each vertex will appear exactly once in a forget bag, each client will be visited exactly once. The overall complexity is $n^{O(\omega Q)}$, as claimed. The algorithm considers all possible solutions and outputs the minimal one, so the resulting cost is optimal.

□

All that remains is to prove Lemma 4.

Lemma 4. *For each join bag b, it is possible to compute, in $O(n^{6\omega Q})$ time, a table $\mathcal{T}_b$ such that $\mathcal{T}_b[(I^0, O^0), (I^1, O^1), (I^2, O^2)]$ is the minimal cost to connect child configurations (I^1, O^1) and (I^2, O^2) to form parent configuration (I^0, O^0) of b.*

Proof. We design a dynamic program to compute this table. The base cases are when $I^0 = I^1 + I^2$. If $O^0 = O^1 + O^2$ the cost is 0, since the configurations are therefore compatible. Otherwise the cost is ∞, because it is not possible to balance incoming and outgoing flow.

For the recursion step, assume $I^0 \neq I^1 + I^2$. Pick the first pair (u, q) such that $I^1_{u,q} + I^2_{u,q} - I^0_{u,q} = x \neq 0$. If $x < 0$, the incoming flow at u with capacity q is bigger in b than in its child bags. Since this is not possible, the cost is ∞. Otherwise, some flow entering Cluster 1 comes from Cluster 2 (or vice versa). Suppose this flow exits Cluster 2 at vertex v: it means that

$$\mathcal{T}_b[(I^0, O^0), (I^1, O^1), (I^2, O^2)] = \mathcal{T}_b[(I^0, O^0), (\hat{I}^1, O^1), (I^2, \hat{O}^2)] + d(u, v)$$

where $\hat{I}^1 = \{I^1, I^1_{u,q} - 1\}$ and $\hat{O}^2 = \{O^2, O^2_{v,q} - 1\}$. By this equation, the algorithm connects one segment exiting Cluster 2 at v with capacity q to a segment entering Cluster 1 at u. The value of $\mathcal{T}_b[(I^0, O^0), (I^1, O^1), (I^2, O^2)]$ can therefore be computed in ω steps, by applying the above equality for each vertex v of the boundary and storing the minimum value. This computation requires $O(\omega Q)$ operations to find the pair (u, q), and then $O(\omega)$ operations to compute the value of the table. The recursion step therefore requires $O(\omega Q)$ time.

As there are $O(n^{6\omega Q})$ states for this DP, the overall complexity is therefore $O(\omega Q n^{6\omega Q}) =$

$O(n^{6\omega Q})$, concluding the proof. □

3.3.3 Metric Embeddings

In this section we present a general strategy for designing approximation schemes for Capacitated Vehicle Routing with fixed capacities. The strategy reduces the problem of designing such an approximation scheme to that of finding an appropriate *metric embedding* of the input graph. Specifically, we leverage the fact that Capacitated Vehicle Routing can be solved in polynomial time on graphs of bounded treewidth when the capacity Q is fixed, as shown in Theorem 5. If the input graph can be embedded into a bounded-treewidth graph without distorting distances too much, a near-optimal solution can then be found in polynomial time. We now formalize this reduction.

Recall, a *metric embedding* of a *guest* graph G into a *host* graph H, is a mapping $\phi(\cdot)$ of the vertices of G to the vertices of H. Metric embeddings are a useful tool for designing approximation algorithms. Suppose an embedding ϕ can be found such that the host graph H has simple structure and such that distances in H approximately preserve distances in G (i.e. $d_G(u, v)$ is similar to $d_H(\phi(u), \phi(v))$ for all $u, v \in V$). The problem instance *induced* in H can then be solved, and this solution can be *mapped back* to G. If the structure of H is simple enough, the solution can be found efficiently, and if distances in H are nearly preserved, the resulting solution in G will be near-optimal.

Related Metric Embedding Work

The general idea of this approach is not new. Bartal [17] presented a randomized algorithm that finds an embedding ϕ of a graph G into a tree T such that for all vertices u and v in G, $d_T(\phi(u), \phi(v))$ is *in expectation* at most $\log^c(n) \cdot d_G(u, v)$ for some constant c. This distortion factor was improved to $O(\log n)$ by Fakcharoenphol, Rao, and Talwar [43].

A natural question is whether this result can be improved for special graph classes. Chakrabarti et al. [27] proved that even for the easier problem of embedding unit-weighted planar graphs into graphs with $o(\sqrt{n})$ treewidth, a distortion factor of $o(\log n)$ cannot be achieved. If we consider *additive* rather than multiplicative distortion in distances, Fox-Epstein, Klein, and Schild [48] showed a planar graph G can be embedded into a graph H of bounded treewidth such that $d_H(\phi(u), \phi(v)) \leq d_G(u, v) + \epsilon \cdot \text{diam}(G)$.

When the input graph G has bounded *doubling dimension* θ and *aspect ratio* γ, Talwar [92] gave a randomized algorithm that finds an embedding of G into a graph H in which $E[d_H(\phi(u), \phi(v))] \leq$

$(1+\epsilon)d_G(u,v)$ for all u and v in G and such that the treewidth of H is bounded by some function $f(\gamma,\theta,\epsilon)$ that is polylogarithmic in γ. As we will discuss in Chapter 6, Feldman, Fung, Könemann, and Post [47] gave gave a similar embedding result for graphs of bounded highway dimension.

Metric Embedding Strategy for Capacitated Vehicle Routing

Of particular interest for designing Capacitated Vehicle Routing approximation schemes are metric embeddings that preserve distances between clients, up to the error allowance given by Error Allowance 3.3 and such that the host graph has bounded treewidth. By Theorem 5, Capacitated Vehicle Routing can then be solved exactly in the host graph in time exponential only in the treewidth and capacity. This solution, SOL_H in H can then be straightforwardly mapped back to a solution SOL_G in G: for all u,v in $S \cup \{r\}$, include $(u,v) \in SOL_G$ if and only if $(\phi(u),\phi(v)) \in SOL_H$. Because the embedding preserves distances up to the error allowance, the cost of SOL_G is near-optimal.

We formalize this approach in this section and prove Theorem 6, which can be applied to *any* graph class that has such an embedding. Given this theorem, all that remains to designing a PTAS for a given graph class is to find such an embedding. We will give examples of finding such embeddings for planar graphs (see Chapter 5) and graphs with bounded highway dimension (see Chapter 6).

Theorem 6. *Fix a family of graphs $\mathcal{G}$, function f, and $Q,\epsilon > 0$. If there exists a polynomial-time algorithm that, for any $G \in \mathcal{G}$ and $r \in V_G$, constructs a host graph H with treewidth at most $f(\epsilon,Q)$ and an embedding $\phi(\cdot)$ of G into H such that $\forall u,v \in V_G$,*

$$d_G(u,v) \leq d_H(\phi(u),\phi(v)) \leq d_G(u,v) + \frac{\epsilon}{Q}(d_G(u,r) + d_G(v,r))$$

then there exists an algorithm for Capacitated Vehicle Routing *with capacity Q on graphs in $\mathcal{G}$ that in polynomial time returns a solution whose cost is at most $1+\epsilon$ times optimal.*

Proof. Let $\mathcal{G}$ be some such graph class that has an embedding algorithm that satisfies the condition of the theorem statement. Let $Q > 0$ and $\epsilon > 0$ be fixed, let G be a graph $\mathcal{G}$, and let $r \in V_G$ be a specified vertex. Let f and ϕ be as given by the theorem.

Consider the instance of Capacitated Vehicle Routing on G with depot r and capacity Q. Use ϕ to embed G into host graph H, and consider the resulting *induced* instance of Capacitated

VEHICLE ROUTING in H, with capacity Q, depot $\phi(r)$ and client set $\{\phi(s)|s \in S\}$. Use the dynamic program of Theorem 5 to solve this instance in H.

Let SOL_H be the resulting optimum solution in H, and let SOL_G be the solution resulting from mapping SOL_H back to G.

Because the construction runtime is polynomial, $|V_H| \leq n^c$ for some constant c. Additionally, since the treewidth of H is at most $f(\epsilon, Q)$, the dynamic program given by Theorem 5 runs in time $n^{O(c \cdot f(\epsilon,Q)Q)}$. The overall runtime of this algorithm is therefore $n^{f'(\epsilon,Q)}$ where f' is some function that depends only on ϵ and Q.

Finally, we show that SOL_G is near-optimal. Let OPT be the optimal solution in G and let OPT_H be the corresponding induced solution in H. Since the dynamic program finds an optimal solution in H, we have $cost_H(SOL_H) \leq cost_H(OPT_H)$. Additionally, since distances in H are no shorter than distances in G, $cost_G(SOL_G) \leq cost_H(SOL_H)$. Putting these pieces together, we have,

$$\begin{aligned}
cost_G(SOL_G) &\leq cost_H(SOL_H) \leq cost_H(OPT_H) \\
&= \sum_{(u,v)\in OPT} d_H(\phi(u), \phi(v)) \\
&\leq \sum_{(u,v)\in OPT} d_G(u,v) + \frac{\epsilon}{Q}(d_G(u,r) + d_G(v,r)) \\
&= \sum_{(u,v)\in OPT} d_G(u,v) + 2\frac{\epsilon}{Q}\sum_{s\in S} d_G(s,r) \\
&\leq cost_G(OPT) + \frac{2\epsilon}{Q}\sum_{s\in S} dem(s) d_G(s,r) \\
&\leq (1+\epsilon) cost_G(OPT)
\end{aligned}$$

where the final inequality comes from Lower Bound 3.2 (see Section 3.2). □

Note that if f or the runtime for constructing H are allowed to depend polylogarithmically on n, the analogous result is a QPTAS for the given graph class. Additionally, we show the following analogous result for randomized embeddings.

Theorem 7. *Fixed a family of graphs $\mathcal{G}$, function f, and $Q, \epsilon > 0$. If there exists a randomized algorithm that, for any $G \in \mathcal{G}$ and $r \in V_G$, in polynomial time constructs a host graph H with*

treewidth at most $f(\epsilon, Q)$ *and an embedding* $\phi(\cdot)$ *of* G *into* H *such that* $\forall u, v \in V_G$,

$$d_G(u,v) \leq d_H(\phi(u), \phi(v)) \text{ and } E[d_H(\phi(u), \phi(v))] \leq d_G(u,v) + \frac{\epsilon}{Q}(d_G(u,r) + d_G(v,r))$$

then there exists an algorithm for Capacitated Vehicle Routing *with capacity* Q *on graphs in* $\mathcal{G}$ *that in polynomial time returns a solution whose expected cost is at most* $1+\epsilon$ *times optimal.*

Proof. The proof is nearly identical to that of Theorem 6, except for the following cost analysis. We use the same notation as before.

$$\begin{aligned}
E[cost_G(SOL_G)] &\leq E[cost_H(SOL_H)] \leq E[cost_H(OPT_H)] \\
&= E[\sum_{(u,v)\in OPT} d_H(\phi(u), \phi(v))] \\
&= \sum_{(u,v)\in OPT} E[d_H(\phi(u), \phi(v))] \\
&\leq \sum_{(u,v)\in OPT} d_G(u,v) + \frac{\epsilon}{Q}(d_G(u,r) + d_G(v,r)) \\
&= \sum_{(u,v)\in OPT} d_G(u,v) + 2\frac{\epsilon}{Q}\sum_{s\in S} d_G(s,r) \\
&\leq cost_G(OPT) + 2\frac{\epsilon}{Q}\sum_{s\in S} dem(s) d_G(s,r) \\
&\leq (1+\epsilon) cost_G(OPT)
\end{aligned}$$

□

3.4 Capacitated Vehicle Routing Extensions

In this section we consider generalizations of Capacitated Vehicle Routing and show how to extend our methods to address them.

3.4.1 Multi-Depot Capacitated Vehicle Routing

One natural generalization is the multiple depot setting. There are several reasonable ways to generalize single-depot Capacitated Vehicle Routing. We define Multi-Depot Capacitated Vehicle Routing as follows. Let $R \subseteq V$ be the set of depots denoted as $R = \{r_1, r_2, ..., r_{|R|}\}$.

Each vehicle must start at some source depot r_i and end at some destination depot r_j, though we do not require $i = j$. As before, each vehicle has a capacity Q and the vehicles must collectively cover the client-demand without exceeding their capacities. Routes are now depot-to-depot paths in G, and again the objective is to minimize the sum of the route lengths.

Notably, we assume that the vehicles are *not* assigned source and destination depots a priori and that the algorithm can determine how to preposition the vehicles. In fact, *any* way of formally defining a multiple-depot generalization requires assumptions about the fleet that were not required of the single-depot setting. For example, fleet size is irrelevant in single-depot Capacitated Vehicle Routing, as traversing a set of tours with a single vehicle costs the same as traversing the same set of tours with several vehicles. In multple-depot settings, however, fleet size and distribution become relevant.

Recall the distance of a vertex $v \in V$ to a subset $R \subseteq V$ is defined to be $d(v, R) = \min_{r \in R} d(v, r)$. We give the following lower bound for Multi-Depot Capacitated Vehicle Routing.

$$cost(OPT) \geq \frac{2}{Q} \sum_{s \in S} dem(s) d(s, R) \tag{3.4}$$

The justification is analogous to the single-depot setting. Consider any r_i to r_j vehicle path π in OPT, and let s^* be the client covered by π that is farthest from R. That is, $s^* = \arg\max_{s \in \pi} d(s, R)$. The length of π, denoted $\ell(\pi)$ is at least $d(s^*, r_i) + d(s^*, r_j)$ which itself is at least $2d(s^*, R)$. Noting that there are at most Q clients covered by π and that the maximum client-to-R distance is at least as much as the (weighted) average client-to-R distance, the lower bound follows from summing over all vehicle paths in OPT.

We use Lower Bound 3.4 to justify the following multi-depot error allowance for consecutive clients in a solution.

$$\textbf{Multi} - \textbf{Depot Error Allowance}: \frac{\epsilon}{2} d(u, v) + \frac{\epsilon}{2Q} (d(u, R) + d(v, R)) \tag{3.5}$$

Unlike in the single-depot setting, where, by the triangle inequality, $d(u, v) \leq d(u, r) + d(v, r)$, it is now possible that $d(u, v) \gg d(u, R) + d(v, R)$. We therefore must explicitly add a term to reflect the error that can be charged to $d(u, v)$.

Additionally, the dynamic program of Theorem 5 can be adapted to accommodate multiple

depots, though the resulting runtime will be exponential in $|R|$. Using such a DP and Error Allowance 3.5, we can then design an embedding strategy analogous to Theorem 6.

Theorem 8. *Fix a family of graphs $\mathcal{G}$, function f, and $Q, \rho, \epsilon > 0$. If there exists an algorithm that, for any $G \in \mathcal{G}$ and $R \subseteq V_G$ with $|R| = \rho$, in polynomial time constructs a host graph H with treewidth at most $f(\epsilon, Q, \rho)$ and an embedding $\phi(\cdot)$ of G into H such that $\forall u, v \in V_G$,*

$$d_G(u,v) \leq d_H(\phi(u), \phi(v)) \leq (1 + \frac{\epsilon}{2}) d_G(u,v) + \frac{\epsilon}{2Q}(d_G(u,R) + d_G(v,R))$$

then there exists an algorithm for Multi-Depot Capacitated Vehicle Routing *with capacity Q on graphs in $\mathcal{G}$ that in polynomial time returns a solution whose cost is at most $1+\epsilon$ times optimal.*

Proof. We use the same notation as in the proof of Theorem 6. The two proofs are nearly identical, except for the following error analysis.

$$\begin{aligned}
cost_G(SOL_G) &\leq cost_H(SOL_H) \leq cost_H(OPT_H) \\
&= \sum_{(u,v) \in OPT} d_H(\phi(u), \phi(v)) \\
&\leq \sum_{(u,v) \in OPT} (1 + \frac{\epsilon}{2}) d_G(u,v) + \frac{\epsilon}{2Q}(d_G(u,R) + d_G(v,R)) \\
&= (1 + \frac{\epsilon}{2}) \sum_{(u,v) \in OPT} d_G(u,v) + 2\frac{\epsilon}{2Q} \sum_{s \in S} d_G(s,R) \\
&\leq (1 + \frac{\epsilon}{2}) cost_G(OPT) + \frac{\epsilon}{Q} \sum_{s \in S} dem(s) d_G(s,R) \\
&\leq (1 + \epsilon) cost_G(OPT)
\end{aligned}$$

where the final inequality comes from Lower Bound 3.4. □

3.4.2 Applications to k-Center and k-Median

Another application of the embedding described in Theorem 8 is to give fixed-parameter approximations (FPAs) for k-Center and k-Median, i.e. algorithms with running time $f(k)n^{O(1)}$.

k-Center

We first present the FPA framework for k-Center. Recall that the k-Center problem is to find the set Z of k vertices (called *centers*) such that the maximum distance of a client to the nearest center is minimized. That is, find $\arg\min_{Z\subseteq V:|Z|\leq k}(\max_{s\in S} d(s,Z))$. Note that we address a generalization of the standard problem formulation as not every vertex needs to be covered.

Theorem 9. *Fix a family of graphs $\mathcal{G}$, function f_1, and $k,\epsilon>0$. If there exists an algorithm that, for any $\epsilon'>0$, $G\in\mathcal{G}$, and $Z\subseteq V_G$ with $|Z|=k$, in polynomial time constructs a host graph H with treewidth at most $f_1(\epsilon',k)$ and an embedding $\phi(\cdot)$ of G into H such that $\forall u,v\in V_G$,*

$$d_G(u,v)\leq d_H(\phi(u),\phi(v))\leq(1+\epsilon')d_G(u,v)+\epsilon'(d_G(u,Z)+d_G(v,Z))$$

then there exists a function f_2 and constant c such that there is an $f_2(\epsilon,k)n^c$ time algorithm that, given an instance of k-Center in G, finds a solution whose cost is at most $1+\epsilon$ times optimum.

The algorithm first computes an α-approximation Z in polynomial time, where α is $O(1)$ (see [60] or [53] for a 2-approximation). Applying the given embedding to G with the subset Z and $\epsilon'=\frac{\epsilon}{7\alpha}$ gives a host graph H. The algorithm then runs a DP that gives a $(1+\epsilon')$-approximation, SOL_H of the optimal solution in the host graph. Note that any vertices added in the construction of H are not required to be covered. Finally, the solution SOL_H is mapped back to a solution SOL_G in G.

We first prove that SOL_G is a $(1+\epsilon)$-approximation of the optimal solution G.

Lemma 5. *A $(1+\epsilon')$-approximation of k-Center in the host graph given by Theorem 8 is a $(1+\epsilon)$-approximation of k-Center in the original graph.*

Proof. Let OPT_G denote the optimal solution in G, and OPT_H denote the optimal solution in host graph H. For each vertex u, let c_u denote the closest center to u in OPT_G. We have the following:

$$\begin{aligned}
\mathrm{cost}_H(\mathrm{OPT}_G) &= \max_{u\in V_G} d_H(u,c_u)\\
&\leq \max_{u\in V_G}(1+\epsilon')d_G(u,c_u) \;+\; \epsilon'(d_G(u,Z)+d_G(c_u,Z))\\
&\leq (1+\epsilon')\max_{u\in V_G} d_G(u,c_u) \;+\; 2\epsilon'\max_{u\in V_G} d_G(u,Z)\\
&\leq (1+\epsilon')cost_G(OPT_G) \;+\; 2\epsilon'\alpha cost_G(OPT_G)\\
&\leq (1+3\alpha\epsilon')cost_G(OPT_G)
\end{aligned}$$

We want to show that $cost_G(SOL_G) \leq (1+\epsilon)cost_G(OPT_G)$.

$$\begin{aligned} cost_G(SOL_G) &= cost_G(SOL_H) \\ &\leq cost_H(SOL_H) \\ &\leq (1+\epsilon')cost_H(OPT_H) \\ &\leq (1+\epsilon')cost_H(OPT_G) \end{aligned}$$

Substituting the above inequality gives,

$$\begin{aligned} cost_G(SOL_G) &\leq (1+\epsilon')(1+3\alpha\epsilon')cost_G(OPT_G) \\ &\leq (1+7\alpha\epsilon')cost_G(OPT_G) \\ &= (1+\epsilon)cost_G(OPT_G) \end{aligned}$$

That is, since the optimal solution in H is an approximate solution in G, an approximate solution in H is also an approximate solution in G. □

The runtime of the DP given by Schild, Fox-Epstein and Klein [48] for a graph with treewidth ω is $O(n(\log n)^{\omega})$, which is $O(n^{O(1)}\omega^{2\omega})$ (following Lemma 1 in Katsikarelis et al. [70]). Since the treewidth of H is $f_1(\epsilon', k)$, this gives a runtime of $f_1(\epsilon', k)^{2f_1(\epsilon',k)}n^{O(1)}$, proving the fixed-parameter approximation of Theorem 8.

k-MEDIAN

We now present an analogous FPA framework for k-MEDIAN. The k-MEDIAN problem is to find the set Z of k vertices (called *centers*) that minimizes the *sum* of distances between clients and their nearest center. That is, find $\arg\min_{Z\subseteq V:|Z|\leq k}(\sum_{s\in S} d(s,Z))$. Again, we address a generalization of the standard problem as not every vertex needs to be covered.

Theorem 10. *Fix a family of graphs $\mathcal{G}$, function f_1, and $k, \epsilon > 0$. If there exists an algorithm that, for any $\epsilon' > 0$, $G \in \mathcal{G}$, and $Z \subseteq V_G$ with $|Z| = k$, in polynomial time constructs a host graph H with treewidth at most $f_1(\epsilon', k)$ and an embedding $\phi(\cdot)$ of G into H such that $\forall u, v \in V_G$,*

$$d_G(u,v) \leq d_H(\phi(u),\phi(v)) \leq (1+\epsilon')d_G(u,v) + \epsilon'(d_G(u,Z) + d_G(v,Z))$$

*then there exists a function f_2 and constant c, such that there is a $f_2(\epsilon,k)n^c$ time algorithm that, given an instance of k-*MEDIAN *in G, finds a solution whose cost is at most $1+\epsilon$ times optimum.*

The outline is the same as for k-CENTER. The algorithm first computes an α-approximation Z for some $\alpha = O(1)$ (see [91]). Then the embedding algorithm is applied using the set Z and $\epsilon' = \frac{\epsilon}{9\alpha}$. Finally the algorithm computes an approximate solution in the host graph. The dynamic program for k-CENTER can be adapted to solve k-MEDIAN with the same complexity (again, any vertices added during the host graph construction do not contribute to the cost). The following lemma is straightforward:

Lemma 6. *A $(1+\epsilon')$-approximation of k-median in the host graph given by Theorem 10 is a $(1+\epsilon)$-approximation of k-median in the original graph.*

The proof is almost identical to that of Lemma 5, replacing the max by a sum. The difference in ϵ' values comes from the following:

$$\text{cost}_H(\text{OPT}_G) \leq \sum_{u \in V_G} (1+\epsilon') d_G(u, c_u) + \epsilon'(d_G(u, Z) + d_G(c_u, Z))$$

The triangle inequality gives,

$$\begin{aligned}
\text{cost}_H(\text{OPT}_G) &\leq \sum_{u \in V_G} (1+\epsilon') d_G(u, c_u) + \epsilon'(d_G(u, Z) + d_G(c_u, u) + d_G(u, Z)) \\
&\leq (1+2\epsilon') \sum_{u \in V_G} d_G(u, c_u) + 2\epsilon' \sum_{u \in V_G} d_G(u, Z) \\
&\leq (1+2\epsilon') cost_G(OPT_G) + 2\epsilon' \alpha cost_G(OPT_G) \\
&\leq (1+4\alpha\epsilon') cost_G(OPT_G)
\end{aligned}$$

3.4.3 CAPACITATED VEHICLE ROUTING WITH PENALTIES

Another natural generalization to consider is CAPACITATED VEHICLE ROUTING WITH PENALTIES, in which a penalty value $\hat{p}(s)$ is specified for each client s, and a solution is allowed to avoid covering a subset of clients in exchange for paying the associated penalties for these uncovered clients. The objective then is to minimize the sum of the tour lengths and the penalties for skipped clients. One application of this generalization is the *vehicle-routing problem with private fleet and common carrier* (VRPPC), "where customers may either be served by using owned vehicles with traditional

routes or be assigned to a common carrier, which serves them directly at a prefixed cost" [62, p. 13].

Adapting the dynamic program of Theorem 3.3.2 for solving Capacitated Vehicle Routing instances in graphs of bounded treewidth to account for penalties is fairly straightforward. Specifically, the only required change is that when processing the forget bag of a client s, the algorithm also considers the cost of paying $\hat{p}(s)$ instead of covering s.

Using such a dynamic program, the metric embedding strategy introduced in Section 3.3.3 can be extended to accommodate such penalties. Specifically, we can derive the following theorem analogous to Theorem 6

Theorem 11. *Fix a family of graphs $\mathcal{G}$, function f, and $Q, \epsilon > 0$. If there exists an algorithm that, for any $G \in \mathcal{G}$, $r \in V_G$, $S \subseteq V_G$ and function $\hat{p} : S \to \mathbb{Z}$, in polynomial time constructs a host graph H with treewidth at most $f(\epsilon, Q)$ and an embedding $\phi(\cdot)$ of G into H such that $\forall u, v \in V_G$,*

$$d_G(u,v) \leq d_H(\phi(u), \phi(v)) \leq d_G(u,v) + \frac{\epsilon}{Q}(d_G(u,r) + d_G(v,r))$$

then there exists an algorithm for Capacitated Vehicle Routing with Penalties *with capacity Q on graphs in $\mathcal{G}$ that in polynomial time returns a solution whose cost is at most $1+\epsilon$ times optimal.*

Note that the same embedding can be used for Capacitated Vehicle Routing and Capacitated Vehicle Routing with Penalties, where for all $s \in S$, the penalty of $\phi(s)$ in H is $\hat{p}(s)$.

Given the dynamic program described above, all that remains to proving Theorem 11 is to show that the optimal solution to Capacitated Vehicle Routing with Penalties in the host graph has a cost at most $(1+\epsilon)\text{cost}(\text{OPT}_G)$.

Proof. The clients can be divided into two sets U and W, where OPT visits every vertex in U and pays the penalty for the ones in W. We use $(u, v) \in OPT$ to denote that u and v are consecutive *covered* clients in some tour of OPT. Applying Lower Bound 3.2 to the client subset U gives

$$\text{cost}_G(\text{OPT}) \geq \frac{2}{Q} \sum_{v \in U} dem(v) \cdot d_G(v, r) + \sum_{v \in W} \hat{p}(v)$$

Using this lower bound and the notation from Theorem 6 gives,

$$
\begin{aligned}
cost_G(SOL_G) &\leq cost_H(SOL_H) \leq cost_H(OPT_H) \\
&= \sum_{(u,v)\in OPT} d_H(\phi(u),\phi(v)) + \sum_{v\in W} \hat{p}(v) \\
&\leq \sum_{(u,v)\in OPT} d_G(u,v) + \frac{\epsilon}{Q}(d_G(u,r) + d_G(v,r)) + \sum_{v\in W} \hat{p}(v) \\
&= \sum_{(u,v)\in OPT} d_G(u,v) + 2\frac{\epsilon}{Q}\sum_{s\in U} d_G(s,r) + \sum_{v\in W} \hat{p}(v) \\
&\leq \sum_{(u,v)\in OPT} d_G(u,v) + \sum_{v\in W} \hat{p}(v) + \frac{2\epsilon}{Q}\sum_{s\in U} dem(s) d_G(s,r) \\
&\leq (1+\epsilon) cost_G(OPT)
\end{aligned}
$$

□

Chapter 4

A QPTAS for Capacitated Vehicle Routing in Planar Graphs

4.1 Introduction

In this chapter, we present a quasi-polynomial-time approximation scheme (QPTAS) for Capacitated Vehicle Routing with fixed capacities in planar graphs. Specifically, we prove the following.

Theorem 12. *For any $\epsilon > 0$ and any $Q > 0$, there is a quasi-polynomial-time algorithm that, given an instance of* Capacitated Vehicle Routing *in which the capacity is Q and the graph is planar, finds a solution whose cost is at most $1 + \epsilon$ times optimum.*

Our quasi-polynomial-time approximation scheme can also be extended to handle Multi-Depot Capacitated Vehicle Routing as well as Capacitated Vehicle Routing with Penalties (see Section 3.4). The algorithm can also be generalized to graphs of bounded genus and polylogarithmic capacity Q. (Q is considered constant in Theorem 12.) We state our result in its full generality.

Theorem 13. *For any $\epsilon > 0$, any $g \geq 0$, any $R \geq 0$ and any $c \geq 0$, there is a quasi-polynomial-time algorithm that, given an instance of* Multi-Depot Capacitated Vehicle Routing with Penalties *in which the capacity Q is $O(\log^c n)$ and the graph has genus g with R designated depots, finds a solution whose cost is at most $1 + \epsilon$ times optimum.*

Our algorithm uses a recursive decomposition (see Section 1.3) of the graph using shortest-path separators. That is, there is a recursive clustering in which the vertices on the boundary of each cluster lie on a small number of shortest paths. This general idea has been used in several previous approximation schemes for planar and bounded-genus graphs [7, 31, 40, 54]. The closest previous use was in addressing the k-Center problem [40], and we use a lemma from their paper stating that such a recursive decomposition exists that has logarithmic depth.[1]

We introduce several new ideas in order to apply the recursive decomposition to vehicle routing. Before finding the recursive decomposition, the algorithm must *prune* the graph to eliminate vertices that are too far from the depot to participate in an optimal solution. In our algorithm, the shortest paths bounding each cluster are *subpaths of shortest paths from the depot.* This ensures that if vertices u and v are on a bounding shortest path P and u is farther along P than v, then u must also be farther from the depot than v. This allows us to make use of Lower Bound (3.2), which depends on distances of clients from the depot.

The algorithm designates some of the vertices of the cluster boundaries to be *portals*, and requires (some) paths of the solution to enter and leave clusters only at these portals. This in itself is not novel; portals have been used before. We introduce two new ideas. The first is how we select portals. In previous use of portals in approximation schemes for planar graphs, portals are selected uniformly along a path. In our algorithm, it is essential that the portals be selected in a nonuniform fashion: the farther from the depot, the greater the spacing between portals. This introduces more error in solution segments farther from the depot but such error can be afforded due to Lower Bound (3.2).

Second, requiring the *entire* solution to pass in and out of clusters via portals would introduce too much error. Instead, we only require a tour to use portals when entering or exiting the clusters that contain clients covered by that tour. The tour is not required to use portals to pass through other clusters. This way, we can bound the solution error in terms of clients and their distance from the depot. (This error also depends on the depth of nesting in the recursive decomposition, since in the process of picking up a client the tour might pass through many clusters at different levels of nesting, but the depth of nesting is merely logarithmic.)

After showing that an approximately optimal solution can be assumed to cross clusters in this way, we reduce the problem to one we can address using dynamic programming. The dynamic

[1] Such a decomposition was earlier described by Thorup [93] in the context of approximate distance oracles. However, in addressing the generalization to multiple depots we use a generalization of the decomposition that follows from slight modification of the proof in [40].

program introduced in Section 3.3.2 does not quite suffice here, because the solution can sometimes pass through the boundary without using portals. The dynamic program must also make use of the metric on the entire graph as well.[2]

4.2 Preliminaries

Recall that for any edge set $F \subseteq E$ the *boundary* of F, denoted $\partial(F)$, is the set of vertices that are incident both to edges in F and edges in $E \setminus F$.

We assume that the length of any edge (u, v) is $d_G(u, v)$; if not, it would not be used and can be removed from the graph.

For a graph G and vertex r, an *r-rooted shortest path tree* T is an r-rooted tree in which for all v in V the r-to-v path in T is a shortest path in G. For any vertex u on a shortest r-to-v path, we call the u-to-v subpath a *from-r shortest subpath*. Such a subpath must also be a shortest u-to-v path.

A *triangulated* planar graph is one in which every face has exactly three incident edges. A planar graph can easily be triangulated in linear time by adding edges that recursively subdivide faces with more than three incident edges. Each new edge (u, v) is given cost $d(u, v)$. Triangulating a planar graph requires adding $O(n)$ edges.

A *recursive clustering* of a graph G is a recursive partition (see Section 1.3) of the vertices V_G. Specificially, it is a rooted binary tree T in which

- each node is labeled with a *cluster* $\mathcal{C} \subseteq V_G$,
- If x is a child of y in T, then the cluster associated with x is a subset of the cluster associated with y, and
- there is a mapping $\phi(\cdot)$ that maps each vertex v of G to a leaf cluster $\phi(v)$ that contains v.

Each vertex v is considered to be *assigned to* each cluster containing $\phi(v)$.

A vertex v of a cluster $\mathcal{C}$ is a *boundary vertex* of the cluster if v also belongs to a cluster $\mathcal{C}'$ that neither contains nor is contained in $\mathcal{C}$. An edge (u, v) of G is a *boundary edge* of a cluster $\mathcal{C}$ if u is in $\mathcal{C}$ and v is not. The *depth* of a recursive clustering is the depth of the rooted binary tree.

[2] A similar technique was used in [40].

4.3 Decomposing the Graph

In this section, we describe the recursive clustering performed by our algorithm.

4.3.1 Graph Pruning

As a preprocessing step, the algorithm prunes from the graph those vertices that are not clients and are very far from the depot. Specifically, the algorithm deletes each vertex that does not lie on some u-to-v shortest path with $u, v \in S \cup \{r\}$. Since the optimal solution is composed of such paths, pruning does not increase $cost(OPT)$.

Lemma 7. *For all vertices w that remain after the preprocessing step, $d(r, w) \leq cost(OPT)$.*

Proof. Since w survived the pruning step, it must lie on some u-to-v shortest path with $u, v \in S \cup \{r\}$. Without loss of generality, assume that the optimal solution visits u before visiting v. Therefore $cost(OPT) \geq d(r, u) + d(u, v) \geq d(r, u) + d(u, w) \geq d(r, w)$ where the final inequality comes from the triangle inequality. □

4.3.2 Cluster Decomposition

The following lemma is a slight generalization of a lemma in [40].

Lemma 8 (Generalization of Lemma 3.1 in [40])**.** *Let T be a tree of degree at most three. Let Y be a subset of the vertices. There is a depth-$O(\log |Y|)$ recursive clustering of T such that*

- *there are no boundary vertices,*
- *each leaf cluster is assigned only one vertex of Y, and*
- *each cluster $\mathcal{C}$ has at most four boundary edges.*

Let G be a planar embedded graph and T be a spanning tree of G. Let G' be obtained from G by adding artificial edges to triangulate G and G^* be the planar dual of G'. Finally, let T^* be the set of edges of G^* corresponding to edges of G' that are not in T. Then T^* is a spanning tree of G^* (namely, the *interdigitating tree*). Each edge of T^* corresponds to a nontree edge in G, which in turn defines a cycle consisting of that nontree edge together with a path through T. Since G' is triangulated, G^* has degree at most three. Map each vertex of G' to one of the faces to which it

is incident. Let Y be the set of faces to which elements of S are mapped. By choosing T to be an r-rooted shortest-path tree of G and applying Lemma 8, we obtain the following generalization of a corollary of [40].

Lemma 9 (Generalization of Corollary 3.1 of [40])**.** *Let G be a planar embedded triangulated graph with edge costs, let S be a subset of the vertices, and let r be a vertex of G. There is a depth-$O(\log |S|)$ recursive clustering of G with the following properties:*

- *there are no boundary edges,*
- *for each cluster, there are at most eight from-r shortest paths such that the boundary vertices of the cluster are the vertices that lie on these paths, and*
- *at most three vertices of S are assigned to each leaf cluster.*

The algorithm computes a recursive clustering as described in Lemma 9.

4.3.3 Choosing Portals

The algorithm designates *portals* along each cluster boundary in a two-step process. First it designates some of the vertices as *spacers*. Second, for each cluster it designates a subset of the cluster's boundary vertices to be portals, including all designated spacers as well as some additional vertices.

The choice of spacers depends on a parameter δ. We will choose

$$\delta = \frac{\epsilon}{(Q+4)c \log |S|} \tag{4.1}$$

where c is an absolute constant to be determined in the proof of Theorem 14.

We first describe spacer selection. Let T be the from-r shortest-path tree. Let $\hat{v}$ be the vertex farthest from r that remains after the pruning step and let $\delta > 0$. Designate r to be a spacer, and consider the unpruned vertices in increasing order of distance from r. For each vertex v in turn, let x_v be the *closest* ancestor of v that has already been designated as a spacer. Designate v as a spacer if $d(v, x_v) > \delta \cdot \max(d(r, x_v), \frac{1}{|S|} d(r, \hat{v}))$.

Lemma 10. *Each from-r shortest path has at most $2 + \log_{1+\delta} \frac{|S|}{\delta}$ spacers.*

Proof. Let P be a from-r shortest path, and let $r{=}x_0, x_1, \ldots, x_l$ be the spacers on P in increasing order of distance from r. We bound l as follows. For each $i > 0$, $d(x_{i-1}, x_i) > \delta\ d(r, x_{i-1})$, so

$$d(r, x_i) = d(r, x_{i-1}) + d(x_{i-1}, x_i) > (1+\delta) d(r, x_{i-1})$$

By induction,

$$d(r, x_l) > (1+\delta)^{l-1} d(r, x_1)$$

Since $d(r, x_1) > \delta \frac{1}{|S|} d(r, \hat{v})$, we infer $d(r, x_l) > (1+\delta)^{l-1} \frac{\delta}{|S|} d(r, \hat{v})$, which implies

$$(1+\delta)^{l-1} < \frac{|S|}{\delta} \frac{d(r, x_l)}{d(r, \hat{v})} \leq \frac{|S|}{\delta}$$

which shows

$$l - 1 < \log_{1+\delta} \frac{|S|}{\delta}.$$

□

The algorithm then designates some of each cluster's boundary vertices as *portals*. For each cluster $\mathcal{C}$, the boundary vertices of $\mathcal{C}$ lie on $O(1)$ from-r shortest subpaths. For each such from-r subpath P, designate as portals the first vertex of P and all the vertices of P that are spacers.

By Lemma 10, each path P contributes $O(\frac{1}{\delta} \log(\frac{1}{\delta}|S|))$ portals. Using the definition of δ in Equation 4.1, we obtain,

Lemma 11. *For each cluster boundary, the above algorithm designates $O(Q\frac{1}{\epsilon} \log^2 |S|)$ portals.*

We also show a bound on the distance along the boundary to the nearest portal.

Lemma 12. *For every cluster $\mathcal{C}$ and boundary vertex $v \in \partial(\mathcal{C})$, there is a portal p of $\mathcal{C}$ and a p-to-v path of cost at most $\delta(d(r, v) + \frac{cost(OPT)}{|S|})$.*

Proof. By Lemma 9, v must lie on some from-r shortest subpath P that bounds cluster $\mathcal{C}$. Let x be v's closest ancestor in T that is a spacer. By the algorithm for designating spacers and Lemma 7,

$$d(x, v) \leq \delta\ \max(d(r, x), \frac{1}{|S|} d(r, \hat{v})) \leq \delta\ \max(d(r, v), \frac{cost(OPT)}{|S|})$$

If x belongs to P then x is a boundary vertex of $\mathcal{C}$ and hence a portal of $\mathcal{C}$. In this case, we take $p = x$. If not, then let p be the first vertex of P and note that $d(p, v) \leq d(x, v)$. □

Finally, we note that for any cluster $\mathcal{C}$ that contains the depot r, it can be assumed that r is on the boundary of $\mathcal{C}$ and thus a portal of $\mathcal{C}$. In particular, for the topmost cluster (that contains all of V_G) r is the only portal.

4.4 Structure Theorem

Consider a solution to Capacitated Vehicle Routing. It consists of a set of tours. Associated with each tour π is a set of vertices in $V_\pi \cap S$ that the tour visits. For a cluster $\mathcal{C}$, a tour *fragment* with respect to $\mathcal{C}$ is a maximal subpath of a tour all of whose vertices are in $\mathcal{C}$. Every tour starts and ends at the depot r. By construction, if r is in $\mathcal{C}$ then it is a boundary vertex of $\mathcal{C}$. Therefore, by maximality, each endpoint of a tour fragment with respect to $\mathcal{C}$ is a boundary vertex of $\mathcal{C}$.

A tour fragment is *visiting* if it visits clients and *passing* if it does not. The endpoints of a visiting fragment are called *gates*.

Lemma 13. *Any solution to* Capacitated Vehicle Routing *crosses through* $O(\log |S|)$ *gates between two consecutive visits to clients.*

Proof. Consider a solution to Capacitated Vehicle Routing. Let u and v be two consecutive clients visited by tour π of the solution, and let P be the subpath of π between visiting u and visiting v. Suppose $\mathcal{C}$ is a cluster in which P' is a subpath of π that is a visiting tour fragment with respect to $\mathcal{C}$, such that P' has an endpoint x that's an internal vertex of P. Since P' is a visiting fragment, it visits some client, and since no clients are visited on P between u and v, the other endpoint y of P' must not also be an internal vertex of P. Thus either u or v must be assigned to the cluster $\mathcal{C}$.

If u is assigned to $\mathcal{C}$ then all edges on the u-to-x subpath of P belong to $\mathcal{C}$, and the edge of P after x does not. If v is assigned to $\mathcal{C}$ then all edges on the x-to-v subpath of P belong to $\mathcal{C}$, and the edge of P before x does not.

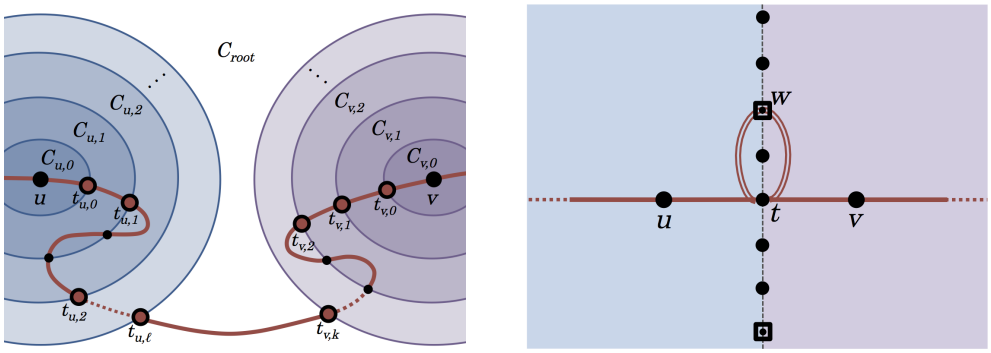
Now, let $\mathcal{C}_{u,0} \subset \mathcal{C}_{u,1} \subset ... \subset \mathcal{C}_{u,l}$ be the clusters that contain u but not v, and let $\mathcal{C}_{v,0} \subset \mathcal{C}_{v,1} \subset ... \subset \mathcal{C}_{v,k}$ be the clusters that contain v but not u.

For $i = 0, 1, \ldots, l$, let $t_{u,i}$ be the *last* vertex x of P such that the u-to-x subpath of P consists of edges of cluster $\mathcal{C}_{u,i}$. For $i = 0, 1, \ldots, k$, let $t_{v,i}$ be the *first* vertex x of P such that the x-to-v subpath of P consists of edges of cluster $\mathcal{C}_{v,i}$.

Since the clusters are non-crossing, P visits $t_{u,0}, t_{u,1}, \ldots, t_{u,l}, t_{v,k}, t_{v,k-1}, ..., t_{v,0}$ in order (See

Figure 4.1a). The argument above shows that every gate that π crosses between u and v is one of $t_{u,0}, \ldots, t_{v,0}$.

By Lemma 9, the depth of the decomposition is $O(\log |S|)$, so $k + l$ is $O(\log |S|)$. □



(a) Gates (b) Detour

Figure 4.1: (a) Gates are depicted by large, hollow circles at crossings. Non-gate crossings enclose passing segments (b) Boundary portals are denoted by squares. The double-line paths depict a detour.

A solution is said to be *portal-respecting* if for every tour in the solution, every gate in the tour is a portal.

Theorem 14. *There exists a portal-respecting solution with cost at most* $(1 + \epsilon)cost(OPT)$.

Proof. Let OPT be an optimal solution to Capacitated Vehicle Routing. To prove the theorem, we show how to modify OPT to construct a portal-respecting solution $\mathcal{SOL}$ by introducing *detours* at every gate, and bound the cost incurred by these detours.

Consider a tour fragment P with respect to some cluster $\mathcal{C}$, and let t be an endpoint of P. The corresponding detour is the subpath of a from-r shortest subpath from t to the nearest portal, and back. (See Figure 4.1b.) Splicing such a detour into the solution at each gate ensures that the solution is still feasible and is portal-respecting. It remains to show that the cost of these detours is small.

Let u and v be two consecutive clients visited by OPT. By Lemma 13 there is some constant c such that there are at most $c \log |S|$ gates between u and v where detours will be added. We use this constant c in the definition of δ given in Equation 4.1. By Lemma 12 the distance from any crossing t to the nearest portal is at most $\delta(d(r,t) + \frac{cost(OPT)}{|S|})$, so the cost of each detour is at most $2\delta(d(r,t) + \frac{cost(OPT)}{|S|})$.

Since OPT is optimal, the path that it takes from u to v must be a shortest path. By the triangle inequality, $d(r,t) \leq d(r,v) + d(v,t) \leq d(r,v) + d(v,u)$. Therefore, the cost of each detour is at most $2\delta(d(u,v) + d(r,v) + \frac{cost(OPT)}{|S|})$.

Summing over all gates gives $2\delta c \log|S|\big(d(u,v)+d(r,v)+\frac{cost(OPT)}{|S|}\big)$, and summing over all pairs of consecutive clients and using Lower Bound 3.2,

$$\begin{aligned}
&\sum_{(u,v)\in OPT} 2\delta c \log|S|\big(d(u,v) + d(r,v) + \frac{cost(OPT)}{|S|}\big) \\
&\leq 2\delta c \log|S|(cost(OPT) + \frac{Q}{2}cost(OPT) + cost(OPT)) \\
&= \delta c \log|S|(Q+4)cost(OPT)
\end{aligned}$$

The definition of δ given in Equation 4.1 ensures that the total detour cost is at most $\epsilon cost(OPT)$. □

The notion of *portal-respecting* can be applied not just to a solution but to a partial solution as well. This is used in the next section.

4.5 Dynamic Program

We present two dynamic programs: the first is slow but is the basis of the second, which gives a QPTAS. Both have the same configurations but enumerate the transitions in different ways.

4.5.1 Configurations

The goal of the dynamic program is to compute the minimal cost of a *portal-respecting* solution. Recall that by construction if the depot r is contained in a cluster, it is a portal of its boundary. Additionally, since each client is assigned to exactly one leaf cluster, we avoid overcounting any client's demand. For a client s, let $dem_{\mathcal{C},s}$ be the demand of s inside the cluster $\mathcal{C}$. We let $dem_{\mathcal{C},s} = dem(s)$ if s is assigned to $\mathcal{C}$, otherwise $dem_{\mathcal{C},s} = 0$.

A *configuration* $\mathcal{X}$ for cluster $\mathcal{C}$ specifies, for each pair of portals (i,o) of the cluster and for each $q \in \{1,...,Q\}$, a number $\mathcal{X}_{i,o,q}$ of segments that enter $\mathcal{C}$ at portal i, cover exactly q units of client demand in $\mathcal{C}$, and leave $\mathcal{C}$ at portal o. A configuration for cluster $\mathcal{C}$ is *admissible* if $\Sigma_{i,o,q} q\mathcal{X}_{i,o,q} = \Sigma_{s\in\mathcal{C}} dem_{\mathcal{C},s}$.

A *partial solution* $SOL_{\mathcal{C}}$ for cluster $\mathcal{C}$ is a set of tour fragments with respect to $\mathcal{C}$. We say that a partial solution $SOL_{\mathcal{C}}$ for cluster $\mathcal{C}$ *induces the configuration* $\mathcal{X}$ if every *visiting* fragment of $SOL_{\mathcal{C}}$ corresponds to a fragment described in $\mathcal{X}$ (recall that a visiting fragment is one that visits at least one client).

Our dynamic program computes, for each cluster $\mathcal{C}$ and admissible configuration $\mathcal{X}$, the weight $DP(\mathcal{C}, \mathcal{X})$ of the minimum-weight, portal-respecting partial solution that induces the configuration $\mathcal{X}$.

4.5.2 Compatibility

To compute the minimal cost of a partial solution for a cluster $\mathcal{C}_0$ that induces the configuration $\mathcal{X}^0$, the algorithm determines possible configurations for the two child clusters of $\mathcal{C}_0$, namely $\mathcal{C}_1$ and $\mathcal{C}_2$. Let $\mathcal{C}_0$ be a cluster, with two child clusters $\mathcal{C}_1$ and $\mathcal{C}_2$, and let $\mathcal{X}^0$, $\mathcal{X}^1$, and $\mathcal{X}^2$ be three configurations such that $\mathcal{X}^i$ is admissible for $\mathcal{C}_i$. We say that $\mathcal{X}^1$ and $\mathcal{X}^2$ are *compatible* with $\mathcal{X}^0$ if each segment (i, o, q) described in $\mathcal{X}^0$ can be mapped to a tuple of segments of $\mathcal{X}^1$ and $\mathcal{X}^2$, $\big((b_1, e_1, j_1, q_1), \cdots, (b_K, e_K, j_K, q_K)\big)$, where $j_i \in \{1, 2\}$, b_i and e_i are portals of $\mathcal{C}_{j_i}$, and such that $0 < q_i < Q$ and $\Sigma_{i=1}^{K} q_i = q$. We use $(\mathcal{X}^1, \mathcal{X}^2) \sim \mathcal{X}^0$ to denote that $\mathcal{X}^1$ and $\mathcal{X}^2$ are compatible with $\mathcal{X}^0$. To ensure that partial solutions are portal-respecting, configurations only need to describe the segments that visit clients. The other segments need not cross boundaries at portals, so we assume them to be shortest paths in the original graph.

Conversely, every segment of $\mathcal{X}^1$ and $\mathcal{X}^2$ must correspond to exactly one segment of $\mathcal{X}^0$. Intuitively, this means that every segment in $\mathcal{X}^0$ can be broken into subsegments that respect the portals of $\mathcal{C}_1$ and $\mathcal{C}_2$. The path from i to o can be divided into a shortest path from i to b_1, segments from b_i to e_i visiting q_i clients in the cluster $\mathcal{C}_{j_i}$, and shortest paths from e_i to b_{i+1} and from e_K to o visiting 0 clients.

We define the *price*, ζ, of the compatible configurations to be the cost of connecting the segments: $\zeta(\mathcal{C}_0, \mathcal{X}^0, \mathcal{X}^1, \mathcal{X}^2) = d(i, b_1) + \Sigma_{i=1}^{K-1} d(e_i, b_{i+1}) + d(e_K, o)$. Therefore, the minimal cost of a partial solution for a cluster $\mathcal{C}_0$ that induces the configuration $\mathcal{X}^0$ is $DP(\mathcal{C}_0, \mathcal{X}^0) = \min_{(\mathcal{X}^1, \mathcal{X}^2) \sim \mathcal{X}^0} DP(\mathcal{C}_1, \mathcal{X}^1) + DP(\mathcal{C}_2, \mathcal{X}^2) + \zeta(\mathcal{C}_0, \mathcal{X}^0, \mathcal{X}^1, \mathcal{X}^2)$.

The algorithm enumerates all possible configurations $\mathcal{X}^1$ and $\mathcal{X}^2$ that are compatible with $\mathcal{X}^0$. As in the above definitions, the algorithm breaks every segment of $\mathcal{X}^0$ into subsegments, each visiting some clients in $\mathcal{C}_1$ or in $\mathcal{C}_2$. It then adds each subsegment to the corresponding subconfiguration:

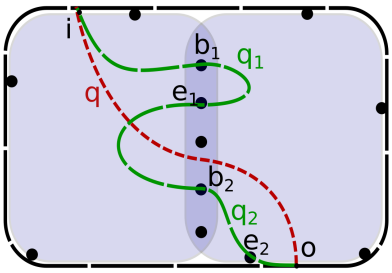


Figure 4.2: The segment (i, o, q) of the parent cluster is shown in red (short-dashed line). The green path (long-dashed line) shows one way to map this segment onto the child clusters: segment (b_1, e_1) covers q_1 client demand and (b_2, e_2) covers q_2 client demand with $q_1 + q_2 = q$. Segments (i, b_1), (e_1, b_2), and (e_2, o) are passing segments and visit no clients. (Figure by David Saulpic).

the subsegment is added to $\mathcal{X}^{j_\imath}$. As $q_\imath > 0$ and $\Sigma_{i=\imath}^{K} q_\imath = q \leq Q$, $K \leq Q$. The algorithm enumerates all possibilities and calculates the value of $DP(\mathcal{C}_0, \mathcal{X}^0)$.

4.5.3 Base Cases

Each base case is a cluster $\mathcal{C}$ in which there are at most three clients. Since the configuration $\mathcal{X}$ is admissible, there are at most three tuples i, o, q such that $\mathcal{X}_{i,o,q} \neq 0$. There are therefore a constant number of ways of assigning each client of the cluster to a segment of the configuration. The algorithm tries them all, in constant time, and keeps the smallest cost.

4.5.4 Final Output and Correctness

Since the topmost cluster has r as its only portal, all configurations will consist of r-to-r segments visiting at most Q clients, and collectively visiting all clients of S. These are exactly the feasible Capacitated Vehicle Routing solutions. The algorithm returns the minimum over all admissible configurations of the top-level cluster, which is the cost of the optimal portal-respecting solution.

Theorem 15. *The dynamic programming algorithm described above outputs the minimal weight of a portal-respecting solution to* Capacitated Vehicle Routing.

Proof. For a cluster $\mathcal{C}$ and an admissible configuration $\mathcal{X}$, we prove that $DP(\mathcal{C}, \mathcal{X})$ is the minimum weight of a partial solution that induces the configuration and is portal-respecting. We proceed by induction.

Base cases. The algorithm tries all the possible ways to produce a portal-respecting partial solution that induces the configuration and keeps the minimal one.

Induction step. Let $\mathcal{C}_0$ be a cluster and $\mathcal{X}^0$ an admissible configuration for $\mathcal{C}_0$, and suppose that the DP is true for $\mathcal{C}_0$'s children clusters, $\mathcal{C}_1$ and $\mathcal{C}_2$. Let $\mathcal{SOL}_{C_0}$ be the optimal partial solution that induces the configuration $\mathcal{X}^0$ and $cost(\mathcal{SOL}_{C_0})$ be its cost. We prove that $DP(\mathcal{C}_0, \mathcal{X}^0) = cost(\mathcal{SOL}_{C_0})$. By definition, all configurations enumerated by the algorithm correspond to portal-respecting solutions that induce $\mathcal{X}^0$, so $DP(\mathcal{C}_0, \mathcal{X}^0) \geq cost(\mathcal{SOL}_{C_0})$. For the other inequality, we break each segment induced by the solution $\mathcal{SOL}_{C_0}$ into at most Q (see Section 4.5.2) segments of $\mathcal{C}_1$ and $\mathcal{C}_2$ to obtain configurations $\mathcal{X}^1$ and $\mathcal{X}^2$. These configurations are compatible by construction, so they appear in the enumeration of the algorithm. Thus, by induction, $DP(\mathcal{C}_0, \mathcal{X}^0) \leq cost(\mathcal{SOL}_{C_0})$.

Overall Solution. Let $\mathcal{C}_E$ be the topmost cluster, namely the cluster that contains all edges. Recall that r is the only portal on $\partial(\mathcal{C}_E)$ so all admissible configurations are composed of r-to-r segments each visiting at most Q clients and collectively visiting all clients in S. By induction, $DP(\mathcal{C}_E, \mathcal{X})$ is the minimum weight of a portal-respecting partial solution for $\mathcal{C}_E$ that induces $\mathcal{X}$, which itself is a portal-respeciting solution to Capacitated Vehicle Routing. The optimal cost is therefore $\min_{\mathcal{X}} DP(\mathcal{C}_E, \mathcal{X})$ over all admissible $\mathcal{X}$. □

4.5.5 Complexity Analysis

The complexity is determined by the number of compatible configurations. For each cluster and each admissible configuration for this cluster, the algorithm computes

$$|\{\text{ways of breaking a segment}\}|^{|\{\text{segments}\}|}$$

compatible subconfigurations. We first count the number of admissible configurations for a cluster.

Lemma 14. *The number of admissible configurations $\mathcal{X}$ for a given cluster $\mathcal{C}$ is $|S|^{O(Q^3\epsilon^{-2}\log^2|S|)}$.*

Proof. An admissible configuration is a vector $\mathcal{X}$ indexed by two portals (i, o) and client demand q. $\mathcal{X}_{i,o,q}$ is the number of segments going from i to o covering q units of client demand. As the configuration is admissible and demand is unsplittable, this number cannot exceed the total number of clients, so $\mathcal{X}_{i,o,q} \leq |S|$. Moreover, $q \leq Q$ and because by Lemma 11 there are $O(Q\epsilon^{-1}\log|S|)$ portals for $\mathcal{C}$ the number of choices of (i, o) is less than $O(Q^2\epsilon^{-2}\log^2|S|)$. The number of admissible configurations $\mathcal{X}$ for a given cluster is thus $|S|^{O(Q^3\epsilon^{-2}\log^2|S|)}$. □

We now count the number of compatible subconfigurations for a given configuration $\mathcal{X}$.

Lemma 15. *There are $O\big((2Q^3\epsilon^{-2}\log^2|S|)^{Q|S|}\big)$ compatible subconfiguration pairs for a given configuration.*

Proof. Using the same argument as in Lemma 14 and the fact that $q > 0$, we can bound the number of segments of a configuration: $\Sigma_{i,o,q}\mathcal{X}_{i,o,q} \leq |S|$. To break a segment, the algorithm chooses at most Q subsegments, each one consisting of a boolean, a pair of portals and a number of client-demand units covered by the segment (see Section 4.5.2). As there are $O(Q\epsilon^{-1}\log|S|)$ child-cluster portals by Lemma 11 and the capacity is bounded by Q, there are $O((2\cdot Q\cdot Q^2\epsilon^{-2}\log^2|S|)^Q)$ ways of breaking a single segment. As there are fewer than $|S|$ segments, we have $O((2Q^3\epsilon^{-2}\log^2|S|)^{Q|S|})$ compatible subconfiguration pairs per configuration. □

Lemma 16. *The overall complexity of the dynamic program is $|S|^{O(Q^3\epsilon^{-2}\log^2|S|)}\big(\frac{Q\log|S|}{\epsilon}\big)^{O(Q|S|)}$.*

Proof. As stated above, the dynamic program computes for each cluster and each admissible configuration, all compatible subconfigurations. As the decomposition is a binary tree with at most one leaf per client, the number of clusters is $O(S)$. Combining this with Lemma 14 and Lemma 15, the total complexity is therefore $|S|\cdot|S|^{O(Q^3\epsilon^{-2}\log^2|S|)}\big(2Q^3\epsilon^{-2}\log^2|S|\big)^{Q|S|} = |S|^{O(Q^3\epsilon^{-2}\log^2|S|)}\big(\frac{Q\log|S|}{\epsilon}\big)^{O(Q|S|)}$.

□

4.5.6 QPTAS

The slowest operation in the DP is generating all compatible subconfigurations for a given cluster. But this is very redundant: two different segments can be broken into the same subsegments. We present a preprocessing step that computes, for every cluster and every triplet of configurations for parent and children clusters, the minimal price $\zeta(\mathcal{C}_0, \mathcal{X}^0, \mathcal{X}^1, \mathcal{X}^2)$ of *passing* segments needed to make the configurations compatible. Recall that a passing segment is one that visits no clients and that they are necessary to connect the *visiting* segments of the subconfigurations.

Preprocessing Algorithm

We describe a recursive algorithm. The inputs are a cluster $\mathcal{C}_0$, a configuration $\mathcal{X}^0$ of $\mathcal{C}_0$ and two configurations $\mathcal{X}^1$ and $\mathcal{X}^2$ of the children of $\mathcal{C}_0$. To determine the price for connecting these three configurations, the algorithm considers the first segment appearing in $\mathcal{X}^0$ and breaks it into subsegments belonging to the children. It then deletes this segment from $\mathcal{X}^0$ (giving a new configuration

$\hat{\mathcal{X}}^0$) and deletes the subsegments from the children configurations, to obtain the subconfigurations $\hat{\mathcal{X}}^1$ and $\hat{\mathcal{X}}^2$. It tries all possibilities for breaking this segment and returns the cheapest one :

$$\zeta[\mathcal{C}_0, \mathcal{X}^0, \mathcal{X}^1, \mathcal{X}^2] = \min\left(\zeta(\mathcal{C}_0, \hat{\mathcal{X}}^0, \hat{\mathcal{X}}^1, \hat{\mathcal{X}}^2) + d(i, b_1) + \Sigma d(e_\imath, b_{\imath+1}) + d(b_K, o)\right)$$

The base case is when there are no segments in $\mathcal{X}^0$ (i.e. $\mathcal{X}^0_{i,o,q} = 0,\ \forall i, o, q$). Here the algorithm just checks that there are no remaining segments in $\mathcal{X}^1$ and $\mathcal{X}^2$. It returns 0 if there are none and ∞ otherwise. This algorithm can be memoized because the number of segments in the first configuration is strictly decreasing.

Pseudo-code of the algorithm using this preprocessing step is given in Algorithm 1

Algorithm 1 Preprocessing Step

function PRICE($\mathcal{C}_0, \mathcal{X}^0, \mathcal{X}^1, \mathcal{X}^2$)
  if $\mathcal{X}^0 = \mathbf{0}$ **then** ▷ Base case
    if $\mathcal{X}^1 = \mathbf{0}$ and $\mathcal{X}^2 = \mathbf{0}$ **then**
      Return 0
    else
      Return ∞
  else
    Let (i, o, q_0) be a segment such that $\mathcal{X}^0_{i,o,q_0} > 0$
    Let $\hat{\mathcal{X}}^0 \leftarrow \{\mathcal{X}^0, \mathcal{X}^0_{i,o,q_0} - 1\}$
    $\mathcal{C}_1, \mathcal{C}_2 \leftarrow$ subclusters of $\mathcal{C}_0$
    for all pairs of portals $(b_1, e_1) \cdots (b_K, e_K)$ of $\partial(\mathcal{C}_1) \cup \partial(\mathcal{C}_2)$, $q_1, \cdots, q_K \in \mathbf{N}^*$ such that $\Sigma q_i = q_0$ valid assignment of the subsegments to $\mathcal{C}_1$ or $\mathcal{C}_2$ **do**
      Let $\hat{\mathcal{X}}^1 \leftarrow \{\mathcal{X}^1, \mathcal{X}^1_{b_i,e_i,q_i} - 1 \ \ \forall (b_i, e_i, q_i)$ assigned to $\mathcal{C}_1\}$
      Let $\hat{\mathcal{X}}^2 \leftarrow \{\mathcal{X}^2, \mathcal{X}^2_{b_i,e_i,q_i} - 1 \ \ \forall (b_i, e_i, q_i)$ assigned to $\mathcal{C}_2\}$
      $\zeta(\mathcal{C}_0, \mathcal{X}^0, \mathcal{X}^1, \mathcal{X}^2) = \min \zeta(\mathcal{C}_0, \mathcal{X}^0, \mathcal{X}^1, \mathcal{X}^2)$,
   PRICE($\mathcal{C}_0, \hat{\mathcal{X}}^0, \hat{\mathcal{X}}^1, \hat{\mathcal{X}}^2$) + $d(i, b_1) + \Sigma d(e_i, b_{i+1}) + d(b_K, o)$)

Algorithm 2 Dynamic Program

function DP($\mathcal{C}_0, \mathcal{X}^0$)
  if $\mathcal{C}_0$ is a leaf **then**
    enumerate the base cases
  $\mathcal{C}_1, \mathcal{C}_2 \leftarrow$ subclusters of $\mathcal{C}_0$
  for all $\mathcal{X}^1$ admissible for $\mathcal{C}_1$ **do**
    for all $\mathcal{X}^2$ admissible for $\mathcal{C}_2$ **do**
      $DP(\mathcal{C}_0, \mathcal{X}^0) \leftarrow \min(DP(\mathcal{C}_0, \mathcal{X}^0),$ PRICE$(\mathcal{C}_0, \mathcal{X}^0, \mathcal{X}^1, \mathcal{X}^2))$ + DP$(\mathcal{C}_1, \mathcal{X}^1)$ + DP$(\mathcal{C}_2, \mathcal{X}^2)$
  Return $DP(\mathcal{C}_0, \mathcal{X}^0)$

Correctness

We prove the correctness of this algorithm by induction. The base case is an empty configuration for the parent cluster. The two subconfigurations are therefore compatible with it only if they are also empty. If $\mathcal{X}^0$ is compatible with $\mathcal{X}^1$ and $\mathcal{X}^2$, then the chosen segment of $\mathcal{X}^0$ corresponds by definition to at most Q subsegments in $\mathcal{X}^1$ and $\mathcal{X}^2$. The algorithm enumerates all possible ways to break the segment, so at least one will result in a compatible configuration. If $\mathcal{X}^0$ is not compatible with $\mathcal{X}^1$ and $\mathcal{X}^2$, the first segment cannot be broken in such a way that results in three compatible configurations, so the algorithm avoids false positives.

Complexity

The complexity of this preprocessing step is

$$O\big(|\{\text{clusters}\}| \cdot |\{\text{configurations}\}|^3 \cdot |\{\text{ways of breaking a segment}\}|\big).$$

We showed in Lemma 14 that there is $|S|^{O(Q^3\epsilon^{-2}\log^2|S|)}$ configurations and we showed in the proof of Lemma 15 that there are $O\big((2Q^3\epsilon^2\log^2|S|)^Q\big)$ ways of breaking a segment, so the preprocessing can be achieved in $O(|S| \cdot |S|^{O\left(Q^3\epsilon^{-2}\log^2|S|\right)}(2Q^3\epsilon^{-2}\log^2|S|)^Q) = O(2^{P(\log|S|,Q,\epsilon^{-1})})$ where P is some polynomial.

We can use this preprocessing step to improve the complexity of the main DP. Instead of breaking segments we try all compatible configurations for the children clusters, of which there are at most $O(|\{\text{configuration}\}|^2)$. The complexity of this improved DP is therefore $O(|\{\text{clusters}\}| \times |\{\text{configuration}\}|^3)$ and is dominated by the preprocessing step. Combining this analysis with Theorem 14 gives a QPTAS for planar graphs, proving Theorem 12.

4.6 Generalizations

4.6.1 Multi-Depot Capacitated Vehicle Routing

Our techniques can be extended to address Multi-Depot Capacitated Vehicle Routing, assuming a constant number of depots. Recall, in this variation, each tour can start and end at different depots. Let R denote the set of depots, and for $v \in S$ let r_v denote the closest depot to v

(note that the tour that visits v does not necessarily visit r_v). The generalization relies on two key observations.

First, the recursive clustering can be slightly modified in the following way. Let a *from-R shortest path* be a from-r shortest path for some $r \in R$.

Lemma 17. *Let G be a planar embedded graph with edge costs, and let R and S be subsets of the vertices. There is a depth-$O(\log |S|)$ recursive clustering of G with the following properties:*

- *there are no boundary edges,*
- *for each cluster, there are $O(|R|)$ from-R shortest subpaths such that the boundary vertices of the cluster are the vertices that lie on these paths, and*
- *at most three vertices of S are assigned to each leaf cluster.*

Proof. We sketch the proof. It follows the proof of Lemma 9, which in turn follows that of [40]. Consider the R-rooted shortest-path forest F. Each tree is rooted at some $r \in R$, and consists of those vertices v for which $r_v = r$. Construct a tree T by arbitrarily linking the trees of F, and then use the construction of Lemma 9. This gives a decomposition such that each cluster is bounded by four fundamental cycles of the tree T. Since a fundamental cycle in T consists of at most $2|R|$ from-R shortest paths, this concludes the proof. □

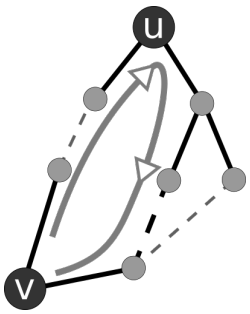


Figure 4.3: Here, u and v are depots. The forest is in black, the plain lines are the shortest-path trees from u and v and the dashed one is the connecting edge. The dashed, gray lines are edges not in T. The arrow is the boundary of a cluster: there is one fundamental cycle in T, and thus two from-R shortest paths. (Figure by David Saulpic).

Portals in this decomposition are designated the same way as in Section 4.3. The cost of a detour becomes $\delta(d(v, r_v) + \frac{cost(OPT)}{|S|})$ (using the notation of Section 4.3). Using Lower Bound 3.4, we can prove a multiple-depot version of the structure theorem, analogous to Theorem 14. Assuming that $|R|$ is constant, we can adapt the dynamic program for this decomposition to get a QPTAS.

4.6.2 Bounded Genus

To extend our algorithm to handle the case when G is embedded on a surface of genus $g > 0$, we adapt a technique previously used by Eistenstat et al. [40]. Let T be any spanning tree of G, in our case the shortest-path tree rooted at the depot. The algorithm selects [41] $2g$ edges not in T such that cutting the surface along the corresponding cycles (each edge forms a cycle with the corresponding simple path in T) yields a surface (with boundary) of genus 0, and a graph embedded on this surface. The vertices of these cycles lie on shortest from-r paths where r is the depot. The algorithm then cuts along these cycles, duplicating the vertices (an edge belonging to such a cycle ends up on one side or the other). The resulting graph is planar.

Next the algorithm forms the cluster decomposition for that graph. Finally, the algorithm merges duplicate vertices together. Merging the duplicates can result in those merged vertices being boundary vertices of the clusters, but fortunately all of those merged vertices lie on at most $2g$ from-r shortest paths, so designation of portals can continue as in Section 4.3.3 and in total the number of portals per cluster will be $O(Qg\epsilon^{-1}\log^2|S|)$.

4.6.3 Handling Penalties

Recall in Capacitated Vehicle Routing with Penalties, the algorithm is allowed to exclude some clients in exchange for paying the penalties associated with those clients.

The dynamic program of Section 4.5 computes, for each cluster and each admissible configuration of that cluster, the minimum cost partial solution that induces that configuration. To handle penalties, we change the definition of *solution*, of *admissible* and of *cost*. A solution is now allowed to not visit all the clients, an admissible configuration is allowed to not visit all the clients, and the cost includes the penalties of unvisited clients. The base cases change slightly to accommodate these changes, but otherwise the dynamic program is mostly unchanged.

One other change to the algorithm is needed. As described in Section 4.3.1, the algorithm needs to prune the graph, removing vertices that are too far to be included. To handle penalties, we need a more complicated pruning step. The algorithm computes an upper bound $\widehat{cost}$ on the value of the optimum that is at most Q times the value of the optimum, and then prunes away every vertex whose distance from the depot is greater than $\frac{\widehat{cost}}{2}$. This ensures that, for any cluster found in the pruned graph, the value $dist(r, \hat{v})$ is at most $\frac{Q}{2}cost(\mathrm{OPT})$, and our analysis can be adapted to show

that the number of portals is not too large.

Lemma 18. *There exists a polynomial-time algorithm that computes a Q-approximation for* Capacitated Vehicle Routing with Penalties.

Proof. Consider an instance of Capacitated Vehicle Routing with Penalties with capacity Q and a modified version in which the capacity is 1. Solving the modified instance is easy: for each client, include a depot-to-client round-trip if the cost of this trip is no more than the client's penalty. It remains to show that the optimum value for the modified instance is at most Q times the optimum value for the original instance.

Consider an optimal solution for the original instance. For each tour π in that solution, the cost of π is at least the cost of a round-trip from the depot to the farthest client visited by π. Replace π by a collection of tours, one visiting each of the clients visited by π. Each of these tours is a round trip, and there are at most Q of them, so their total cost is at most Q times the cost of π. The set of unvisited clients has not changed so the sum of their penalties remains unchanged. Thus the total value of the solution thus obtained is at most Q times the optimum value for the original instance. □

Chapter 5

Capacitated Vehicle Routing

PTAS for Planar Graphs

5.1 Introduction

In this chapter we improve upon our PTAS result of Chapter 4 and present a *polynomial-time* approximation scheme (PTAS) for Capacitated Vehicle Routing, on instances in which the input graph is planar and the capacity Q is bounded. In particular, we present the first known PTAS for Capacitated Vehicle Routing on planar graphs.

Theorem 16. *For any $\epsilon > 0$ and capacity Q, there is a polynomial-time algorithm for* Capacitated Vehicle Routing *on planar graphs that returns a solution whose cost is at most $1+\epsilon$ times optimal.*

Much of the analysis for proving Theorem 16 was already completed in Chapter 3. Recall, an *embedding* of a guest graph G into a host graph H is a mapping $\phi : V_G \longrightarrow V_H$ (see Section 3.3.3). By Theorem 6 and the dynamic program given by Theorem 5, all that remains to designing a PTAS for Capacitated Vehicle Routing with capacity Q on planar graphs is to find an algorithm that embeds planar graphs into host graphs with bounded treewidth such that distances are sufficiently preserved.

To obtain our PTAS result, we first design a *randomized* embedding that satisfies the conditions of Theorem 7 (see Section 3.3.3). Specifically, we embed the input graph into a bounded-treewidth host graph while preserving, *in expectation*, the client-to-client distances up to a small additive error

proportional to client distances to the depot, namely Error Allowance 3.3 (see Section 3.3.1). This gives a solution whose *expected* cost is at most $(1+\epsilon)cost(OPT)$. We then show in Section 5.3 how this result can be derandomized.

Our new embedding result builds on that of Fox-Epstein, Klein, and Schild [48]. Recall, their result is an embedding for planar graphs into bounded treewidth graphs such that distances are preserved up to a small additive error, of ϵ times the diameter of the input graph. The challenge in directly applying their embedding result to Capacitated Vehicle Routing is that this error exceeds the error allowance for those clients close to the depot. Instead, we divide the graph into annuli (*bands*) defined by distance ranges from the depot and apply the embedding result to each induced subgraph independently, with an increasingly large error tolerance for the annuli farthest from the depot. In this way, each client *can* afford an error proportional to the diameter of the *subgraph* it belongs to.

How can these subgraph embeddings be combined into a global embedding with the desired properties? In particular, clients that are close to each other in the input graph may be separated into different annuli. How can we ensure that the embedding approximately preserves these distances while still achieving bounded treewidth?

We use a technique common in metric embedding algorithms. We show that by *randomizing* the choice of where to define the annuli boundaries, and connecting all vertices of all subgraph embeddings to a new, global depot, client distances are approximately preserved (to within their error allowance) *in expectation* by the overall embedding, without substantially increasing the treewidth. To do so, we must ensure that the annuli are *wide* enough that the probability of nearby clients being separated (and thus generating large error) is small. Simultaneously, the annuli must be *narrow* enough that, within a given annulus, the clients closest to the depot can afford an error proportional to the distance of the farthest clients from the depot.

As described in Chapter 3, given the embedding result, a dynamic-programming algorithm can then be used to find an optimal solution to Capacitated Vehicle Routing in the bounded-treewidth host graph, and the solution can be lifted to obtain a solution in the input graph that in expectation is near-optimal.

Finally we describe how this result can be derandomized by trying all possible (relevant) choices for defining annuli and noting that for *some* such choice, the resulting solution cost must be near-optimal.

5.2 Embedding

In this section, we construct a randomized embedding that satisfies the conditions of Theorem 7 for the class of planar graphs. Specifically, we show the following.

Theorem 17. *There is a constant c and a randomized polynomial-time algorithm that, given a planar graph G with specified root vertex r and given $0 < \epsilon < 1$, computes a graph H with treewidth at most $(\frac{1}{\epsilon})^{c\epsilon^{-1}}$ and an embedding ϕ of G into H, such that, for every pair of vertices u, v of G, $d_G(u,v) \leq d_H(\phi(u), \phi(v))$ with probability 1, and*

$$E[d_H(\phi(u), \phi(v))] \leq d_G(u,v) + 3\epsilon[d_G(u,r) + d_G(v,r)] \tag{5.1}$$

Using $\hat{\epsilon} = \frac{\epsilon}{3Q}$ in such an embedding satisfies the conditions for Theorem 7, which proves the following corollary.

We assume all nonzero distances in G are at least one. If not, the graph can be rescaled. We also assume values of ϵ are less than one. If not, any $\epsilon \geq 1$ can be replaced with a number ϵ' slightly less than one. This only helps the approximation guarantee and does not significantly increase runtime. Of course for very large values of ϵ, an efficient constant-factor approximation can be used instead (see Section 3.2).

Corollary 2. *For any $\epsilon > 0$ and capacity Q, there is a randomized polynomial-time algorithm for* Capacitated Vehicle Routing *on planar graphs that returns a solution whose expected cost is at most $1 + \epsilon$ times optimal.*

To prove Theorem 17, we use the following result from [48] as a black box:

Lemma 19 ([48])**.** *There is a number c and a polynomial-time algorithm that, given a planar graph G with specified root vertex r and diameter D, computes a graph H of treewidth at most $(\frac{1}{\epsilon})^c$ and an embedding ϕ of G into H such that, for all vertices u and v,*

$$d_G(u,v) \leq d_H(\phi(u), \phi(v)) \leq d_G(u,v) + \epsilon D$$

Recall, the *diameter* of a graph G is the maximum distance $d_G(u,v)$ over all choices of u and v.

Our embedding partitions vertices of G into *bands* of vertices defined by distances from r. Choose $x \in [0,1]$ uniformly at random. Let B_0 be the set of vertices v such that $d_G(r,v) < \frac{1}{\epsilon}^{(x)\frac{1}{\epsilon}}$, and

for $i \in \{1, 2, 3, ...\}$ let B_i be the set of vertices v such that $\frac{1}{\epsilon}^{(i+x-1)\frac{1}{\epsilon}} \leq d_G(r, v) < \frac{1}{\epsilon}^{(i+x)\frac{1}{\epsilon}}$ (see Figure 5.1).

Let G_i be the subgraph induced by B_i, together with all u-to-v and v-to-r shortest paths for all $u, v \in B_i$. Note that although the B_i partition V, the G_i do not partition G. Note also that the diameter of G_i is at most $4\frac{1}{\epsilon}^{(i+x)\frac{1}{\epsilon}}$. The factor of 4 addresses the fact that for $u, v \in B_i$, the u-to-v shortest path is included in G_i and may contain a vertex $w \notin B_i$. But for any such w, it must be that $d_G(r, w) \leq 2\frac{1}{\epsilon}^{(i+x)\frac{1}{\epsilon}}$.

For each G_i, let H_i be the host graph resulting from applying Lemma 19 using $\epsilon' = \epsilon^{\frac{1}{\epsilon}+1}$ and let ϕ_i be the corresponding embedding. Let H be the graph resulting from adding a new vertex r' and for all i and all $v \in B_i$ adding an edge $(\phi_i(v), r')$ of length $d_G(v, r)$. That is, H is formed by connecting (all vertices of) all the H_i to r' (see Figure 5.2). Finally, set $\phi(v) = \phi_i(v)$ for all $v \in B_i - \{r\}$ and set $\phi(r) = r'$.

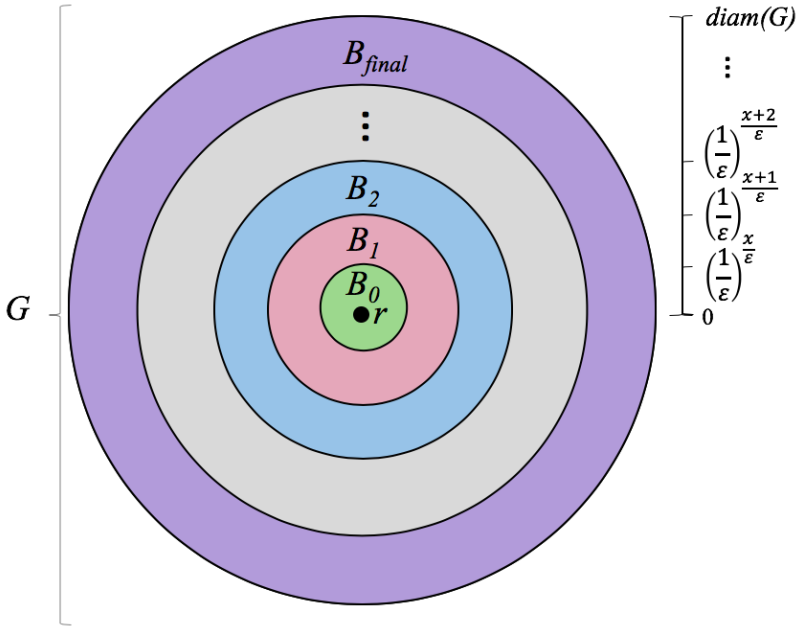


Figure 5.1: G is divided into bands $B_0, B_1, ..., B_{final}$ based on distance from r.

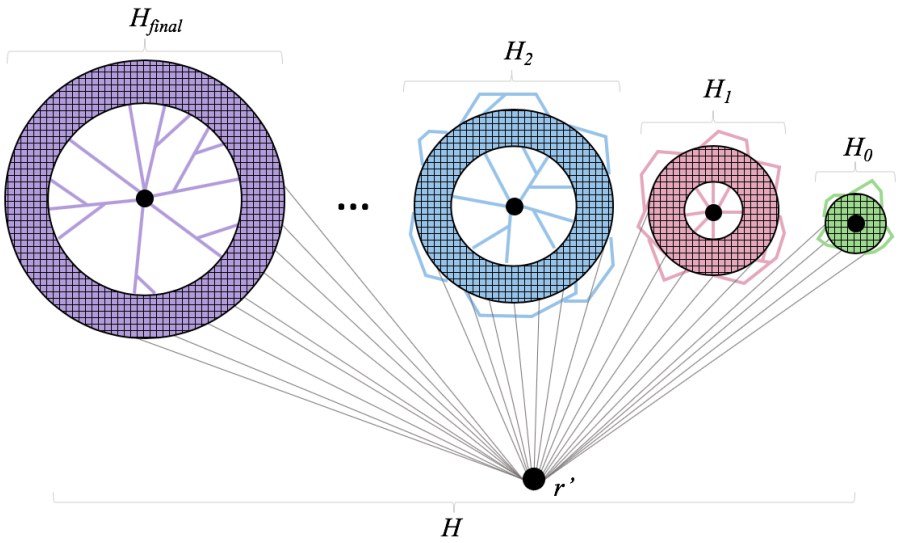


Figure 5.2: Each subgraph G_i of G is embedded into a host graph H_i. These graphs are joined via edges to a new depot r' to form a host graph for G.

We can assume that there are at most n bands, since empty bands would not contribute to the embedding. The runtime for constructing H is dominated by the construction of the H_i, which by Lemma 19 is polynomial.

Let H^- be the graph obtained from H by deleting r'. The connected components of H^- are $\{H_i\}_i$. By Lemma 19, the treewidth of each host graph H_i is at most $(\frac{1}{\epsilon'})^{c_0} = (\frac{1}{\epsilon})^{c_0(\epsilon^{-1}+1)}$ for some constant c_0. This also bounds the treewidth of H^-. Adding a single vertex to a graph increases the treewidth by at most one, so after adding r' back, the treewidth of H is $(\frac{1}{\epsilon})^{c_0(\epsilon^{-1}+1)} + 1 = (\frac{1}{\epsilon})^{c_1\epsilon^{-1}}$ for some constant c_1.

As for the metric approximation, it is clear that $d_G(u,v) \leq d_H(\phi(u),\phi(v))$ with probability 1. We use the following lemma to prove our desired distance bound (Inequality 5.1).

Lemma 20. *If $\epsilon d_G(v,r) < d_G(u,r) \leq d_G(v,r)$, then the probability that u and v are in different bands is at most ϵ.*

Proof. Let i be the nonnegative integer such that $d_G(u,r) = \frac{1}{\epsilon}^{(i+a)\frac{1}{\epsilon}}$ for some $a \in [0,1]$. Let b be the number such that $d_G(v,r) = \frac{1}{\epsilon}^{(i+b)\frac{1}{\epsilon}}$.

$$\frac{1}{\epsilon} \geq \frac{d_G(v,r)}{d_G(u,r)} = \frac{\frac{1}{\epsilon}^{(i+b)\frac{1}{\epsilon}}}{\frac{1}{\epsilon}^{(i+a)\frac{1}{\epsilon}}} = \frac{1}{\epsilon}^{(b-a)\frac{1}{\epsilon}}$$

Therefore

$$b - a \leq \epsilon$$

Consider two cases. If $b \leq 1$, then the probability that u and v are in different bands is $Pr[a \leq x < b] \leq \epsilon$.

If $b > 1$ then the probability that u and v are in different bands is $Pr[x \geq a \text{ or } x \leq b-1] \leq 1 - a + b - 1 = b - a \leq \epsilon$ □

We now prove Inequality 5.1. Let u and v be vertices in G. Without loss of generality, assume $d_G(u,r) \leq d_G(v,r)$. First we address the case where $d_G(u,r) \leq \epsilon d_G(v,r)$. Since $\phi(u)$ and $\phi(v)$ are both adjacent to r' in H,

$$\begin{aligned} d_H(\phi(u),\phi(v)) &\leq d_H(\phi(u),r') + d_H(\phi(v),r') \\ &= d_G(u,r) + d_G(v,r) \\ &\leq 2d_G(u,r) + d_G(u,v) \\ &\leq d_G(u,v) + 2\epsilon d_G(v,r) \end{aligned}$$

Therefore $E[d_H(\phi(u),\phi(v))] \leq d_G(u,v) + 3\epsilon[d_G(u,r) + d_G(v,r)]$

Now, suppose $d_G(u,r) > \epsilon d_G(v,r)$. If u and v are in the same band B_i, then by Lemma 19,

$$\begin{aligned} d_H(\phi(u),\phi(v)) &\leq d_{H_i}(\phi(u),\phi(v)) \leq d_G(u,v) + \epsilon' diam(G_i) \\ &\leq d_G(u,v) + \epsilon' 4\frac{1}{\epsilon}^{(i+x)\frac{1}{\epsilon}} = d_G(u,v) + \epsilon^{\frac{1}{\epsilon}+1} 4\frac{1}{\epsilon}^{(i+x)\frac{1}{\epsilon}} \\ &= d_G(u,v) + \epsilon 4\frac{1}{\epsilon}^{(i+x-1)\frac{1}{\epsilon}} \leq d_G(u,v) + 2\epsilon(d_G(u,r) + d_G(v,r)) \end{aligned}$$

In the final inequality, when $i = 0$, we use the fact that all nonzero distances are at least one to give a lower bound on $d_G(u,r)$ and $d_G(v,r)$.

If u and v are in different bands, then since $\phi(u)$ and $\phi(v)$ are both adjacent to r' in H, $d_H(\phi(u),\phi(v)) \leq d_H(\phi(u),r') + d_H(\phi(v),r') = d_G(u,r) + d_G(v,r)$. By Lemma 20, this case occurs with probability at most ϵ.

Therefore

$$E[d_H(\phi(u),\phi(v))] \leq (d_G(u,v) + 2\epsilon(d_G(u,r) + d_G(v,r))) + \epsilon[d_G(u,r) + d_G(v,r)]$$
$$\leq d_G(u,v) + 3\epsilon[d_G(u,r) + d_G(v,r)]$$

which proves Inequality 5.1 and completes the proof of Theorem 17.

The construction depends on planarity only via Lemma 19. We observe that if given a black box analogous to Lemma 19 for some other family of graphs, we can construct a comparable embedding. We formalize this generalization in the following lemma.

Lemma 21. *Let $\mathcal{G}$ be a family of graphs closed under vertex-induced subgraphs. Suppose that there is a function f and a polynomial-time algorithm that, for any graph G in $\mathcal{G}$, computes a graph H of treewidth at most $f(\epsilon)$ and an embedding ϕ of G into H such that, for all vertices u and v,*

$$d_G(u,v) \leq d_H(\phi(u),\phi(v)) \leq d_G(u,v) + \epsilon \cdot diam(G)$$

Then there is a function g and a randomized polynomial-time algorithm that, for any graph G in $\mathcal{G}$, computes a graph H with treewidth at most $g(\epsilon)$ and an embedding ϕ of G into H, such that, for every pair of vertices u, v of G, with probability 1 $d_G(u,v) \leq d_H(\phi(u),\phi(v))$, and

$$E[d_H(\phi(u),\phi(v))] \leq d_G(u,v) + \epsilon\,[(d_G(u,r) + d_G(v,r)]$$

This result is of particular interest for future research in Capacitated Vehicle Routing. Consider some such family of graphs $\mathcal{G}$ in which any $G \in \mathcal{G}$ can be embedded into a host graph of bounded treewidth with distances preserved up to an additive error of $\epsilon \cdot diam(G)$. A randomized PTAS for Capacitated Vehicle Routing with fixed capacity Q would follow as an immediate corollary of Lemma 21 and Theorem 7.

5.3 Derandomization

In this section, we prove Theorem 16 by derandomizing the embedding of Section 5.2 to give a deterministic PTAS for Capacitated Vehicle Routing on planar graph instances with fixed capacity Q.

The algorithm can be derandomized using a standard technique. The embedding of Theorem 17 partitions the vertices of the input graph into rings depending on a value x chosen uniformly at random from $[0, 1]$. However, the partition depends only on the distances of vertices from the root r. It follows that the number of partitions that can arise from different choices of x is at most the number of vertices. The deterministic algorithm tries each of these partitions, finding the corresponding solution, and returns the least costly of these solutions.

In particular, consider the optimum solution OPT. As shown in Theorem 7,

$$E[\sum_{(u,v)\in OPT} d_H(\phi(u), \phi(v))] = \sum_{(u,v)\in OPT} E[d_H(\phi(u), \phi(v))]$$

$$\leq (1+\epsilon) cost_G(OPT)$$

So for some choice of x, the induced cost of OPT in H is nearly optimal, and the dynamic program will find a solution that costs at most as much. This completes the proof of Theorem 16.

Chapter 6

A PTAS for Capacitated Vehicle Routing in Graphs of Bounded Highway Dimension

6.1 Introduction

In this chapter we present a PTAS for Capacitated Vehicle Routing with fixed capacities in graphs of bounded highway dimension. To achieve this result, we again use the strategy outlined in Section 3.3 of embedding a graph of bounded highway dimension into a graph of bounded treewidth in a way that preserves distances up to Error Allowance 3.3 (see Section 3.3.1). Specifically, in this chapter we present the following embedding result.

Theorem 18. *There is a function f, such that for every $\epsilon > 0$, graph G of highway dimension η, and vertex $r \in V_G$ there is a polynomial-time algorithm that constructs a host graph H with treewidth at most $f(\epsilon, \eta)$ and an embedding $\phi(\cdot)$ of G into H such that $\forall u, v \in V_G$,*

$$d_G(u,v) \leq d_H(\phi(u), \phi(v)) \leq d_G(u,v) + \epsilon(d_G(u,r) + d_G(v,r))$$

Given such an embedding, the dynamic-programming algorithm presented in Section 3.3.2 can

then be used to find an optimal solution in the host graph which can be mapped back to a near-optimal solution in the input graph. The following PTAS follows as a corollary of Theorem 6 by using $\epsilon' = \frac{\epsilon}{Q}$ in the above embedding (see Section 3.3).

Theorem 19. *For any $\epsilon > 0$, $Q > 0$, and $\eta > 0$, there exists a polynomial-time algorithm for* Capacitated Vehicle Routing *with capacity Q on graphs of highway dimension η that returns a solution whose cost is at most $1 + \epsilon$ times optimal.*

We then show how to extend this embedding to accommodate multiple depots. Specifically, we show the following embedding result.

Theorem 20. *There is a function f, such that for every $\epsilon > 0$, graph G of highway dimension η, and set R of vertices of G there is a polynomial-time algorithm that constructs a host graph H with treewidth at most $f(\epsilon, \eta, |R|)$ and an embedding $\phi(\cdot)$ of G into H such that $\forall u, v \in V_G$,*

$$d_G(u, v) \leq d_H(\phi(u), \phi(v)) \leq (1 + \epsilon) d_G(u, v) + \epsilon \min(d_G(R, u), d_G(R, v))$$

Given such an embedding, the following PTAS for Multi-Depot Capacitated Vehicle Routing follows as a corollary of Theorem 8 by using $\epsilon' = \frac{\epsilon}{2Q}$ in the above embedding (see Section 3.4.1).

Theorem 21. *For any $\epsilon > 0$, $\eta > 0$, ρ and $Q > 0$, there is a polynomial-time algorithm that, given an instance of* Multi-Depot Capacitated Vehicle Routing *in which the capacity is Q, the number of depots is ρ and the graph has highway dimension at most η, finds a solution whose cost is at most $1 + \epsilon$ times optimum.*

Additionally, the multi-depot embedding of Theorem 23 satisfies the requirements of Theorems 9 and 10, giving fixed-parameter approximations (FPAs) for k-Center and k-Median in graphs of bounded highway dimension as corollaries (see Section 3.4.2).

Theorem 22. *There is a function $f(\cdot, \cdot, \cdot)$ and a constant c such that, for each of the problems k-Center and k-Median, for any $\eta > 0, k > 0$ and $\epsilon > 0$, there is an $f(\eta, k, \epsilon)n^c$ algorithm that, given an instance in which the graph has highway dimension at most η, finds a solution whose cost is at most $1 + \epsilon$ times optimum.*

6.1.1 Background on Highway Dimension

The notion of *highway dimension* was introduced by Abraham et al. [1, 3] to explain the efficiency of some shortest-path heuristics. The motivation of this parameter comes from the work of Bast et al. [18, 19] who observed that, on a road network, a shortest path from a compact region to points that are far enough must go through one of a small number of nodes. They experimentally showed that the US road network has this property. Abraham et al. [1–3] proved results on the efficiency of shortest-path heuristics on graphs with bounded highway dimension.

For $\varkappa \in \mathbb{R}^+$ and $v \in V$, $B_\varkappa(v) = \{u \in V | d(u,v) \leq \varkappa\}$ denotes the ball with center v and radius $\varkappa$. The definition of highway dimension we use is from [47]. Let c be a constant greater than 4.

Definition 1. The *highway dimension* of a graph G is the smallest integer η such that for every $\varkappa \in \mathbb{R}^+$ and $v \in V$, there is a set of at most η vertices in $B_{c\varkappa}(v)$ such that every shortest path in $B_{c\varkappa}(v)$ of length at least $\varkappa$ intersects this set.

Abraham et al. [3] recognized that parameterizing problems by highway dimension could lead to improved algorithms. As described in Chapter 1, small highway dimension is a property observed in many transportation and road networks, so NP-hard optimization problems that arise in road networks are natural candidates for study. Feldmann [46] and Feldmann et al. [47] inaugurated this line of research, giving (respectively) a constant-factor approximation algorithm for one problem and quasi-polynomial-time approximation schemes for several other problems. In this chapter, we give the first *polynomial-time approximation schemes* (PTASs) for classical optimization problems in graphs of small highway dimension.

Alternative Definitions of Highway Dimension

The definition of highway dimension we use is the one given by Feldmann et al. [47]. However, alternate definitions exist. We summarize here the differences between them that are discussed in Feldmann et al. The original definition comes from Abraham et al. [3], in 2010. Their work uses $c = 4$, but interestingly they remark that "one could use constants bigger than 4". Nonetheless, Feldmann et al. [47] show that changing the constant is not innocuous: for any constant c, there is a graph with n vertices that has highway dimension 1 with respect to c and highway dimension $\Omega(n)$ with respect to any $c' > c$.

Another definition of highway dimension comes from a 2011 paper of Abraham et al. [1]. Their

definition differs from Definition 1 in that they use $c = 2$ and all shortest paths of length in $(\rho, 2\rho]$ that *intersect* the ball $B_{2\rho}(v)$ (not just the ones that stay inside the ball). This is a generalization of Definition 1 for $c = 4$: a path of length at most 2ρ that intersects the ball $B_{2\rho}(v)$ is also entirely contained in the ball $B_{4\rho}(v)$. As is, the results of Feldmann et al. and, consequently, ours as well cannot be generalized to this definition.

The last noteworthy definition of highway dimension was also introduced by Abraham et al. [2] in a journal paper in 2016 (we use h to denote this parameter, differentiating it from η). This definition implies that if a metric has bounded highway dimension, then it has also bounded doubling dimension. This definition is stricter than the one of 2010 and therefore stricter than ours as well. Feldmann et al. show that if a metric has highway dimension h according to the 2016 definition, it has highway dimension $O(h^2)$ according to the 2010 definition.

The definition chosen by Feldmann et al. is motivated by several considerations. Their definition is more flexible, as the choice of the constant c gives some slack. It also models more networks than the definition from [2]. Since the latter implies low doubling dimension, it cannot, for example, represent air traffic networks. These hub-and-spoke networks are star-like at large airports which causes large doubling dimension. Nevertheless, as noted in Feldman et al. [47], these networks have low highway dimension according to their definition, "since the airports act as hubs, which become sparser with growing scales as longer routes tend to be serviced by bigger airports."

6.1.2 Related Work

Metric embeddings of bounded-highway-dimension graphs

As mentioned in Section 6.1.1, Feldmann [46] and Feldmann et al. [47] inaugurated research into approximation algorithms for NP-hard problems in bounded-highway-dimension graphs. Feldmann et al. [47] discovered quasi-polynomial-time approximation schemes for Traveling Salesman, Steiner Tree, and Facility Location. The key to their results is a probabilistic metric embedding of bounded-highway dimension graphs into graphs of small treewidth. The *aspect ratio* of a graph with edge-lengths is the ratio of the maximum vertex-to-vertex distance to the minimum vertex-to-vertex distance. Feldmann et al. show that, for any $\epsilon > 0$, for any graph G of highway dimension η, there is a probabilistic embedding $\phi(\cdot)$ of G of expected distortion $1+\epsilon$ into a randomly chosen graph H whose treewidth is polylogarithmic in the aspect ratio of G (and also depends on ϵ

and η). There are two obstacles to using this embedding in achieving approximation schemes:

- The distortion is achieved only in expectation. That is, for each pair u, v of vertices, the expected $\phi(u)$-to-$\phi(v)$ distance in H is at most $(1+\epsilon)$ times the u-to-v distance in G.
- The treewidth depends on the aspect ratio of G, so is only bounded if the aspect ratio is bounded.

The first is an obstacle for problems (e.g. k-Center) where individual distances need to be bounded; this does not apply to problems such as Traveling Salesman or Capacitated Vehicle Routing where the objective is a sum of lengths of paths. The second is the reason that Feldmann et al. obtain only quasi-polynomial-time approximation schemes, and seems to be an obstacle to obtaining true polynomial-time approximation schemes.

Nevertheless, the techniques introduced by Feldmann et al. are at the core of our embedding results. We build heavily on their framework.

We refer the reader to Section 3.2 for an overview of related work on Capacitated Vehicle Routing.

k-Center

Recall, the goal in k-Center is to select a set of k vertices (*centers*) so as to minimize the maximum distance of a vertex to the nearest center. This problem might arise, for example, in selecting locations for k firehouses.

For k-Center, when the number k of centers is unbounded, for any $\delta > 0$, it is NP-hard [53, 60] to obtain a $(2-\delta)$-approximation, even in the Euclidean plane under L_1 or L_∞ metrics,[1] even in unweighted planar graphs [87], and even in n-vertex graphs with highway dimension $O(\log^2 n)$ [46]. It is not yet known to be NP-hard in graphs with constant highway dimension, but Feldmann [46] shows that, under the Exponential Time Hypothesis, the running time of an algorithm achieving a $(2-\delta)$-approximation would be doubly exponential in a polynomial in the highway dimension.

These negative results suggest considering the problem in which k is bounded by a constant. However, Feldmann [46] shows that $(2-\epsilon)$-approximation is $W[2]$ hard for parameter k, suggesting that the running time of any such approximation algorithm would not be bounded by a polynomial whose degree is independent of k. Thus even for constant k, finding a much better approximation seems to

[1] Approximation better than 1.822 is hard under L_2, see [44].

require that we restrict the metric. Feldmann [46] gave a polynomial-time 3/2-approximation algorithm for bounded-highway-dimension graphs, and raised the question of whether a better approximation ratio could be achieved. Our Theorem 22 answers this question by showing a fixed-parameter approximation scheme for k-Center.

Note that the results of Feldmann [46] are based on the definition of highway dimension of 2011 [1], but can be adapted to the definition we use here.

k-Median

Recall, the objective of k-Median is to find k centers so as to minimize the *sum* (or average) vertex-to-center distance. For k-Median, constant-factor approximation algorithms have been found for general metric spaces [9, 63, 91]. The best known approximation ratio for k-Median in general metrics is $1 + \sqrt{3}$ [77], and it is NP-hard to approximate within a factor of $1 + 2/e$ [55]. For k-Median in d-dimensional Euclidean space, PTASs have been found when k is fixed (e.g. [12]) and when d is fixed (e.g. [8]) but there exists no PTAS if k and d are part of the input [57]. Recently Cohen-Addad et al. [32] gave a local search-based PTAS for k-Median in edge-weight planar graphs that extends more generally to minor-closed graph families.

6.2 Preliminaries

Recall, for any vertex subsets $W \subseteq V$ and vertex $v \in V$ we use $d(v, W)$ to denote $\min_{w \in W} d(v, w)$, and we use $diam(W)$ to denote $\max_{u,v \in W} d(u, v)$.

Let $Y \subseteq X$ be a subset of elements in a metric space (X, d). Y is a *δ-covering* of X if for all $x \in X$, $d(x, Y) \leq \delta$. Y is a *β-packing* of X if for all $y_1, y_2 \in Y$ with $y_1 \neq y_2$, $d(y_1, y_2) \geq \beta$. Y is an *ϵ-net* if it is both an ϵ-covering and an ϵ-packing.

6.2.1 Shortest-Path Covers

Instead of working directly with Definition 1, we use the concept of a *shortest-path cover*, which as noted in Abraham et al. [1] is a closely related and more convenient tool.

Recall that c is a constant greater than 4.

Definition 2. For a graph G with vertex set V and $\boldsymbol{\imath} \in R^+$, a *shortest-path cover* $\mathrm{SPC}(\boldsymbol{\imath}) \subseteq V$ is a set of *hubs* such that every shortest path of length in $(\boldsymbol{\imath},\ c\boldsymbol{\imath}/2]$ contains at least one hub from this

set. Such a cover is called *locally k-sparse* for scale $\varkappa$ if every ball of diameter $c\varkappa$ contains at most k vertices from SPC($\varkappa$).

For a graph of highway dimension η, Abraham et al. [1] showed how to find a $\eta \log \eta$-sparse shortest-path cover in polynomial time (though they show it for a different definition of highway dimension ($c = 4$), the algorithm can be straightforwardly adapted). This result justifies using shortest-path covers instead of directly using highway dimension.

Town Decomposition

Feldmann et al. observed that a shortest-path cover for scale $\varkappa$ naturally defines a clustering of the vertices into *towns* [47]. Informally, a *town* at scale $\varkappa$ is a subset of vertices that are close to each other and far from other towns and from the shortest-path cover for scale $\varkappa$. Formally, a town is defined by at least one $v \in V$ such that $d(v, \text{SPC}(\varkappa)) > 2\varkappa$ and is composed of $\{u \in V | d(u, v) \leq \varkappa\}$.

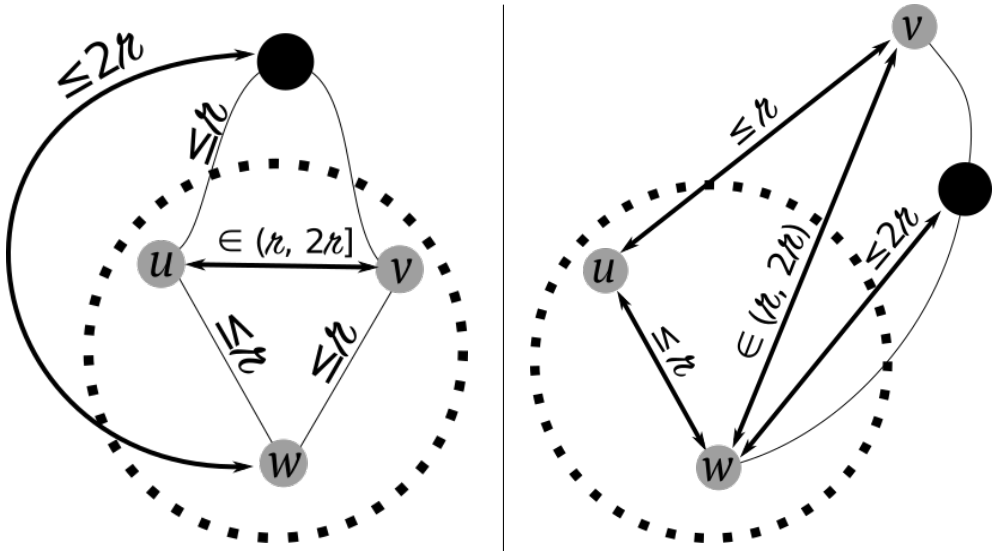
Lemma 22 describes key properties of towns proved in Feldmann et al and depicted in Figure 6.2.

Lemma 22 (Lemma 3.2 in [47])**.** *If T is a town at scale $\varkappa$, then*

1. *$diam(T) \leq \varkappa$, and*
2. *$d(T, V \setminus T) > \varkappa$.*

Proof. We give a sketch of the proof from [47]. For the first claim (illustrated in Figure 6.1a), let u and v be two vertices of the same town. By the definition of towns, there exists w such that $d(u, w) \leq \varkappa$ and $d(v, w) \leq \varkappa$. So $d(u, v) \leq 2\varkappa$. Assume by contradiction that $d(u, v) > \varkappa$. That is, $d(u, v) \in (\varkappa, 2\varkappa] \subseteq (\varkappa, c\varkappa/2]$ (because $c > 4$). Then there exists a point of the shortest-path cover for scale $\varkappa$ on the path between u and v. This point is at a distance at most $\varkappa$ from u or v (because $d(u, v) \leq 2\varkappa$), and therefore is at a distance at most $2\varkappa$ from w, which contradicts the construction of the town. Therefore $d(u, v) \leq \varkappa$, and $diam(T) \leq \varkappa$.

We proceed similarly for the second point, illustrated in Figure 6.1b. Let $u \in T, v \in V \setminus T$, and suppose by contradiction that $d(u, v) \leq \varkappa$. The definition of T gives a point w such that $d(u, w) \leq \varkappa$, $d(v, w) > \varkappa$ and $d(w, SPC(\varkappa)) > 2\varkappa$. Combining the inequalities gives that $\varkappa < d(v, w) \leq d(u, v) + d(u, w) < 2\varkappa < c\varkappa/2$. By definition of the shortest-path cover, there is a hub h on the shortest path between v and w, so $d(w, h) \leq d(w, v) \leq 2\varkappa$, which contradicts the choice of w. □



(a) First contradiction: $d(u, v) \leq r$ (b) Second contradiction : $d(T, V \setminus T) > r$

Figure 6.1: The shortest path cover is represented with black dots, the other points are grey dots. The dashed circles are towns. (Figure by David Saulpic).

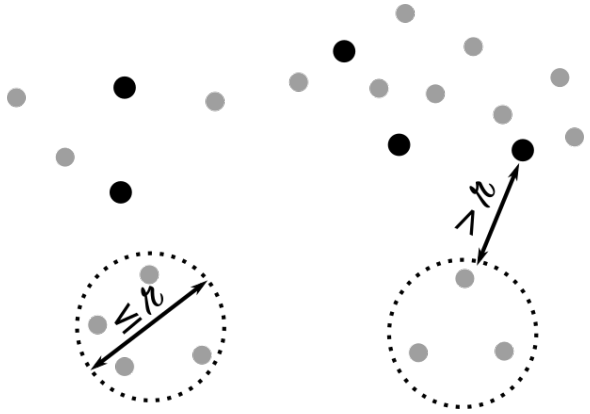


Figure 6.2: Illustration of Lemma 22 (Figure by David Saulpic).

Feldmann et al. [47] define a recursive decomposition of the graph using the concept of towns, which we adopt for this chapter. First, scale all distances so that the shortest point-to-point distance is a little more than $c/2$. Then fix a set of scales $r_i = (c/4)^i$. We say that a town T at scale r_i is on *level* i. Note that the scaling ensures that $SPC(r_0) = \emptyset$, and therefore at level 0 every vertex forms a singleton town. Also note that the largest level is $r_{max} = \lceil \log_{c/4} diam(G_{scaled}) \rceil = \lceil \log_{c/4}(\frac{c}{2} \cdot \theta_G) \rceil$, where θ_G is the aspect ratio of the input graph. Similarly at this topmost level, $SPC(r_{max}) = \emptyset$ since there are no shortest paths that need to be covered. The only town at scale r_{max} is the town that contains the entire graph. We say that the town at scale r_{max} and the singleton towns at scale

ρ_0 are *trivial* towns. Since c is a constant greater than four, the total number of scales is linear in the input size.

Consider the set $\mathcal{T} = \{T \subseteq V | T \text{ is a town on level } i \in \mathbb{N}\}$ of towns for all levels. Because of the properties of Lemma 22, this set forms a laminar family and therefore has a tree structure. Towns on the same level are disjoint from each other. By the isolation property of the town (the second property of Lemma 22), vertices outside of a town must be far from the town, so a smaller town cannot be both inside and outside of a larger town. Indeed, if two towns intersect, then the smaller town must be entirely inside the bigger (see [47] for more details). The set $\mathcal{T}$ is called the *town decomposition* of G, with respect to the shortest-path cover, and is a key concept used in our embedding. In particular, the decomposition has the following properties:

Lemma 23 (Lemma 3.3 in [47])**.** *For every town T in a town decomposition $\mathcal{T}$,*

1. *T has either 0 children or at least 2 children, and*
2. *if T is a town at level i and has child town T' at level j, then $j < i$.*

Proof. We sketch the proof from [47]. The first property comes from the fact that every singleton is a town at level 0 and that if T' is a child town of T, then $T \setminus T' \neq \emptyset$. The second property is a consequence of the first one combined with the isolation of the child town T'. □

Approximate Core Hubs

Another insight that we adopt from Feldmann et al. is that rather than working with *all* hubs in the shortest-path covers, it is sufficient for approximation algorithms to retain only a representative subset of the hubs: for $\epsilon > 0$, Feldmann et al. define for each town T a set X_T of *approximate core hubs* which is a subset of $T \bigcap \cup_i \text{SPC}(\rho_i)$ with the properties described in Lemma 24. One key property of these sets is a bound on their *doubling dimension*. The doubling dimension of a metric is the smallest θ such that for every ρ, every ball of radius 2ρ can be covered by at most 2^θ balls of radius ρ.

Lemma 24 (Theorem 4.2 and Lemma 5.1 in [47])**.** *Let G be a graph of highway dimension η, and $\mathcal{T}$ be a town decomposition with respect to an inclusion-wise minimal, k-sparse shortest-path cover. For any town $T \in \mathcal{T}$,*

1. *if T_1 and T_2 are different child towns of T, and $u \in T_1$ and $v \in T_2$, then there is some $h \in X_T$ such that $d(P[u,v], h) \leq \epsilon d(u,v)$, where $P[u,v]$ is the shortest u-to-v path, and*

2. the doubling dimension of X_T is $\theta = O(\log(\eta k \log(1/\epsilon))$.

Intuitively, the shortest-path cover at scale $\boldsymbol{\imath}_i$ forms a set of points that covers *exactly* the shortest-paths of length in $(\boldsymbol{\imath}_i, \boldsymbol{\imath}_{i+1}]$. Therefore to cover the shortest-path between $u \in T_1$ and $v \in T_2$, we can use a hub at level j such that $d(u,v) \in (\boldsymbol{\imath}_j, \boldsymbol{\imath}_{j+1}]$. Unfortunately, taking all the shortest-path covers gives a set too big for our purpose. But since we want to cover *approximately* the distances between points in different child towns, we can take only a subset of the shortest-path covers that has a low doubling dimension. This subset can be found in $n^{O(1)}$ time as explained in Feldmann et al. [47].

Minimality of Shortest-Path Covers

Note that the result of Lemma 24 requires the shortest-path covers be inclusion-wise minimal. For the embedding we present in Section 6.4, however, it is useful to assume that the depot is not a member of any town except for the trivial topmost town containing all of G and bottommost singleton town containing just the depot. One way to ensure this is to add the depot to the shortest-path cover at every level, but this violates the minimality requirement. Lemma 25, however, shows that this assumption can be made *safely* without asymptotically changing our results.

We show that adding the depot to the shortest-path cover at every scale is *safe*. This is not immediately obvious, as this modification can greatly alter the town decomposition. However, the only risk in adding a hub to the shortest-path cover is exceeding the bound on the doubling dimension of X_T for some town T. Indeed, the only place where Feldmann et al. [47] depend on the shortest-path covers being inclusion-wise minimal is in the proof of this doubling-dimension bound. It is fairly simple to adapt their proofs to show that adding a fixed number of vertices to a minimal shortest-path cover at every level adds at most a small factor to the bound on the doubling dimension.

Instead of reproving their results, however, we modify the graph to give the desired property. In fact, we prove a more general statement for a set R of depots. Recall that c is a constant greater than four and that all edges have been scaled so that the smallest point-to-point distance is slightly more than $c/2$.

Lemma 25. *Any graph $G = (V, E)$ with highway dimension η, diameter Δ_G, and designated vertex set $R \subseteq V$ can be modified by adding $O(\eta^2 |R|^3 \log \Delta_G)$ new vertices and edges, such that the resulting*

graph $G' = (V', E')$

- *has highway dimension at most* $\eta + |R|$
- *for all* $u, v, \in V'$, $d_{G'}(u, v) \in (\frac{c}{2},\ \frac{3c}{4}\Delta_G]$
- *for all* $u, v \in V$, $d_{G'}(u, v) = d_G(u, v)$, *and*
- *for every* $r \in R$, *the only towns containing* r *in the town decomposition of* G' *are the trivial towns.*

Proof. Let $a \in \mathbb{Z}$ be the smallest integer such that $(\frac{c}{4})^a > \frac{c}{2}$ and let $b \in \mathbb{Z}$ be the smallest integer such that $(\frac{c}{4})^b > \Delta_G$. We modify G by adding, for each $i \in \{a,\ a+1, ...,\ b\}$ and each $r \in R$, $(\eta + |R|)^2$ copies of vertex v_i^r and an edge (r, v_i^r) of length $\ell_i = (\frac{c}{4})^i$ for each copy. This modification adds $(\eta + |R|)^2|R|(b - a + 1) = O(\eta^2|R|^3 \log \Delta_G)$ vertices and edges (see Figure 6.3). We show that each of the listed properties holds for the modified graph G'.

First, this modification increases the highway dimension by at most $|R|$, since adding all depots in R as hubs covers all newly introduced shortest paths.

Second the smallest introduced edge has length $(\frac{c}{4})^a > \frac{c}{2}$, and all point-to-point distances in G are already assumed to be greater than $\frac{c}{2}$. The largest introduced edge has length $(\frac{c}{4})^b > \Delta_G \geq (\frac{c}{4})^{b-1}$, so the largest point-to-point distance in G' is between two copies of v_b^r from distinct vertices $r_1, r_2 \in R$, namely $\Delta_{G'} \leq 2(\frac{c}{4})^b + \Delta_G < 3(\frac{c}{4})^b = \frac{3c}{4}(\frac{c}{4})^{b-1} \leq \frac{3c}{4}\Delta_G$.

Third, all newly added edges are only connected to vertices in R, so all point-to-point distances between vertices in V are preserved in G'.

Finally, recall that the trivial towns in the town decomposition of G' are the singleton towns at scale $\ell_0 = (\frac{c}{4})^0 = 1$ and the topmost town at scale $\ell_{max} = \lceil \log_{c/4} diam(G') \rceil$ that contains all of G'. By the second property above, all point-to-point distances in G' are greater than $c/2$ and, clearly, at most $diam(G')$. Therefore there are no shortest paths in G' in the intervals $(1, c/2]$ or $(\ell_{max}, \frac{c}{2}\ell_{max}]$, so $SPC(\ell_0) = SPC(\ell_{max}) = \emptyset$. So all of G' is in the trivial town at scale ℓ_{max} and each vertex $v \in V'$, including each $r \in R$, is in a trivial singleton town at scale ℓ_0. Consider some $r \in R$. We must show that r does not appear in a town at any scale $\ell_i \in [\ell_1, \ell_{max})$.

We first show that for every scale $\ell_i = (\frac{c}{4})^i \in [\frac{c}{4}, (\frac{c}{4})^b) = [\ell_1, \ell_b)$, every locally-sparse shortest-path cover $SPC\big((\frac{c}{4})^i\big)$ of G' includes r, and therefore r cannot be in any town at these scales. The shortest path cover $SPC\big((\frac{c}{4})^i\big)$ must contain a hub on each path with length in $((\frac{c}{4})^i, \frac{c}{2}(\frac{c}{4})^i]$. By

the first property above, $SPC\big((\frac{c}{4})^i\big)$ is guaranteed to be locally $(\eta+|R|)\log(\eta+|R|)$-sparse [1], so $\big|B_{c(\frac{c}{4})^i}(r)\cap \mathrm{SPC}((\frac{c}{4})^i)\big| \leq (\eta+|R|)\log(\eta+|R|) < (\eta+|R|)^2$. There are two cases to consider.

If $\frac{c}{4} \leq (\frac{c}{4})^i < (\frac{c}{4})^a$, then $(\frac{c}{4})^a \in ((\frac{c}{4})^i, \frac{c}{2}(\frac{c}{4})^i]$, since $(\frac{c}{4})^{a-1} \leq \frac{c}{2}$ implies $(\frac{c}{4})^a \leq \frac{c}{2}(\frac{c}{4}) \leq \frac{c}{2}(\frac{c}{4})^i$. Since each edge (r, v_a^r) has length $(\frac{c}{4})^a$, SPC $\big((\frac{c}{4})^i\big)$ must contain either r or all $(\eta+|R|)^2$ copies of v_a^r. By the sparsity argument above, $\mathrm{SPC}\big((\frac{c}{4})^i\big)$ must therefore contain r.

Otherwise, $(\frac{c}{4})^a \leq (\frac{c}{4})^i < (\frac{c}{4})^b$. Therefore, $(\frac{c}{4})^{i+1} \leq (\frac{c}{4})^b$, so there are $(\eta+|R|)^2$ newly added edges (r, v_{i+1}^r) of length $(\frac{c}{4})^{i+1}$. Furthermore $(\frac{c}{4})^{i+1} \in ((\frac{c}{4})^i, \frac{c}{2}(\frac{c}{4})^i]$, since $\frac{c}{4} < \frac{c}{2}$. Therefore $SPC\big((\frac{c}{4})^i\big)$ must contain either r or all $(\eta+|R|)^2$ copies of v_{i+1}^r. Again, by the sparsity argument above, $SPC\big((\frac{c}{4})^i\big)$ must therefore contain r.

What remains to show is that r is in no town at scales in $[\mathscr{r}_b, \mathscr{r}_{max})$. We show that in fact there are no non-trivial towns at these levels. Assume to the contrary that T is a non-trivial town at scale $\mathscr{r}_i \in [\mathscr{r}_b, \mathscr{r}_{max})$. By Lemma 22, $d_{G'}(T, V' \setminus T) > \mathscr{r}_i$. In particular, any edges in the cut $\delta(T)$ must have length greater than $\mathscr{r}_i$. Furthermore since T is non-trivial, $T \neq V'$, and since G' is connected, $\delta(T) \neq \emptyset$. Therefore there is some edge in G' that has length greater than $\mathscr{r}_i$ and thus greater than $\mathscr{r}_b$. However all edges from G have length at most $\Delta_G < \mathscr{r}_b$, and all newly added edges have length at most $\mathscr{r}_b$. Therefore no such town exists. □

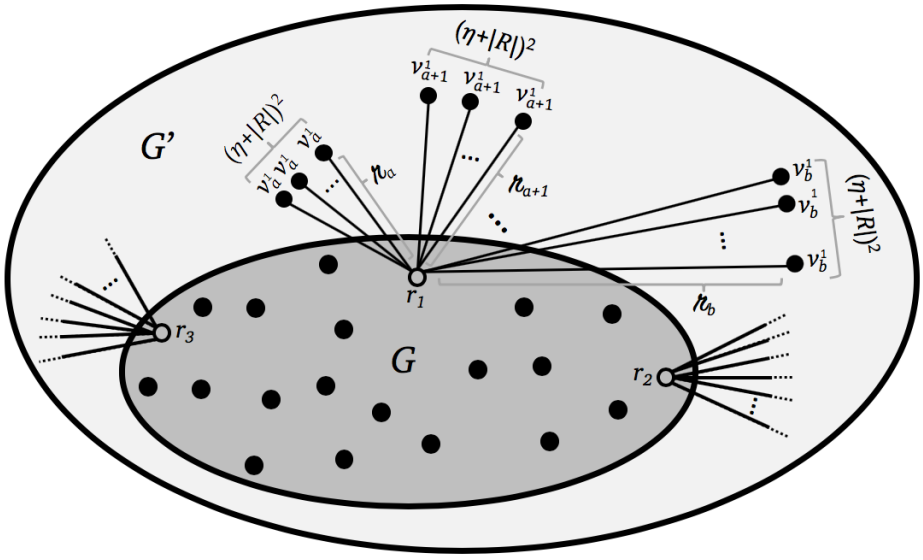


Figure 6.3: Here, $R = \{r_1, r_2, r_3\}$ is depicted as hollow vertices in G. For each $r \in R$, the modification of G to G' introduces new vertices and edges between these new vertices and r.

Note that after applying the modification of Lemma 25 to the (scaled) input graph, the resulting

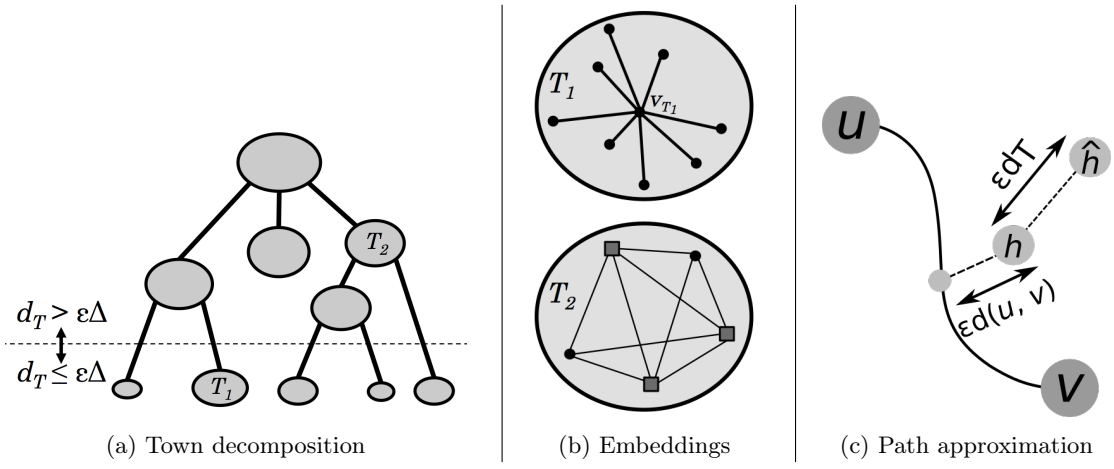
graph has size that is polynomial in the size of the original input.

6.3 Embedding for Graphs of Bounded Diameter

In this section we show how to construct an embedding for the case when the graph with bounded highway dimension also has has bounded diameter. This embedding, as formally described in Lemma 26, gives only a small *additive* error and will prove to be a useful tool for the overall embedding.

Lemma 26. *There is a function f, such that for every $\epsilon > 0$, graph G of highway dimension η and diameter Δ, there is a polynomial-time algorithm that constructs a host graph H with treewidth at most $f(\epsilon, \eta)$ and an embedding $\phi(\cdot)$ of G into H such that*

$$d_G(u, v) \leq d_H(\phi(u), \phi(v)) \leq d_G(u, v) + 4\epsilon\Delta$$



(a) Town decomposition (b) Embeddings (c) Path approximation

Figure 6.4: (a) An example town decomposition. T_1 has diameter at most $\epsilon\Delta$ and T_2 has diameter greater than $\epsilon\Delta$. (b) Two cases of town embeddings. T_1 is embedded as a star with center v_{T_1}. The embedding of T_2 connects all vertices in T_2 to all hubs in $\hat{X}_{T_2}$(depicted as squares). (c) Hub $\hat{h} \in \hat{X}_T$ is close to hub $h \in X_T$ which itself is close to the shortest u-to-v path.

We first present an algorithm to compute the host graph H and a tree decomposition of H. This algorithm relies on the town decomposition $\mathcal{T}$ of G, described in Section 6.2.

The host graph H is constructed as follows. First, consider a town T that has diameter $d \leq \epsilon\Delta$ but has no ancestor towns of diameter $\epsilon\Delta$ or smaller. We call such a town a *maximal* town of diameter at most $\epsilon\Delta$. The town T is embedded into a star: choose an arbitrary vertex v_T in T, and

for each $u \in T$, include an edge in H between u and v_T with length $d_G(u, v_T)$ equal to their distance in G (see Figures 6.4a and 6.4b).

Now consider a town T of diameter $diam_T > \epsilon\Delta$. The set of approximate core hubs X_T can be used as *portals* to preserve distances between vertices lying in different child towns of T. Specifically, by Lemma 24, for every pair of vertices (u, v) in different child towns of T, X_T contains a vertex that is close to the shortest path between u and v. In order to approximate the shortest paths, it is therefore sufficient to consider a set of points *close to* X_T. Let $\hat{X}_T$ be an $\epsilon diam_T$-net of X_T. For each $\hat{h} \in \hat{X}_T$ and $v \in T$, include an edge in H connecting v to $\hat{h}$ with length $d_H(v, \hat{h}) = d_G(v, \hat{h})$ equal to the v-to-$\hat{h}$ distance in G (see Figures 6.4a and 6.4b).

The tree decomposition D mimics the town decomposition tree: for each town T of diameter greater than $\epsilon\Delta$, there is a bag b_T. This bag is connected in D to all of the bags of child towns of T and contains all of the vertices of the net assigned to T and of the nets assigned to T's ancestors in the town decomposition. Formally, if A_T denotes the set of all towns that contain T, $b_T = \bigcup_{T' \in A_T} \hat{X}_{T'}$. Note that if T' is the parent of T in the town decomposition, $b_T = \hat{X}_T \cup b_{T'}$. Now for each maximal town T of diameter at most $\epsilon\Delta$ with parent town T', the tree decomposition contains a bag b_T^0 connected to a bag b_T^u for each vertex $u \in T$. We define $b_T^0 = \{v_T\} \cup b_{T'}$ and $b_T^u = \{u\} \cup b_T^0$.

Following Feldmann et al. [47], the above construction can be shown to be polynomial-time constructible. The following three lemmas therefore prove Lemma 18.

Lemma 27. *D is a valid tree decomposition of H.*

Proof. For D to be a valid tree decomposition of H, it has to satisfy three criteria (see Section 1.3).

First, every vertex must appear in some bag in D. As every vertex v in H is in some maximal town T of diameter at most $\epsilon\Delta$ (because every vertex forms a singleton town at level 0), there is a leaf b_T^v of D that contains v.

Second, for any edge (u, v) in H, u and v must appear together in some bag in D. Every vertex v has a leaf bag b_T^v of D that also contains all of the vertices adjacent to v in H: if an edge connects u and v, then either u or v is the center of the star for T, or u is in the net of some town that contains v. In both cases the construction of D ensure that u is in b_T^v.

Finally, for every vertex v, the bags that contain v must induce a connected subtree of D. Let T be a town such that b_T is the *highest* bag in the tree decomposition that contains v. As the towns at a given height of the town decomposition form a partition of the vertices, this town is

unique. Since the town decomposition has a laminar structure, v cannot appear in a bag that is not a descendant of b_T. Furthermore, by construction, v appears in all descendants of b_T, proving the third property. □

Lemma 28. *H has treewidth $O((\frac{1}{\epsilon})^{2\theta}\log_{\frac{c}{4}}\frac{1}{\epsilon})$, where θ is a bound on the doubling dimension of the sets X_T.*

Proof. Since the size of the bags is clearly bounded by the depth times the maximal cardinality of $\hat{X}_T$, it is enough to prove that, for each town T, $\hat{X}_T$ is bounded by $(\frac{1}{\epsilon})^{\theta}$, and that the tree decomposition has a depth $O(\log_{\frac{c}{4}}\frac{1}{\epsilon})$.

By Lemma 24, the doubling dimension of X_T is bounded by θ. $\hat{X}_T$ is a subset of X_T, so its doubling dimension is bounded by 2θ (see Gupta et al. [56]).Furthermore, the aspect ratio of $\hat{X}_T$ is $\frac{1}{\epsilon}$: the longest distance between members of $\hat{X}_T$ is bounded by the diameter $diam_T$ of the town, and the smallest distance is at least $\epsilon diam_T$ by definition of a net. The cardinality of a set with doubling dimension x and aspect ratio γ is bounded by $2^{x\lceil\log_2\gamma\rceil}$ (see [56] for a proof), therefore $|\hat{X}_T|$ is bounded by $(\frac{1}{\epsilon})^{2\theta}$.

We now prove that the tree decomposition has $O(\log_{\frac{c}{4}}\frac{1}{\epsilon})$ depth. Let T be a town of diameter $diam_T > \epsilon\Delta$ and let ɾ_i be the scale of that town. By Lemma 22, $diam_T \leq \text{ɾ}_i$, and since $\text{ɾ}_i = (\frac{c}{4})^i$ and $diam_T > \epsilon\Delta$, we can conclude that $i > \log_{\frac{c}{4}}\epsilon\Delta$. As the diameter of the graph is Δ, the biggest town has a diameter at most Δ. It follows that $\text{ɾ}_i \leq \Delta$ and therefore $i \leq \log_{\frac{c}{4}}\Delta$. The depth of b_T in the tree decomposition is therefore bounded by $\log_{\frac{c}{4}}\Delta - \log_{\frac{c}{4}}\epsilon\Delta = \log_{\frac{c}{4}}\frac{\Delta}{\epsilon\Delta} = \log_{\frac{c}{4}}\frac{1}{\epsilon}$. Furthermore, the tree decomposition of a town of diameter at most $\epsilon\Delta$ has depth 2. The overall depth is therefore $O(\log_{\frac{c}{4}}\frac{1}{\epsilon})$, concluding the proof. □

Lemma 29. *For all vertices u and v, $d_G(u,v) \leq d_H(u,v) \leq d_G(u,v) + 4\epsilon\Delta$.*

Proof. Let u and v be vertices in V, and let T be the town that contains both u and v such that u and v are in different child towns of T.

If T has diameter $diam_T \leq \epsilon\Delta$, then let T' be the maximal town of diameter at most $\epsilon\Delta$ that is an ancestor of T (possibly T itself). By construction, T' was embedded into a star centered at some vertex $v_{T'} \in T'$, so $d_H(u,v) \leq d_H(u,v_{T'}) + d_H(v_{T'},v) \leq d_G(u,v_{T'}) + d_G(v_{T'},v) \leq 2\epsilon\Delta$.

Otherwise if T has diameter $diam_T > \epsilon\Delta$, then by Lemma 24, there is some $h \in X_T$ such that $d_G(P[u,v],h) \leq \epsilon d(u,v)$. Since $\hat{X}_T$ is an $\epsilon diam_T$ cover of X_T, there is some $\hat{h} \in \hat{X}_T$ such that

$d(h, \hat{h}) \leq \epsilon diam_T$. The host graph H includes edges $(u, \hat{h})$ and $(\hat{h}, v)$, so $d_H(u, v) \leq d_H(u, \hat{h}) + d_H(\hat{h}, v) \leq d_G(u, h) + d_G(h, v) + 2\epsilon d(u, v) + 2\epsilon diam_T \leq d_G(u, v) + 4\epsilon\Delta$ (see Figure 6.4c).

Finally, since edge lengths in H are given by distances in G, $d_G(u, v) \leq d_H(u, v)$ for all $u, v \in V$. □

6.4 Main Embedding

In this section, we prove Theorem 18, restated here for convenience.

Theorem 18. *There is a function f, such that for every $\epsilon > 0$, graph G of highway dimension η, and vertex $r \in V_G$ there is a polynomial-time algorithm that constructs a host graph H with treewidth at most $f(\epsilon, \eta)$ and an embedding $\phi(\cdot)$ of G into H such that $\forall u, v \in V_G$,*

$$d_G(u, v) \leq d_H(\phi(u), \phi(v)) \leq d_G(u, v) + \epsilon(d_G(u, r) + d_G(v, r))$$

6.4.1 Embedding Construction

Instead of proving the distance bound of Theorem 18 directly, we show the following bound:

$$d_G(u, v) \leq d_H(\phi(u), \phi(v)) \leq d_G(u, v) + O(\epsilon)(d_G(u, r) + d_G(v, r))$$

Rescaling ϵ, namely using an $\epsilon' = \frac{\epsilon}{c'}$ (where c' is a constant chosen to compensate for the big-O in the above inequality) then gives the bound of Theorem 18. We further assume $\epsilon \leq \frac{1}{4}$ (if not, ϵ can be replaced by $\frac{1}{4}$ without asymptotically affecting performance).

Our construction relies on the assumption that the depot r does not appear in any non-trivial town. By Lemma 25 (using $R = \{r\}$), this assumption is *safe*, since the input graph can be modified to satisfy this assumption without (asymptotically) changing the highway dimension, diameter, or size of the graph. Furthermore since the modification preserves original distances, and all newly added vertices can be assumed to have no client demand, the modification does not affect the solution.

The root town in the decomposition, denoted T_0, is the town that contains the entire graph. We say that a town T that is a child of the root town is a *top-level town*, which means that the only town that properly contains T is T_0.

The assumption that the depot, r, does not appear in any non-trivial town implies that the top-level town that contains r is the trivial singleton town. This assumption helps to bound the distance between a top-level town T and the depot r: as $r \notin T$, Lemma 22 gives the bound $d(T, r) \geq \text{diam}(T)$. This bound will be useful in the construction of the host graph.

We use Lemma 26 to construct an embedding for each top-level town. It is not immediately obvious how to connect these town embeddings to form a global embedding. We cannot approximate X_{T_0} with a net as we did in Lemma 26, because the diameter of G may be arbitrarily large.

To address this issue, we define inductively the hub sets $X_0^0, X_0^1, ...$ such that X_0^k is a net of $X_{T_0} \cap B_{2^k}(r)$. Let X_0^0 be an ϵ-net of $X_{T_0} \cap B_1(r)$ that contains the depot, r, and for $k \geq 0$ let X_0^{k+1} be an $\epsilon 2^{k+1}$-net of the set $\left(X_{T_0} \cap (B_{2^{k+1}}(r) - B_{2^k}(r))\right) \bigcup X_0^k$ that contains the depot. This construction ensures that $X_0^{k+1} \cap B_{2^k}(r) \subseteq X_0^k$, which will be helpful in Section 6.4.3 to find a tree decomposition of the host graph. Note that we can assume $r \in X_{T_0}$, since adding it increases the doubling dimension by at most one and thus does not change the result of Lemma 24.

For a set of vertices $\mathcal{X} \subseteq V$, we define $l(\mathcal{X}) = \lceil \log_2(\max_{v \in \mathcal{X}} d(r, v)) \rceil$ (See Figure 6.5a).

For every child town T of T_0, the host graph connects every vertex v of T to every hub h in the *band* of hubsets $X_0^{l(T)}, \ldots, X_0^{l(T)+\log_2(1/\epsilon)}$ with an edge of length $d_G(v, h)$ (See Figure 6.5b).

6.4.2 Proof of Error Bound

In Lemma 31 we prove a bound on the error incurred by the embedding. Our proof makes use of the following lemma.

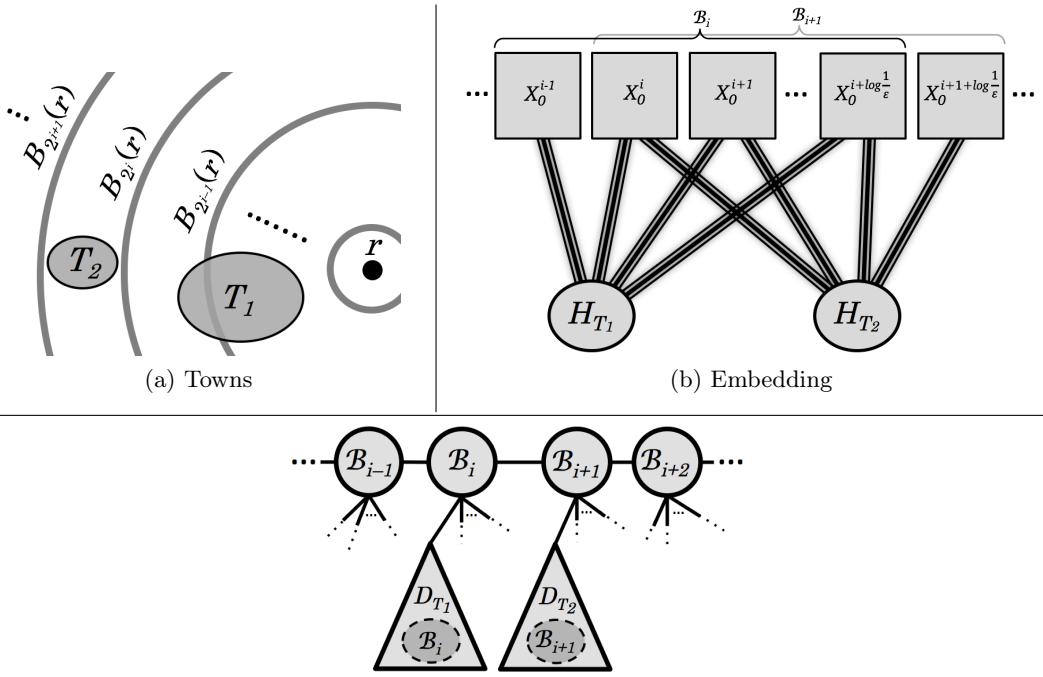
Lemma 30. *For all k, X_0^k is an $\epsilon 2^{k+1}$-covering of $X_{T_0} \cap B_{2^k}(r)$.*

Proof. We proceed by induction. By construction, X_0^0 is an ϵ-net (and thus also an $\epsilon 2^1$-covering) of $X_{T_0} \cap B_{2^0}(r)$. Assume that X_0^{k-1} is an $\epsilon 2^k$-covering of $X_{T_0} \cap B_{2^{k-1}}(r)$, and let $x \in X_{T_0} \cap B_{2^k}(r)$.

X_0^k is an $\epsilon 2^k$-net of the set $\left(X_{T_0} \cap (B_{2^k}(r) - B_{2^{k-1}}(r))\right) \bigcup X_0^k$, so if $x \in B_{2^k}(r) - B_{2^{k-1}}(r)$ then there is a $y \in X_0^k$ such that $d(x, y) \leq \epsilon 2^k < \epsilon 2^{k+1}$. Otherwise $x \in B_{2^{k-1}}(r)$. By assumption, there is an $\hat{x} \in X_0^{k-1}$ such that $d(x, \hat{x}) \leq \epsilon 2^k$, and by construction, there is a $y \in X_0^k$ such that $d(y, \hat{x}) \leq \epsilon 2^k$. Therefore $d(x, y) \leq \epsilon 2^k + \epsilon 2^k = \epsilon 2^{k+1}$. □

Lemma 31. *For all vertices u and v, $d_G(u, v) \leq d_H(u, v) \leq d_G(u, v) + O(\epsilon)(d_G(r, u) + d_G(r, v))$*

Figure 6.5: (a) Towns T_1 and T_2 are top-level towns, with $l(T_1) = i$ and $l(T_2) = i + 1$. (b) The embedding of each top-level town (shown as circles) is connected to a band of $\log_2 \frac{1}{\epsilon} + 1$ hub sets (shown as squares). Edges are striped to convey that they connect *all* vertices of the given hub-set endpoint to *all* vertices of the town-embedding endpoint. (c) The vertices of each bag $\mathcal{B}$ (shown as circles) are added to each bag of each descendant top-level-town tree decomposition (shown as triangles).



(a) Towns

(b) Embedding

(c) Tree decomposition

Proof. Consider two vertices u and v. Let T_u and T_v denote the top-level towns that contain u and v, respectively. There are two cases to consider.

If $T_u = T_v$, Lemma 22 gives $d_G(u, v) \leq diam(T_u) \leq d_G(T_u, V \setminus T_u)$, and therefore $diam(T_u) \leq \min\{d_G(r, u),\ d_G(r, v)\}$. Because $T_u = T_v$ is a top-level town, its embedding is given by Lemma 26, which directly gives the desired bound.

Otherwise $T_u \neq T_v$. Without loss of generality, assume that $d_G(u, r) \geq d_G(v, r)$. We show that there exists some X_0^k connected to u with a vertex $\hat{h} \in X_0^k$ close to $P[u, v]$.

By definition of the approximate core hubs, there exists $h \in X_{T_0}$ such that $d(h, P[u, v]) \leq \epsilon d(u, v)$.

Moreover, $h \in B_{2^{l(T_u)+2}}(r)$:

$$\begin{aligned}
d(r,h) &\leq d(r,u) + d(u,h) \\
&\leq d(r,u) + (1+\epsilon)d(u,v) \\
&\leq d(r,u) + (1+\epsilon)\big(d(r,u) + d(r,v)\big) && \text{by the triangle inequality} \\
&\leq d(r,u) + (1+\epsilon)\cdot 2d(r,u) && \text{since } d(u,r) \geq d(v,r) \\
&\leq (3+2\epsilon)2^{l(T_u)} \\
&\leq 2^{l(T_u)+2}
\end{aligned}$$

Since $h \in X_{T_0} \cap B_{2^{l(T_u)+2}}(r)$, then by Lemma 30, there is an $\hat{h} \in X_0^{l(T_u)+2}$ such that $d(\hat{h},h) \leq \epsilon 2^{l(T_u)+3}$. Since $\log_2 \frac{1}{\epsilon} \geq 2$, u is connected to $\hat{h}$ in the host graph.

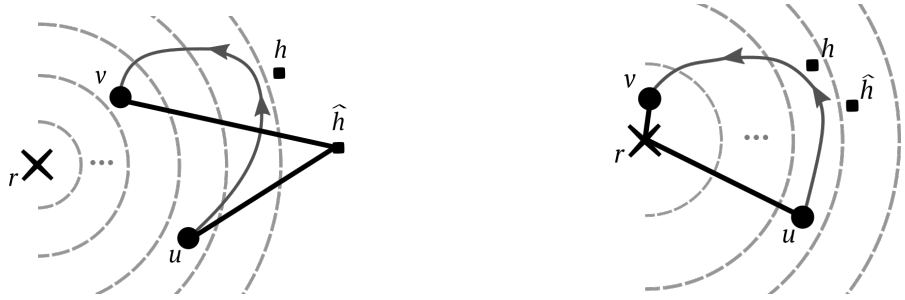
Depending on v, there remain two cases: either v is connected to $\hat{h}$ (see Figure 6.6a) or not (Figure 6.6b). First, if v is connected to $\hat{h}$ in the host graph, $d_H(v,\hat{h}) = d_G(v,\hat{h})$ (and the same holds for u). The triangle inequality gives therefore,

$$d_H(u,v) \leq d_G(u,\hat{h}) + d_G(v,\hat{h}) \leq \underbrace{d_G(u,h) + d_G(v,h)}_{\leq (1+2\epsilon)d_G(u,v) \text{ by definition of } h} + \underbrace{2d_G(\hat{h},h)}_{\leq 2\epsilon 2^{l(T_u)+3} = O(\epsilon)d(r,u)}$$

Since $d_G(u,v) \leq d_G(r,u) + d_G(r,v)$, we can conclude that,

$$d_H(u,v) \leq d_G(u,v) + O(\epsilon)(d_G(r,u) + d_G(r,v))$$

Otherwise, v is not connected to $\hat{h}$. That means that either $l(T_u)+2 < l(T_v)$ or $l(T_u)+2 > l(T_v) + \log_2 \frac{1}{\epsilon}$. We exclude the first case by noting that since the diameter of a town is less than its distance to the depot, $d_G(v,r) \leq d_G(u,r)$ implies that $l(T_v) \leq l(T_u)+1$. The second case implies that $d_G(r,u) \geq O(\frac{1}{\epsilon})d_G(r,v)$. Since the host graph connects the depot r to all vertices, $d_H(u,v) \leq d_G(r,u) + d_G(r,v) \leq d_G(u,v) + 2d_G(r,v) \leq d_G(u,v) + O(\epsilon)(d_G(r,u) + d_G(r,v))$. □



(a) u and v are both connected to $\hat{h}$ (b) v is not connected to $\hat{h}$

Figure 6.6: The shortest path between u and v in G is indicated by the curved, directed lines. The path in the host graph is represented by the straight lines. (Figure by David Saulpic).

6.4.3 Tree Decomposition

We present here the construction of a tree decomposition D of the host graph with a bounded width.

For each $k > 0$ let $\mathcal{B}_k = \bigcup_{i=k-1}^{k+\log_2(1/\epsilon)} X_0^i$ be the bag consisting of the kth band of hubsets. For a top-level town T, the tree decomposition D connects the decomposition D_T given by Lemma 26 to the bag $\mathcal{B}_{l(T)}$. Moreover, we add all vertices that appear in $\mathcal{B}_{l(T)}$ to all bags in the tree D_T. Finally, for every k we connect $\mathcal{B}_k$ to both $\mathcal{B}_{k-1}$ and $\mathcal{B}_{k+1}$ in D. (See Figure 6.5b.)

Lemma 32. *D is a valid tree decomposition of the host graph H.*

Proof. For D to be a valid tree decomposition of H, it has to satisfy the three properties listed in Section 1.3.

First, because the top-level towns are a partition of the vertices, each vertex appears in some tree decomposition D_T. The union of all bags is therefore $V(H)$.

Next, let (u, v) be an edge of H. There are two cases to consider: if u and v are in the same top-level town, Lemma 26 ensures that u and v appear together in some bag. Otherwise, as the top-level towns are disjoint, one of u or v is a hub connected to the other. Without loss of generality assume that v is a hub of X_0^k for some $k \in \{l(T_u), ..., l(T_u) + \log_2 \frac{1}{\epsilon}\}$. In this case, $v \in \mathcal{B}_{l(T_u)}$, so v is added to all the bags of D_{T_u}, and in particular is in some bag that contains u.

Finally, let v be a vertex that appears in two different bags. If the two bags are in the tree decomposition of the same top-level town T, Lemma 26 ensures that the bags are connected in D_T and thus also in D. Otherwise, as the top-level towns are disjoint, v must be a hub. Consider all nets

X_0^k containing v. Any bag $\mathcal{B}_\ell$ containing such a net also contains v. Let $\mathcal{I} = \{k | v \in X_0^k\}$. We prove that $\mathcal{I}$ is an interval, and therefore that the bags $\mathcal{B}_\ell$ are connected. Let $i = \min(\mathcal{I})$ and $j = \max(\mathcal{I})$. As $v \in X_0^i$, it must be that $v \in B_{2^i}(r) \subseteq B_{2^{i+1}}(r) \subseteq ... \subseteq B_{2^j}(r)$. Repeatedly applying the property $X_0^k \cap B_{2^{k-1}}(r) \subseteq X_0^{k-1}$ proves that for all $k \in \{i, i+1, ...j\},\ v \in X_0^k$. Therefore $\mathcal{I}$ is an interval, and the bags $\mathcal{B}_\ell$ such that $v \in \mathcal{B}_\ell$ are connected. Finally, we show that interval $\mathcal{I}$ includes $\mathcal{B}_l(T_v)$. Since v is a hub, $v \in X_0^{l(\{v\})}$. By Lemma 22, $d(v,r) > diam(T_v)$, so $l(T_v) - 1 \leq l(\{v\}) \leq l(T_v)$, and therefore $v \in \mathcal{B}_l(T_v)$. Since the vertices of $\mathcal{B}_l(T_v)$ are added to every bag in D_{T_v}, the bags containing v form a connected subtree of D. □

Lemma 33. *For all k, $|X_0^k| \leq (\frac{2}{\epsilon})^{2\theta}$.*

Proof. Since X_0^k is a subset of X_{T_0}, it has doubling dimension 2θ (see Lemma 24). Since X_0^k is a $\epsilon 2^k$-net, the smallest distance between two hubs in X_0^k is at least $\epsilon 2^k$. Moreover, since $X_0^k \subseteq B_{2^k}(r)$, the longest distance between two hubs is at most $2 \cdot 2^k$, therefore, X_0^k has an aspect ratio of at most $\frac{2}{\epsilon}$. The bound used in Lemma 28 on the cardinality of a set using its aspect ratio and its doubling dimension concludes the proof. □

Lemma 34. *The tree decomposition D has bounded width.*

Proof. Bag $\mathcal{B}_i$ is the union of $\log_2 \frac{1}{\epsilon} + 2$ sets X_0^k. Lemma 33 gives $|X_0^k| \leq (\frac{2}{\epsilon})^\theta$, therefore $|\mathcal{B}_i| \leq (\log_2(\frac{1}{\epsilon}) + 2)(\frac{2}{\epsilon})^{2\theta}$. Moreover, by Lemma 28, each bag of the D_T decompositions has a cardinality bounded by $O((\frac{2}{\epsilon})^\theta \log_{\frac{c}{4}} \frac{1}{\epsilon})$. Therefore, since each bag of the decomposition D is either a bag $\mathcal{B}_i$ for some i or is formed by adding a single bag $\mathcal{B}_i$ to some bag of a D_T decomposition, its size is bounded. Therefore D has a bounded width. □

6.5 Embedding for Multiple Depots

In this section, we extend Theorem 18 to the multiple-depot setting and then show applications of this new embedding. Specifically we prove the following multiple-depot embedding result.

Theorem 23. *There is a function f, such that for every $\epsilon > 0$, graph G of highway dimension η, and set R of vertices of G there is a polynomial-time algorithm that constructs a host graph H with treewidth at most $f(\epsilon, \eta, |R|)$ and an embedding $\phi(\cdot)$ of G into H such that $\forall u, v \in V_G$,*

$$d_G(u,v) \leq d_H(\phi(u), \phi(v)) \leq (1+\epsilon) d_G(u,v) + \epsilon \min(d_G(R,u), d_G(R,v))$$

Instead of proving this distance bound exactly, we show that for all vertices u and v,

$$d_G(u,v) \leq d_H(\phi(u),\phi(v)) \leq (1+O(\hat{\epsilon}))d_G(u,v) + O(\hat{\epsilon})\min(d_G(R,u), d_G(R,v))$$

Theorem 23 follows from setting $\hat{\epsilon} = \frac{\epsilon}{\hat{c}}$ for an appropriate constant $\hat{c}$.

We modify the embedding of Theorem 18 to accommodate the multiple-depot setting. We assume that the vertices of R do not appear in non-trivial towns. This assumption is *safe* because, using R, we can apply the modification of Lemma 25 to satisfy this assumption without asymptotically changing the diameter or size of the graph. Note that the modification preserves all distances from the original input graph but increases the highway dimension to $\eta + |R|$.

The algorithm computes the town decomposition with respect to the shortest-path covers, and embeds the top-level towns using Lemma 26. By analogy with Section 6.4, we define the set X_0^i to be a $\hat{\epsilon}2^i$-net of $\cup_{r\in R} B_{2^i}(r)$ (and we ensure moreover that the X_0^i are nested). We also modify the definition of $l(\mathcal{X})$: for a set $\mathcal{X}$, $l(\mathcal{X}) = \lceil \log_2(\max_{v\in\mathcal{X}} d(R,v))\rceil$. Following Section 6.4, the host graph connects every vertex v of a town T to every hub h in $X_0^{l(T)}, \ldots, X_0^{l(T)+\log_2(1/\hat{\epsilon})}$ with an edge of length $d_G(v,h)$.

We now prove that the embedding described above has the properties of Theorem 23. We use H to denote the host graph produced by the embedding. First, we prove the treewidth bound.

Proof. Let θ_R be the doubling dimension of the union of the approximate core hubs with R. The shortest-path covers are locally $(\eta + |R|)\log(\eta + |R|)$-sparse [1], therefore by Lemma 24, θ_R is $O\Big(\log\big((\eta+|R|)^2\log(\eta+|R|)\log(1/\hat{\epsilon}) + |R|\big)\Big)$, where the final $|R|$ term comes from the extra balls required to cover R. The proof of Lemma 33 directly gives that $|X_0^i| \leq |R|(\frac{2}{\hat{\epsilon}})^{\theta_R}$. Finally, following the proof of Lemma 34, the host graph has a treewidth bounded by a function of η, $|R|$ and $\hat{\epsilon}$. □

We now prove the distance-distortion bound on the embedding, completing the proof of Theorem 23.

Proof. Let u and v be two vertices in G and h be the approximate core hub such that $d_G(u,h) + d_G(v,h) \leq (1+O(\hat{\epsilon}))d_G(u,v)$. Let r_u, r_v and r_h denote the depots of R closest to u, v and h.

We first show the following bound on $d_G(h, r_h)$, which will be useful later in our proof:

$$d_G(h,r_h) \leq (1+O(\hat{\epsilon}))d_G(u,v) + \min(d_G(u,r_u), d_G(v,r_v)) \tag{6.1}$$

The definition of r_h gives $d_G(h, r_h) \leq d_G(h, r_u)$, and by the triangle inequality, $d_G(h, r_h) \leq d_G(h, u) + d_G(u, r_u)$. By definition, $d_G(h, u) \leq (1 + O(\hat{\epsilon}))d_G(u, v)$, so the bound holds with respect to u. The bound with respect to v is symmetric.

We consider three cases, based on $l(h)$, $l(T_u)$, and $l(T_v)$. These cases are illustrated in Figure 6.7.

First, consider the case in which $l(h) \leq l(T_u) + \log_2(1/\hat{\epsilon})$ and $l(h) \leq l(T_v) + \log_2(1/\hat{\epsilon})$ (see Figure 6.7a). Let $\hat{h}$ be the point in $X_0^{l(h)}$ closest to h. By definition of a net, $d_G(h, \hat{h}) \leq \hat{\epsilon}2^{l(h)} \leq 2\hat{\epsilon}d_G(h, r_h)$, and by construction, $\hat{h}$ is adjacent to u and v. In this case,

$$\begin{aligned} d_H(u, v) &\leq d_H(u, \hat{h}) + d_H(\hat{h}, v) \\ &\leq d_G(u, \hat{h}) + d_G(\hat{h}, v) \\ &\leq d_G(u, h) + d_G(v, h) + 2d_G(h, \hat{h}) \end{aligned}$$

By definition of h and $\hat{h}$, $d_H(u, v) \leq (1 + O(\hat{\epsilon}))d_G(u, v) + 4\hat{\epsilon}d_H(h, r_h)$, and using Equation 6.1

$$d_H(u, v) \leq (1 + O(\hat{\epsilon}))d_G(u, v) + O(\hat{\epsilon}) \min(d_G(u, R), d_G(v, R))$$

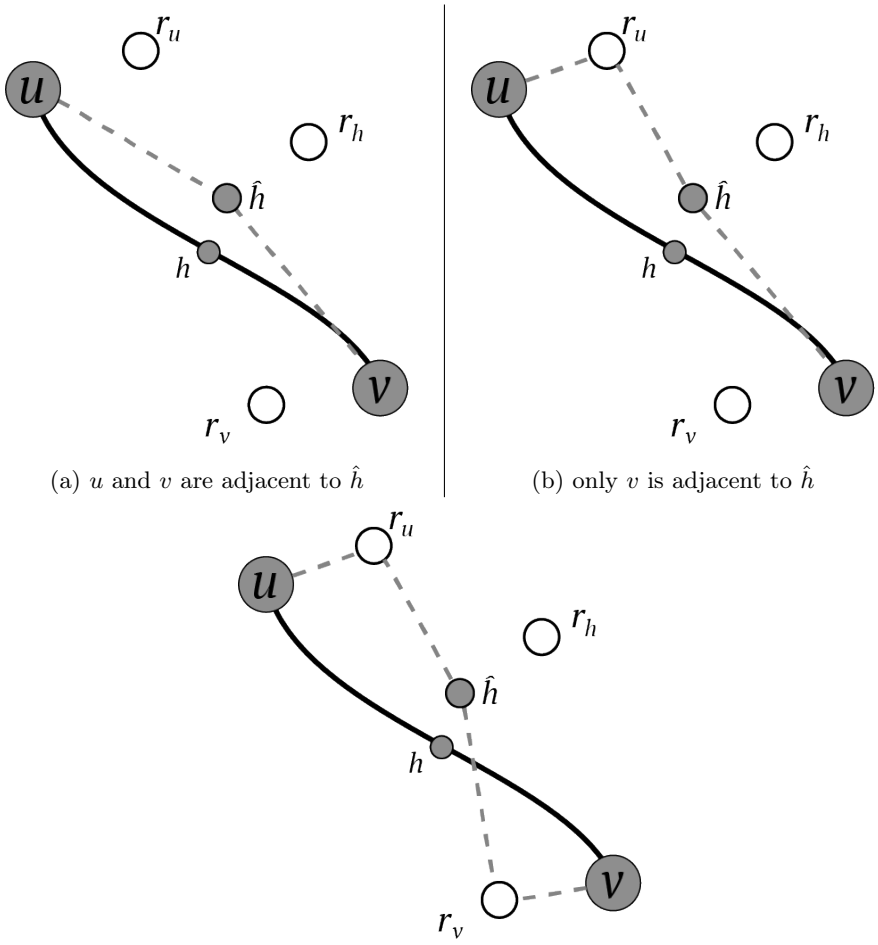
Second, consider the case in which $l(h) > l(T_u) + \log_2(1/\hat{\epsilon})$ but $l(h) \leq l(T_v) + \log_2(1/\hat{\epsilon})$ (see Figure 6.7b). In the host graph, v is adjacent to $\hat{h}$, but u is not. In particular, $d_G(r_h, h) > \frac{1}{\hat{\epsilon}}d_G(r_u, u)$. The shortest path between u and v is therefore approximated in the host graph by the path u, r_u, $\hat{h}$, v. The edge lengths in this path in H, however, are simply the corresponding distances in G, therefore $d_H(u, v) \leq d_H(u, r_u) + d_H(r_u, \hat{h}) + d_H(\hat{h}, v) \leq d_G(u, r_u) + d_G(r_u, \hat{h}) + d_G(\hat{h}, v)$. By the triangle inequality, $d_G(r_u, \hat{h}) \leq d_G(r_u, u) + d_G(u, \hat{h})$. Putting these pieces together gives,

$$\begin{aligned} d_H(u, v) &\leq 2d_G(u, r_u) + d_G(u, \hat{h}) + d_G(\hat{h}, v) \\ &\leq 2\hat{\epsilon}d_G(h, r_h) + d_G(u, h) + d_G(h, v) + 2d_G(\hat{h}, h) \end{aligned}$$

Recall that $d_G(h, \hat{h}) \leq 2\hat{\epsilon}d_G(h, r_h)$ and $d_G(u, h) + d_G(h, v) \leq (1 + \hat{\epsilon})d_G(u, v)$. Therefore, by Equation 6.1,

$$\begin{aligned} d_H(u, v) &\leq (1 + \hat{\epsilon})d_G(u, v) + 4\hat{\epsilon}d_H(h, r_h) \\ &\leq (1 + O(\hat{\epsilon}))d_G(u, v) + O(\hat{\epsilon}) \min(d_G(u, R), d_G(v, R)) \end{aligned}$$

Finally, consider the case in which $l(h) > l(T_u) + \log_2(1/\hat{\epsilon})$ and $l(h) > l(T_v) + \log_2(1/\epsilon)$ (see Figure 6.7c). Neither u nor v is adjacent to $\hat{h}$ in the host graph. In this case, the shortest path between u and v is approximated in the host graph by the path $u, r_u, \hat{h}, r_v, v$. Using the same arguments as in the previous case, $d_H(u,v) \leq (1+O(\hat{\epsilon}))d_G(u,v)+O(\hat{\epsilon})\min(d_G(u,R), d_G(v,R))$. □



(a) u and v are adjacent to $\hat{h}$

(b) only v is adjacent to $\hat{h}$

(c) u and v are not adjacent to $\hat{h}$

Figure 6.7: The shortest path between u and v is approximated in the host graph by the dashed line. In case (a), the distance from u and v to their centers is large compared to the distance between h and r_h; in case (b), only the distance from v to r_v is large, and in case (c) the distance from h to r_h is larger than the other two. (Figure by David Saulpic).

6.5.1 Applications of the Multiple-Depot Embedding

We now show several applications of this multiple-depot embedding that follow as corollaries to Theorem 23 and the extensions proved in Section 3.4.1. Note that the distance bound we show

in Theorem 23 is in fact better than what is required for the embeddings in Section 3.4.1, as $\min(d(u,R), d(v,R)) \leq d(u,R) + d(v,R)$.

Multi-Depot Capacitated Vehicle Routing

The first application we consider is to Multiple-Depot Capacitated Vehicle Routing with a constant number of depots. Let R denote the set of depots, and recall that S is the set of clients. We assume that any vertices added in the modification given in Lemma 25 do not have any client demand.

The embedding in Theorem 23, using $\epsilon' = \frac{\epsilon}{2Q}$, satisfies the embedding conditions required of Theorem 8. A PTAS for Multi-Depot Capacitated Vehicle Routing when the capacity Q, number of depots ρ, and highway dimension η are fixed, follows as a corollary.

Theorem 21. *For any $\epsilon > 0$, $\eta > 0$, ρ and $Q > 0$, there is a polynomial-time algorithm that, given an instance of* Multi-Depot Capacitated Vehicle Routing *in which the capacity is Q, the number of depots is ρ and the graph has highway dimension at most η, finds a solution whose cost is at most $1 + \epsilon$ times optimum.*

k-Center and k-Median

Another application of the embedding given in Theorem 23 is to get fixed-parameter approximations (FPA) for k-Center and k-Median in graphs with highway dimension η.

The embedding in Theorem 23, satisfies the embedding conditions required of Theorem 9 and Theorem 10 (see Section 3.4). Fixed-parameter approximation schemes for k-Center and k-Median in graphs of highway dimension η follow as corollaries.

Theorem 22. *There is a function $f(\cdot,\cdot,\cdot)$ and a constant c such that, for each of the problems k-Center and k-Median, for any $\eta > 0, k > 0$ and $\epsilon > 0$, there is an $f(\eta,k,\epsilon)n^c$ algorithm that, given an instance in which the graph has highway dimension at most η, finds a solution whose cost is at most $1 + \epsilon$ times optimum.*

Chapter 7

A 4/3-Approximation for Capacitated Vehicle Routing with General Capacities in Trees

7.1 Introduction

The results of the previous few chapters have addressed Capacitated Vehicle Routing with *fixed* capacities. What about the considerably more difficult problem of *general* capacity Q? Even in tree metrics, while the fixed-capacity case is polynomial-time solvable, Capacitated Vehicle Routing with general capacities is NP-hard [74]. In fact, for the unsplittable-demand setting, it is NP-hard to approximate to better than a 3/2 factor [52].

In this chapter we give a 4/3-approximation algorithm for Capacitated Vehicle Routing in trees with general capacities and splittable demands. This is the current best-known approximation ratio for this problem, and builds on the approach introduced by Hamaguchi and Katoh [59] and Asano, Katoh, and Kawashima [10].

Hamaguchi and Katoh [59] noted a simple lower bound for the splittable-demand variant of Capacitated Vehicle Routing in trees: every edge must be traversed by *at least* enough vehicles to accommodate all demand from the clients that use that edge on the shortest path to the depot. They then use this lower bound (denoted LB) to give a 1.5 approximation [59]. Following this work,

Asano et al. [10] use the same lower bound to achieve a $(\sqrt{41}-1)/4$ approximation. They also prove the following lemma (see Section 7.3.2 for details):

Lemma 35 ([10])**.** *There exist instances of* Capacitated Vehicle Routing *in trees whose optimal solution costs* $4/3 \cdot LB$*.*

This shows that the best possible approximation ratio using this lower bound would be a 4/3-approximation. Our result, stated in Theorem 24, achieves this ratio, and is therefore tight with respect to LB. No further improvements over our result can be made until a better lower bound is found.

Theorem 24. *There is a 4/3-approximation algorithm for* Capacitated Vehicle Routing *in trees that runs in* $O(n^2)$ *time and is tight with respect to* LB*.*

7.2 Overview of Techniques

Our work extends the techniques introduced by Asano et al. [10] which itself was an extension of the work of Hamaguchi and Katoh [59]. Specifically, Hamaguchi and Katoh describe a very natural lower bound that arises on tree instances [59]: the number of tours that traverse each edge must be at least enough to cover all demand in the subtree below the edge (the tree is assumed to be rooted at the depot). This introduces a minimum *traffic* value on the edge. Multiplying this value by two (each tour crosses each edge once in each direction) times the weight of the edge and summing over all edges provides a lower bound on the cost of any feasible solution.

The algorithm of Asano et al. [10] proceeds in a sequence of rounds. In each round, a set of tours is identified such that the *cost-to-savings ratio*, namely the ratio of the cost (of these tours) to the reduction to the lower bound that results from taking these tours, is bounded by some constant α. The key is that although a given tour itself may cost more than α times its reduction to the lower bound, collectively the set of tours has the desired ratio. If after a round ends, no uncovered demand remains, then the union of these sets of tours is a feasible solution with cost at most α times the lower bound, which is at most α times the optimal cost. For Asano et al. [10], $\alpha = (\sqrt{41}-1)/4$. We use a similar approach to achieve $\alpha = 4/3$.

Asano et al. [10] also introduced the idea of making *safe* modifications to the instance. That is, modifying the structure of the instance in such a way that does not increase the value of the

lower bound or decrease the optimum cost (although it may increase the optimum cost) and such that a feasible solution in the modified instance has a corresponding feasible solution in the original instance. These modifications can be made safely at any point in the algorithm. Our algorithm also makes use of safe modifications, although the ones that we define differ somewhat from those defined by Asano et al. [10].

These modifications allow us to reason better about how the resulting instance must be structured. The idea is that the modifications can be made until one of a few cases arise. Each case has a corresponding *strategy* to find a set of tours with the desired cost-to-savings ratio.

Specifically, the algorithm of Asano et al. [10] classifies the *leafmost* (minimal) subtrees containing at least $2Q$ units of demand into one of a few cases. The main obstacle in extending their algorithm to a 4/3-approximation is that one of their cases does not seem to have a good strategy. A natural idea would be to try modifying their algorithm to instead classify the leafmost subtrees containing at least βQ units of demand for some $\beta > 2$. But this can greatly increase the number of cases that arise.

We overcome this obstacle by generalizing the difficult case into what we call a *p-chain* (See Figure 7.3). Our key insight is that even arbitrarily large p-chains can be addressed efficiently in sibling pairs and at the root. Our algorithm effectively delays addressing the difficult case by pushing it rootward until it finds a pair or reaches the root and is thus easy to address. To keep the number of cases small, we address easy cases as they emerge, and require that any remaining difficult case must have a specific structure that the algorithm can easily detect in subsequent rounds.

7.3 Preliminaries

When the input graph is a tree T it is assumed to be rooted at the depot r. Recall that the problem gives a length $\ell(e) \geq 0$ for each edge $e \in E_T$, and we define the length of a subgraph $A \subseteq E$ to be $\ell(A) = \sum_{(u,v)\in A} \ell((u,v))$. In particular, $\ell(P[u,v]) = \sum_{e\in P[u,v]} \ell(e)$ denotes the the shortest-path distance $d_T(u,v)$ between u and v. Similarly, we define the demand of a subgraph $A \subseteq E$ to be $dem(A) = \sum_{v\in A} dem(v)$.

For rooted tree $T = (V, E)$, and $v \in V$ where $v \neq r$, recall that $p(v)$ denotes the parent of v, and v is the child of $p(v)$. For convenience, an edge labeled (u, v) indicates that u is the parent of v. Recall, the *subtree rooted at* v is the subgraph induced by the descendants of v and is denoted

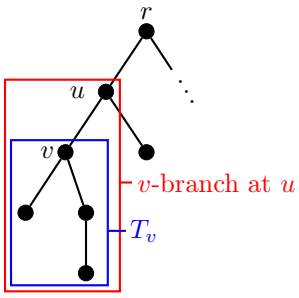


Figure 7.1: An example of T_v, the subtree rooted at node v, and a v-branch at u, which includes the stem edge (u, v). (Figure by Alice Paul).

T_v. If v has parent vertex u, we call $B = T_v \cup \{(u, v)\}$ the *v-branch* at u, and edge (u, v) the *stem* of the branch (see Figure 7.1).

We often extend the notion of parent and child to apply more colloquially to mean the immediately root-ward or leaf-ward instance of the given object. For example, the *parent edge* of edge $e = (u, v)$ is the edge $p(e) = (p(u), u)$. This is also the parent edge of vertex u and subtree T_u. The parent branch of a v-branch at u with stem e is the u-branch at $p(u)$ with stem $p(e)$ and is undefined if $u = r$. The *child edges* of a vertex u are all edges (u, v) such that $u = p(v)$. And so on.

7.3.1 Lower Bound

Here we describe the lower bound introduced by Hamaguchi and Katoh [59] and Asano et al. [10] and adopt similar terminology. Since all demand must be covered and each tour can cover at most Q demand, each edge $e = (u, v)$ must be traversed by enough tours to cover $dem(T_v)$ demand. We call this value the *traffic* on the edge e, denoted $\psi(e)$. Namely, $\psi(e) = \lceil \frac{dem(T_v)}{Q} \rceil$. Each of these tours traverses the edge exactly twice (once in each direction). We say that the lower bound $LB(e)$ of the contribution of edge $e = (u, v)$ to the total solution cost is therefore,

$$LB(e) = 2 \cdot \ell(e) \cdot \psi(e) = 2 \cdot \ell(e) \lceil \frac{dem(T_v)}{Q} \rceil$$

and that the lower bound LB on $cost(OPT)$ is

$$LB = \sum_{e \in E} LB(e)$$

For convenience, we scale down all demand values by a factor of Q and set $Q = 1$. We also assume that the vertices with positive demand are exactly the leaves: If some internal vertex v has positive demand $dem(v)$ we can add a vertex v' with demand $dem(v') = dem(v)$ and edge (v, v') of length zero and set $dem(v)$ to zero. Alternatively, if some leaf v has zero demand, no tour in an optimal solution will visit v, so v and the edge to v's parent can be deleted from the graph. Finally, we assume that no non-root vertex has degree exactly two, as no branching would occur at such a vertex, so the two incident edges can be spliced into one.

A very high level description of the algorithm is as follows: iteratively identify sets of tours in which the ratio of the cost of the tours to the reduction in cost to the lower bound, LB, is at most 4/3. We call such a set of tours a 4/3-*approximate tour set*.

We say that a 4/3-approximate tour set *removes* the demand that the tours cover. If such a tour set removes all demand from a branch, we say that this branch has been *resolved*. After a branch is resolved, it is convenient to think of it as having been deleted from the tree and proceed with the smaller instance.

We note that any 4/3-approximation algorithm for Capacitated Vehicle Routing trivially generates a 4/3-approximate tour set. The converse is also straightforward:

Lemma 36. *If, after iteratively finding and removing demand from* 4/3*-approximate tour sets, no demand remains, then the union of all tour sets is a* 4/3*-approximation for* Capacitated Vehicle Routing.

Given this, we make one more simplifying assumption that each leaf v has demand $dem(v) < 1$. Assume to the contrary that for some leaf v, $dem(v) \geq 1$. A tour that goes directly from r to v and back and covers one unit of demand at v is in fact a 1-approximate tour (and thus also a 4/3-approximate tour). If ever such a leaf exists, we can greedily take such tours until no more such leaves exist [10].

7.3.2 Example Showing Tightness

In this section we present the example of a tree instance that has cost at least $4/3 \cdot LB$ that was given by Asano et al. [10].

They define a tree T with root r, in which r has a single child q, and q has $2n + 1$ children $v_1, v_2, ..., v_{2n+1}$ that are all leaves. Edge (r, q) as well as edges (q, v_i) for all $1 \leq i \leq 2n + 1$ have

length one. That is, T is a tree of height two in which all edges have unit lengths. Furthermore, the demand of each leaf v_i is $0.5 + \epsilon$, where $\epsilon < \frac{1}{4n+2}$.

For this instance, the traffic of every edge (q, v_i) is one, and the traffic of edge (r, q) is $n + 1$, so $LB = 2 \cdot 1 \cdot (n + 1) + (2n + 1) \cdot 2 \cdot 1 \cdot 1 = 2n + 2 + 4n + 2 = 6n + 4$.

An optimum solution for this instance consists of $2n + 1$ tours $t_1, t_2, ..., t_{2n+1}$ in which tour t_i goes from r to v_i, covers all demand at v_i and then immediately returns to the depot.

The cost of this optimum solution, OPT is $cost(OPT) = (2n + 1) \cdot 4 = 8n + 4$.

The ratio of $cost(OPT)$ to LB is $\frac{8n+4}{6n+4}$ which tends to $4/3$ as n goes to infinity [10].

Note that there are in fact many other optimum solutions for this instance. For example consider the following solution consisting of $n + 1$ tours $t_1, t_2, ..., t_{n+1}$. For $i \in \{1, 2, ..., n\}$, t_i goes from r to v_{2i-1}, covers all demand at v_{2i-1}, then covers as much demand at v_{2i} as possible before returning to r. Tour t_{n+1} then covers all demand at v_{2n+1} and then all remaining demand at leaves v_{2i} for $i \in \{1, 2, ..., n\}$. The cost of this solution is $2 \cdot n \cdot 3 + 2 \cdot 1 \cdot (n + 2) = 6n + 2n + 4 = 8n + 4$.

7.3.3 Safe Operations

We say that an operation that modifies the graph is *safe* if it preserves solution feasibility and does not decrease $cost(OPT)$ or change the cost of the lower bound LB. Note that a 4/3-approximate tour set in the modified graph is therefore also a 4/3-approximate tour set in the unmodified graph.

The algorithm proceeds iteratively. In each iteration, the algorithm performs a series of *safe* operations and identifies a 4/3-approximate tour set. We now define these the operations.

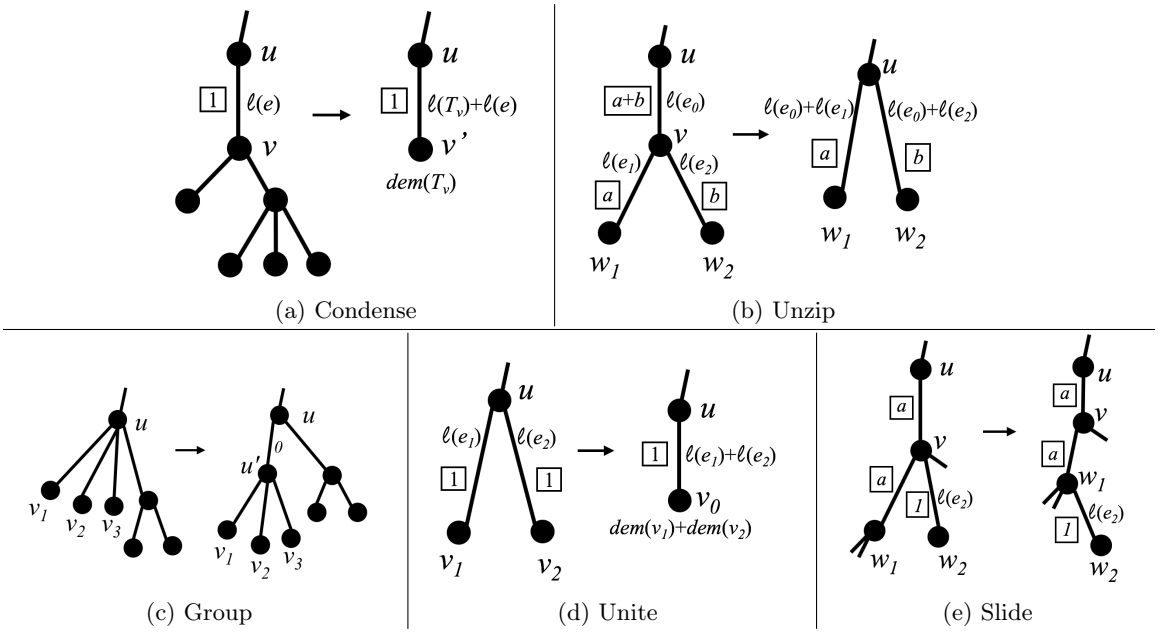
- **Condense**: If edge $e = (u, v)$ has traffic $\psi(e) = 1$ and v is not a leaf, add a vertex v' and replace e and T_v with an edge $e' = (u, v')$ with $\ell(e') = \ell(e) + \sum_{e'' \in T_v} \ell(e'')$ and set $dem(v') = dem(T_v)$ (see Figure 7.2a).

- **Unzip**: If edge $e = (u, v)$ has traffic equal to the sum of the traffic on child edges (v, w_1), (v, w_2), ..., (v, w_k), then delete v and add edges $(u, w_1), (u, w_2), ..., (u, w_k)$ with lengths $\ell((u, w_i)) = \ell(e) + \ell((v, w_i))$ for all $i \in \{1, 2, ..., k\}$ (see Figure 7.2b).

- **Group**: If vertex u has at least four children, including three leaf children v_1, v_2, v_3, such that $1.5 < dem(v_1) + dem(v_2) + dem(v_3) < 2$, add vertex u', edge (u, u') of length zero, and for $i \in 1, 2, 3$, replace (u, v_i) with (u', v_i) (see Figure 7.2c).

- **Unite**: If vertex u has leaf children v_1 and v_2 such that $dem(v_1) + dem(v_2) \leq 1$, delete v_1 and v_2 and add vertex v_0 with demand $dem(v_1) + dem(v_2)$ and edge (u, v_0) with length $\ell((u, v_1)) + \ell((u, v_2))$ (see Figure 7.2d).

- **Slide**: If edge $e_0 = (u, v)$ has child edges $e_1 = (v, w_1)$ and $e_2 = (v, w_2)$ such that traffic $\psi(e_0) = \psi(e_1)$, then delete edge e_2 and add edge (w_1, w_2) of length $\ell(e_2)$ (see Figure 7.2e).



(a) Condense (b) Unzip

(c) Group (d) Unite (e) Slide

Figure 7.2: Safe Operations. Traffic values are shown in rectangles.

Showing That Safe Operations are Safe

In this section we show that safe operations are actually safe. Recall that an operation that modifies the graph is *safe* if it does not decrease $cost(OPT)$ or change the cost of the lower bound LB and it preserves feasibility.

The **condense** operation replaces a branch B in which every edge e has traffic $\psi(e) = 1$ with a single edge e_B of length $\sum_{e \in B} \ell(e)$ and traffic $\psi(e_B) = 1$, so the lower bound LB remains unchanged, since the branch contributes $2 \cdot \sum_{e \in B} \ell(e) \cdot 1 = 2 \cdot \ell(e_B) \cdot 1$ toward LB both before and after the operation. The operation is equivalent to requiring any tour that covers *any* client demand in B to traverse *all of* B, which clearly preserves feasibility and possibly *increases*, but cannot decrease, $cost(OPT)$.

Before the **unzip** operation, the contribution to LB of edges involved in the modification is $2 \cdot \ell(e) \cdot \left(\sum_{i=1}^{k} \psi((v, w_i)) \right) + \sum_{i=1}^{k} 2 \cdot \ell((v, w_i)) \cdot \psi((v, w_i))$ which equals $\sum_{i=1}^{k} 2 \cdot (\ell(e) + \ell((v, w_i))) \cdot \psi((v, w_i))$, the contribution to LB of involved edges *after* the operation. The length and traffic values (and thus also the contribution to LB) of all other edges remain unchanged. The operation is equivalent to requiring every tour that traverses c child edges of e to traverse e a total of $2c$ times. This redundancy may increase the value of $cost(OPT)$ but cannot decrease it. Since any tour can be extended to cover e multiple times, the operation also preserves feasibility.

The **group** operation simply inserts an edge of length zero, resulting in an equivalent instance, and clearly has no effect on LB, $cost(OPT)$, or feasibility. The operation is merely for convenience.

The **unite** operation is equivalent to first inserting an edge of length zero (and traffic one) and then performing a condense operation, both of which are shown above to be safe. The composition of safe operations results in a safe operation.

The **slide** operation essentially *moves* an edge without changing the length or traffic values of any edge, which clearly preserves the value of LB. The operation is equivalent to requiring every tour that traverses edge e_2 to *also* traverse edge e_1, which may increase $cost(OPT)$, but cannot decrease $cost(OPT)$. Since the tree is connected, any tour can be extended to include e_1, so feasibility is preserved.

Reforming Operations from Asano et al. [10]

Asano et al. introduce seven safe *Reforming Operations* [10]. In this section we compare these to our safe operations.

Reforming operation R_1 greedily takes a tour at full capacity to any vertex with demand greater than one. After exhaustively applying R_1, no vertex has demand greater than one. Reforming operation R_2 moves any demand at internal nodes to leaves, by adding edges of length zero when necessary.

In our algorithm, we use simplifying assumptions to accomplish the same result as exhaustive application of these two operations. That is, we assume that the input graph already has the property that leaves are the only vertices with demand, and that the demand of every leaf is at most one. Our algorithm does not need these operations because these properties of the demand are never *undone* by our algorithm.

Reforming operation R_3 addresses the specific case of a vertex u with a leaf vertex v of demand

at most one as its only child, by contracting the edge (u, v), increasing $\ell((p(u), u))$ by $\ell((u, v))$, and setting $dem(u) = dem(v)$. Reforming operation R_4 similarly contracts an entire subtree with total demand at most one to a single leaf. Our **condense** operation generalizes both R_3 and R_4 as it operates on branches rather than subtrees. Note that R_3 also addresses the case where $1 < dem(u) + dem(v) \leq 2$, but this case never arises in our algorithm because of our simplifying assumptions.

Reforming operation R_5 merges leaves if their combined demand is at most one *and* if the parent vertex u of the leaves has no other child v such that the subtree rooted at v has total demand at least two. Our **unite** operation generalizes R_5.

Reforming operation R_6 addresses the case where a vertex v has exactly two children v_1 and v_2 that are leaves such that $1 < dem(v_1) + dem(v_2) < 2$. If v has parent u, R_6 removes v and adds edge (u, v_1) of weight $\ell((u, v)) + \ell((v, v_1))$ and edge (u, v_2) of weight $\ell((u, v)) + \ell((v, v_2))$. Our **unzip** operation generalizes R_6 to accommodate more than two children, non-leaf children, and larger demands.

Finally, reforming operation R_7 addresses the case where a vertex v has children v_1 and v_2 such that v_2 is a leaf and the subtree rooted at v_1 has total demand d_1 such that $1 < d_1 < 2$ *and* such that the subtree rooted at v has total demand d such that $1 < d < 2$. R_7 replaces edge (v, v_2) with edge (v_1, v_2) of length $\ell((v, v_2))$. Our **slide** operation generalizes R_7 to accommodate larger demand values.

7.4 Algorithm Description

Exhaustively applying safe operations is called *flattening* the instance. Note that none of the operations cancel each other, so this process terminates.

We say that a problem instance is *flattened* if no more safe operations are available, no internal vertices have demand, no non-root vertex has degree two, and for every leaf v, $0 < dem(v) < 1$. We say that a branch is flattened if these conditions hold for the branch.

Recall that a *branch* consists of a subtree along with its parent edge (*stem*). If the branch has traffic p, we call it a *p-branch*.

A flattened 2-branch with stem $e_0 = (u, v)$ such that v has exactly three children w_1, w_2, w_3, all of which are leaves and such that $1.5 < dem(w_1) + dem(w_2) + dem(w_3) \leq 2$ is called a 2-*chain*.

Lemma 37. *In a flattened problem instance, all 2-branches are 2-chains.*

Proof. Consider any 2-branch in a flattened problem instance. Clearly no edge in the branch can have traffic greater than two. If more than one child edge of the stem had traffic two, then the traffic of the stem itself must be greater than two. If the stem had exactly one child edge with traffic two, then a slide operation would have been possible. Therefore every child edge of the stem has traffic one. Each of these edges must be leaf edges or else they could be condensed. If there were exactly two such edges, then the stem could be unzipped. Since no two of these edges can be united, then the demand of every pair sums to more than one, so there are exactly three such edges and their demand sums to more than 1.5. □

7.4.1 p-Chains

We now generalize the notion of a 2-chain. For $p \geq 3$ a *p-chain* is a flattened p-branch with stem $e_0 = (u, v)$ such that v has exactly three children w_1, w_2, w_3, in which w_2 and w_3 are leaves with $1 < dem(w_2) + dem(w_3) \leq 1.5$ and (v, w_1) is the stem of a $(p-1)$-chain (see Figure 7.3a).

For convenience, we define a labeling scheme for p-chains. Each vertex v_i^j is doubly indexed by level i and rank (child-order) j. We use e_i^j to denote the parent edge of v_i^j, $\forall i, j$. A 2-chain with parent u has stem $e_2^0 = (u, v_2^0)$ leaves v_1^0, v_1^1, and v_1^2 such that $\ell(e_1^0) \geq \ell(e_1^1) \geq l(e_1^2)$. For $p > 2$, a p-chain with parent u has stem $e_p^0 = (u, v_p^0)$, and children v_{p-1}^0, v_{p-1}^1, and v_{p-1}^2, such that e_{p-1}^0 is the stem of a $p-1$-chain, and v_{p-1}^1 and v_{p-1}^2 are leaves with $\ell(e_{p-1}^1) \geq \ell(e_{p-1}^2)$ (See Figure 7.3a).

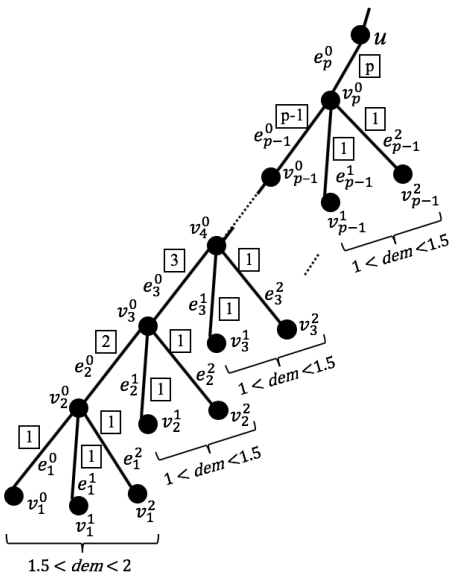
We further classify some p-chains as *long* p-chains: All 2-chains are long, and for $p \geq 3$ a *long p-chain* is a p-chain in which $\ell(e_{p-1}^2) < \ell(P[v_p^0, r])$ and e_{p-1}^0 is the stem of a *long* $(p-1)$-chain. A p-chain in which $\ell(e_{p-1}^2) \geq \ell(P[v_p^0, r])$ is called a *short* p-chain.

Long p-chains are particularly convenient because they can be *resolved* individually at the root and in sibling pairs for internal vertices, as described in the following lemmas (which we prove in Section 7.6).

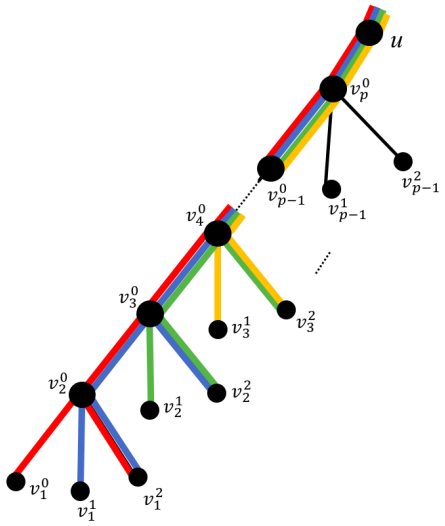
Lemma 38. *Long p-chains can be resolved in linear time at the root.*

Lemma 39. *A long p-chain and long p'-chain can be resolved together in linear time if they are sibling branches.*

As in [10], our algorithm proceeds in a series of iterations. Each iteration performs a set of safe operations and identifies a 4/3-approximate tour set.



(a) p-Chain. Traffic values are shown in rectangles, and *dem* denotes demand.



(b) Cascade. The first four tours of a cascade tour set are depicted.

Figure 7.3

Lemma 40. *Iteration i runs in linear time and either finds a nonempty 4/3-approximate tour set or finds that every branch at the root is either a long p-chain or 1-branch.*

Proof. See Section 7.5. □

These iterations continue until every branch at the root is either a long p-chain or 1-branch. The algorithm then solves the remaining instance, using the following lemma.

Lemma 41. *There is a linear time 4/3-approximation algorithm for instances of* Capacitated Vehicle Routing *on trees in which every child branch of the root is either a long p-chain or a 1-branch.*

Proof. The cost of a tour that traverses a 1-branch at the root is equivalent to the reduction to lower bound LB that results from removing the demand of the branch, so such a tour is a 1-approximate tour (and thus also a 4/3-approximate tour). The lemma result then follows from Lemma 36 and Lemma 38 □

Finally, putting these steps together, gives our overall 4/3-approximation result described in Theorem 24, which we restate here.

Theorem 24. *There is a 4/3-approximation algorithm for* Capacitated Vehicle Routing *in trees that runs in $O(n^2)$ time and is tight with respect to LB.*

Proof. Let m denote the total amount of demand in the graph. Since each iteration removes demand, there are at most m iterations, each of which runs in linear time, for an overall $O(n^2)$ runtime. By Lemma 40, after iteration m, every branch off the root must be a long p-chain or a 1-branch. By Lemma 41 there is a linear-time 4/3-approximation for these branches. Combining this approximation with the collection of tours identified during the iterations results in an overall 4/3 approximation, by Lemma 36. □

All that remains is to prove the above lemmas. In Section 7.5 we describe the iteration subroutine and prove Lemma 40. In Section 7.6 we prove Lemmas 38 and 39.

7.5 Description of Iteration i

We say that a p-branch B is *settled* if it is either a 1-branch or a long p-chain. Otherwise we say that B is *unsettled*. We say that a p-branch B is *minimally unsettled* if it is unsettled and all of its child branches are settled.

Each iteration consists of the following subroutine:

1. Flatten the instance (i.e. exhaustively apply safe operations)
2. If all branches at the root are settled, terminate the iteration.
3. Otherwise, find a minimally unsettled branch B.
   (i) If B has at least two child branches that are long p-chains, apply Lemma 39.
   (ii) Otherwise, if B has at least three child branches that are 1-branches, apply Lemma 42
   (iii) Otherwise, B is a (short) p-chain. Apply Lemma 43.

Lemma 42. *A flattened, minimally unsettled branch B with at least three child branches that are 1-branches admits a 4/3-approximate tour set.*

Proof. Let (u, v_0) be the stem of B, and let (v_0, v_1), (v_0, v_2), and (v_0, v_3) be three (child) 1-branches. Since the branch is flattened, v_1, v_2, and v_3 are leaves. Since the unite operation is unavailable, $dem(v_1) + dem(v_2) + dem(v_3) > 1.5$, and since B is unsettled, it is not a 2-chain. Furthermore,

the group operation is unavailable, so $2 < dem(v_1) + dem(v_2) + dem(v_3) < 3$. Let $a = \ell(P[v_0, r])$, $w_1 = \ell(v_0, v_1)$, $w_2 = \ell(v_0, v_2)$, and $w_3 = \ell(v_0, v_3)$. Without loss of generality, assume $w_1 \leq w_2 \leq w_3$.

If $a \leq w_1 + w_2 + w_3$, then for $i \in \{1, 2, 3\}$, let t_i be the tour that travels from the depot to v_i, covers all demand at v_i, and then returns to the depot. The length of t_i is $2(a + w_i)$. The total cost of the tour set $\{t_1, t_2, t_3\}$ is $2(3a + w_1 + w_2 + w_3)$. This tour set covers all demand of $\{v_1, v_2, v_3\}$ and also reduces demand along $P[v_0, r]$ by two, since $2 < dem(v_1) + dem(v_2) + dem(v_3) < 3$, so the reduction to LB is $2(2a + w_1 + w_2 + w_3)$. The ratio of the cost of the tour set to the reduction to the cost of LB that results from taking these tours is therefore,

$$\frac{2(3a + w_1 + w_2 + w_3)}{2(2a + w_1 + w_2 + w_3)} = 1 + \frac{a}{2a + w_1 + w_2 + w_3} \leq 1 + \frac{a}{3a} = \frac{4}{3}$$

where the inequality comes from $a \leq w_1 + w_2 + w_3$.

Otherwise, $a > w_1 + w_2 + w_3$. Let t be the tour that travels from the depot to v_1, covers all demand at v_1, travels to v_3, covers as much demand as possible at v_3, and then returns to the depot. The cost of t is $2(a + w_1 + w_3)$. Note that since the unite operation is unavailable, then $dem(v_1) + dem(v_3) > 1$, so the vehicle is full, and some demand remains at v_3. Since the vehicle is full, then t reduces demand along $P[v_0, r]$ by one, so the reduction to LB is $2(w_1 + a)$. The ratio of the cost of the tour set $\{t\}$ to the reduction to the cost of LB that results from taking tour t is therefore,

$$\frac{2(a + w_1 + w_3)}{2(w_1 + a)} = 1 + \frac{w_3}{w_1 + a} \leq \frac{4}{3}$$

where the final inequality comes from $a > w_1 + w_2 + w_3 \geq 3w_3$. □

Lemma 43. *A short p-chain admits a 4/3-approximate tour set.*

Proof. Let $a = \ell(P[v_p^0, r])$, $b = \ell(e_{p-1}^2)$, and $c = \ell(e_{p-1}^1)$. By construction, $c \geq b$, and since the p-chain is short, $b \geq a$. Let t_1 be the tour that goes from the depot to v_{p-1}^1, covers all demand at this leaf, and then returns to the depot. Similarly, let t_2 be the tour that goes from the depot to v_{p-1}^2, covers all demand at this leaf, and then returns to the depot. Tour t_1 has cost $2(a + c)$ and t_2 has cost $2(a + b)$. Tour set $\{t_1, t_2\}$ has total cost $2(2a + b + c)$. This tour set covers all demand at these leaves, which sum to greater than one by definition of p-chain, so these tours reduce the demand along $P[v_p^0, r]$ by one. Therefore the reduction to LB from this tour set is $2(a + b + c)$. Therefore the ratio of the cost of the tour set to the reduction to the cost of LB that results from

taking these tours is,

$$\frac{2(2a+b+c)}{2(a+b+c)} = 1 + \frac{a}{a+b+c} \leq \frac{4}{3}$$

The final inequality comes from $a \leq b \leq c$. □

Finally, we prove Lemma 40, which we restate here for convenience.

Lemma 40. *Iteration i runs in linear time and either finds a nonempty 4/3-approximate tour set or finds that every branch at the root is either a long p-chain or 1-branch.*

Proof. To prove this lemma, we first must show that any minimally unsettled branch B in a flattened instance must be in at least one of the three identified cases. Let B be such a branch. B must have child branches, since it is unsettled. Since it is minimally unsettled, every child branch is settled (i.e. either a 1-branch or a long p-chain).

If B has at least two long p-chains as child branches, then case i holds.

If B has no long p-chains as child branches, then all child branches are 1-branches. By Lemma 37, in a flattened instance all 2-branches are 2-chains. Since all 2-chains are long, and thus settled, B must be a j-branch for some $j \geq 3$. Since the unzip operation is unavailable, then there are at least four 1-branches as child branches of B, so case ii holds.

If B has exactly one long p-chain as a child branch and case ii doesn't hold, then B must be a short p'-chain. If there were no 1-branches as child branches, then the degree-two vertex could be spliced. If there were exactly one 1-branch then either $p = p'$ and a slide operation would be available, or $p = p' - 1$ and an unzip operation would be available. Since case ii doesn't hold, there are exactly two 1-branches as child branches. Since an unzip operation is unavailable, $p = p' - 1$, and since a unite operation is unavailable, the demand of the two 1-branches (which are both leaves by the condense operation) sum to greater than one. Since the instance is flattened, then B is a p'-chain. But since B is unsettled, it must be a short p'-chain. Therefore case iii holds.

Since B is covered by some case, then the corresponding lemma (Lemma 39, 42, or 43) guarantees a nonempty 4/3-approximate tour set.

If no unsettled branch can be found, then all branches at the root must be settled.

Finally, to see that the iteration runs in linear time, note that each operation can be performed in constant time. Condense, unite, unzip, (and splicing) all make the graph smaller, so they are exhausted in linear time. Subsequently, if condense and unite are unavailable, then each slide

results in a degree two vertex remaining to be spliced, also making the graph smaller. After all other operations are exhausted, each leaf can participate in at most one group operation in a given iteration. Each of the three cases can likewise be handled in linear time. □

7.6 Resolving p-Chains

7.6.1 Cascades

In order to prove Lemmas 38 and 39 we describe a specific group of tours that collectively covers the demand of a long p-chain.

The labeling on p-chains gives a natural bottom-up ordering on the leaves: $v_1^0, v_1^1, v_1^2, v_2^1, v_2^2$, ..., v_{p-1}^1, v_{p-1}^2, where the order is determined first by level and then by rank. For a long p-chain, we define a *cascade* to be the sequence of p tours in which each tour considers the leaves in this bottom-up order and:

1. Visits and covers all demand from the first leaf with remaining demand.
2. While the vehicle has spare capacity and there is unmet demand in the branch, determines the lowest level i with leaves with unmet demand and covers as much demand as possible from v_i^2.
3. Returns to the depot.

Let the resulting tours be $t_1, ..., t_p$. Tour t_1 first covers all demand from v_1^0 and then covers $1 - dem(v_1^0)$ demand from v_1^2. Since by definition, a p-chain is flattened, the unite operation is unavailable, so $dem(v_1^0) + dem(v_1^2) > 1$, implying the vehicle is full. The second tour t_2 covers all demand from v_1^1, covers all remaining demand at v_1^2 (since $dem(v_1^0) + dem(v_1^1) + dem(v_1^2) < 2$) and covers some demand at v_2^2. Since the slide operation is unavailable, $dem(v_1^0) + dem(v_1^1) + dem(v_1^2) + dem(v_2^2) > 3$, so the vehicle is full. Note that both vehicles have been full, and after two tours no demand remains at level 1. This pattern continues. For $3 \leq i < p$, tour t_i covers all demand at v_{i-1}^1, all remaining demand at v_{i-1}^2, and some demand at v_i^2. Inductively, since the slide operation is unavailable and all previous vehicles have been full, this vehicle must also be full. After t_i, no demand remains below level i. Since all vehicles have been full, there is less than one unit of demand remaining for t_p (and no demand remains below level $p-1$), so t_p covers all demand from v_{p-1}^1 and all remaining demand from v_{p-1}^2. (See Figure 7.3b).

For all i, this cascading pattern results in i tours traversing e_i^0, one tour traversing e_i^1, two tours traversing e_i^2 , and p tours traversing all edges in the path from the depot r to v_p^0. Note in particular that these values match the lower bounds given by the edge traffic, for e_i^0 and e_i^1. The excess cost, therefore, comes from doubling the traffic lower bound on the e_i^2 edges as well as extra traffic along the path to the depot.

7.6.2 Proofs of Lemmas 38 and 39

We restate the lemmas here for convenience. Recall that a branch is *resolved* if all of its demand is covered by a 4/3-approximate tour group. That is, a set of tours whose total cost is at most 4/3 times the reduction to the lower bound LB that results from removing the demand of the branch.

Lemma 38. *Long p-chains can be resolved in linear time at the root.*

Proof. Let B be a long p-chain at the root (depot). Consider the cascade on B. As described in Section 7.6.1, the cost of the p tours in the cascade is

$$2\big(p \cdot \ell(e_p^0) + \sum_{i=1}^{p-1} i \cdot \ell(e_i^0) + 2\ell(e_i^2) + \ell(e_i^1)\big)$$

since the cost of the path to the depot is zero. Additionally the cascade covers all demand in B, so the reduction to LB after covering B is:

$$\sum_{e \in B} LB(e) = \sum_{e=(u,v)\in B} 2 \cdot \ell(e) \cdot \lceil dem(T_v) \rceil = 2\big(p \cdot \ell(e_p^0) + \sum_{i=1}^{p-1} i \cdot \ell(e_i^0) + \ell(e_i^2) + \ell(e_i^1)\big)$$

Taking the ratio of cost to reduction in cost of LB gives our result:

$$\frac{2\big(p \cdot \ell(e_p^0) + \sum_{i=1}^{p-1} i \cdot \ell(e_i^0) + 2\ell(e_i^2) + \ell(e_i^1)\big)}{2\big(p \cdot \ell(e_p^0) + \sum_{i=1}^{p-1} i \cdot \ell(e_i^0) + \ell(e_i^2) + \ell(e_i^1)\big)} = 1 + \frac{\sum_{i=1}^{p-1} \ell(e_i^2)}{p \cdot \ell(e_p^0) + \sum_{i=1}^{p-1} i \cdot \ell(e_i^0) + \ell(e_i^2) + \ell(e_i^1)}$$

$$\leq 1 + \frac{\big(\ell(e_1^2)\big) + \big(\sum_{i=2}^{p-1} \ell(e_i^2)\big)}{\big(\ell(e_1^0) + l(e_1^1) + \ell(e_1^2)\big) + \big(\sum_{i=2}^{p-1} \ell(e_i^2) + \ell(e_i^1) + \sum_{j=i+1}^{p} \ell(e_j^0)\big)}$$

$$\leq 1 + \frac{\big(\ell(e_1^2)\big) + \big(\sum_{i=2}^{p-1} \ell(e_i^2)\big)}{\big(3 \cdot \ell(e_1^2)\big) + \big(\sum_{i=2}^{p-1} 3 \cdot \ell(e_i^2)\big)} \leq \frac{4}{3}$$

Where the second to last inequality comes from the fact that because B is a long p-chain, $\ell(e_i^2) < \ell(P[v_{i+1}^0, r]) = \sum_{j=i+1}^{p} \ell(e_j^0)$, and because $\ell(e_i^2) \leq \ell(e_i^1)$ by construction. □

Lemma 39. *A long p-chain and long p'-chain can be resolved together in linear time if they are sibling branches.*

Proof. Let u be a vertex with child branches B a long p-chain and B' a long p'-chain.

Consider the cascades on B and B'. The cost of these $p + p'$ tours is

$$2\big[\big(p\cdot\ell(e_p^0)+\sum_{i=1}^{p-1} i\cdot\ell(e_i^0)+2\ell(e_i^2)+\ell(e_i^1)\big)+\big(p'\cdot\ell(e'^0_{p'})+\sum_{i=1}^{p'-1} i\cdot\ell(e'^0_i)+2\ell(e'^2_i)+\ell(e'^1_i)\big)+(p+p')\ell(P[u,r])\big]$$

Additionally the cascade covers all demand in B and B', so the reduction to LB is

$$\sum_{e\in B\cup B'} LB(e) = \sum_{e\in B\cup B'} 2\cdot\ell(e)\cdot\psi(e)$$

which is

$$2\big[\big(p\cdot\ell(e_p^0)+\sum_{i=1}^{p-1} i\cdot\ell(e_i^0)+\ell(e_i^2)+\ell(e_i^1)\big)+\big(p'\cdot\ell(e'^0_{p'})+\sum_{i=1}^{p'-1} i\cdot\ell(e'^0_i)+\ell(e'^2_i)+\ell(e'^1_i)\big)+(p+p'-1)\ell(P[u,r])\big]$$

Note that the $p+p'-1$ factor in the reduction to the edges along $P[u,r]$ arises because by definition the total demand in a p-chain is between $p-0.5$ and p, so covering all demand in B and B' reduces the demand in T_u by at least $p+p'-1$. Rearranging the terms, this reduction to the lower bound is greater than:

$$2\big[\big(\sum_{i=0}^{2}\ell(e_1^i)+\sum_{i=2}^{p-1}\ell(P[v_{i+1}^0,r])+\ell(e_i^2)+\ell(e_i^1)\big)$$

$$+\big(\sum_{i=0}^{2}\ell(e'^i_1)+\sum_{i=2}^{p'-1}\ell(P[v'^0_{i+1},r])+\ell(e'^2_i)+\ell(e'^1_i)\big)+3\ell(P[u,r])\big]$$

$$= 2(X+Y+X'+Y'+Z)$$

Where

$$X = \sum_{i=0}^{2}\ell(e_1^i)$$

$$Y = \sum_{i=2}^{p-1}\ell(P[v_{i+1}^0,r])+\ell(e_i^2)+\ell(e_i^1)$$

$$X' = \sum_{i=0}^{2} \ell({e'}_1^i)$$

$$Y' = \sum_{i=2}^{p'-1} \ell(P[{v'}_{i+1}^0, r]) + \ell({e'}_i^2) + \ell({e'}_i^1)$$

and

$$Z = 3\ell(P[u, r]).$$

The ratio of cost to reduction in cost of LB is therefore at most,

$$1 + \frac{\sum_{i=1}^{p-1} \ell(e_i^2) + \sum_{i=1}^{p'-1} \ell({e'}_i^2) + \ell(P[u, r])}{X + Y + X' + Y' + Z} \leq \frac{4}{3}$$

Where the final inequality comes from noting that, by construction, $\ell(e_1^2) \leq X/3$ and $\ell({e'}_1^2) \leq X'/3$, and by definition of a long p-chain, $\ell(e_i^2) \leq \min\{\ell(e_i^1), \ell(e_i^2), \ell(P[v_{i+1}^0, r])\}$ for all $2 \leq i < p$, so $\sum_{i=2}^{p-1} \ell(e_i^2) \leq Y/3$ and $\sum_{i=2}^{p-1} \ell({e'}_i^2) \leq Y'/3$.

□

Chapter 8

An Approximation-Scheme Framework for Vehicle Routing in Trees

8.1 Introduction

In this chapter we move from Capacitated Vehicle Routing to consider a wider variety of vehicle-routing problems, including Minimum Makespan Vehicle Routing, Distance-Constrained Vehicle Routing, and School Bus Routing. We first introduce a general problem formulation that encompasses all of these problems. Recall that *vehicle-routing problems* are those in which a set of vehicles are collectively responsible for serving a set of clients. We are interested in the *rooted* versions of these problems, in which the vehicles originate at one or more depots.

We classify a family of rooted vehicle-routing problems by introducing the notions of *vehicle load*, the problem-specific vehicle constraint (e.g. number of clients per vehicle, distance traveled by the vehicle, client regret, etc.), and *fleet budget*, the problem-specific fleet constraint (e.g. number of vehicles, sum of distances traveled, etc.).

Most vehicle-routing problems can then be framed as either a Min-Max Vehicle Load problem: minimize the maximum vehicle load, given a bound k on fleet budget (e.g. Minimum Makespan Vehicle Routing) or a Minimum Fleet Budget problem: minimize the required fleet budget,

given a bound D on vehicle load (e.g. Distance-Constrained Vehicle Routing). In fact, these are two optimization perspectives of the same decision problem: does there exist a solution with maximum vehicle load D and fleet budget k?

In this chapter we present a framework for designing polynomial-time approximation schemes for Min-Max Vehicle Load and Minimum Fleet Budget in trees. As described in Chapter 1, tree (and treelike) transportation networks occur in building and warehouse layouts, mining and logging industries, and along rivers and coastlines [69, 74]. Our framework applies directly to Min-Max Vehicle Load problems and generates results of the following form.

Theorem 25. *For every $\epsilon > 0$, there is a polynomial-time algorithm that, given an instance of* Min-Max Vehicle Load *on a tree, finds a feasible solution whose maximum vehicle load is at most $1 + \epsilon$ times optimum.*

An immediate corollary of Theorem 25 is the following *bicriteria* result for the associated Minimum Fleet Budget problem.

Theorem 26. *Given an instance of* Minimum Fleet Budget *on a tree, if there exists a solution with fleet budget k and vehicle load D, then for any $\epsilon > 0$, there is a polynomial-time algorithm that finds a solution with fleet budget k and vehicle load at most $(1 + \epsilon)D$.*

The input to the framework is a tree $T = (V, E)$, edge lengths $\ell(u, v) \geq 0$ for all $(u, v) \in E$, and a specified depot vertex $r \in V$ at which all vehicles start. Without loss of generality, we assume the tree to be rooted at the depot r and that the set of clients corresponds to the set of leaves in the tree: any subtree without a client can be safely removed from the instance and clients at internal vertices can be moved to leaves by introducing edges of length zero. Since every edge must then be traversed by at least one vehicle, the problems are equivalent to corresponding tree-cover problems.

As stated, the framework can be customized to a wide range of problems. In Section 8.5, we illustrate in detail how to customize the framework to give a PTAS for the Minimum Makespan Vehicle Routing problem of finding k *tours* each starting and ending at a depot r that serve all clients in T such that the *makespan*, the maximum length of any tour, is minimized. Here, vehicle load is the tour length, and fleet budget is the number of vehicles. A bicriteria PTAS for the associated Minimum Fleet Budget problem, Distance-Constrained Vehicle Routing, follows as a corollary.

We will also demonstrate how the framework can be applied to give similar results for other vehicle-routing variants, including Capacitated Vehicle Routing (see Section 8.6.1) and School Bus Routing (see Section 8.6.2). Additionally, we show how to generalize to the multiple depot setting (see Section 8.6.3). We choose a range of problems that showcase the flexibility and power of our framework, though we believe the applicability extends beyond these problems as well.

At a high level, the framework partitions the tree into *clusters* such that there exists a near-optimal solution that within each cluster has a very simple form, effectively coarsening the solution space. Then, given this simplified structure, a dynamic program can be designed to find such a near-optimal solution.

The clusters are designed to be *small* enough so that simplifying vehicle routes at the cluster level does not increase the optimal load by too much, but also *large* enough that the (coarsened) solutions can be enumerated efficiently. To bound the error introduced by this simplification we design a load-reassignment tool that makes cluster coverage adjustments *globally* in the tree.

Finally, standard dynamic programming techniques can result in a large accumulation of rounding error. To limit the number of times that the load of any single route is rounded, we introduce a *route projection* technique that essentially pays in advance for load that the vehicle *anticipates* accumulating, allowing the dynamic program to round only once instead of many times for this projected load.

8.2 Related Work

For trees, Minimum Makespan Vehicle Routing is equivalent to Minimum Makespan Rooted Tree Cover: the minimum makespan for rooted tree cover is exactly half the minimum makespan for vehicle routing, since tours traverse edges twice. Minimum Makespan Rooted Tree Cover is NP-hard even on star instances but admits an FPTAS if the number, k, of subtrees is constant [89] and a PTAS for general k [61]. For covering a *general graph* with rooted subtrees, [42] provides a 4-approximation; this bound was later improved to a 3-approximation by [83]. For tree metrics, an FPTAS is known for constant k [96], and a $(2+\epsilon)$-approximation is known for general k [83]. Our result improves this to a PTAS. Although a recent paper of Chen and Marx [28] also claimed to present a PTAS, in Section 8.5.3 we show that their result is incorrect and cannot be salvaged

using the authors' proposed techniques[1]. Additionally, we compare their approach to our own and describe how we successfully overcome the challenges where their approach fell short.

In general metrics Minimum Makespan Vehicle Routing (also called Fleet-Constrained Vehicle Routing and Rooted Min-Max Tour Cover) admits a 6-approximation [82]. Of note, an α-approximation to Minimum Makespan Rooted Tree (or Path) Cover gives a 2α approximation to Minimum Makespan Vehicle Routing by *doubling* the trees to give a tour of twice the length (equivalent to depth-first search and returning to the depot).

The associated Distance-Constrained Vehicle Routing problem is to minimize the number of tours of length at most D required to cover all client demand. Even restricted to star metrics, this problem is NP-hard, and for tree instances it is hard to approximate to better than a factor of 3/2 [84]. Nagarajan and Ravi give a simple 2-approximation for trees [84]. For general metrics, they give an $O(\log D)$-approximation and show that an $O(\log |S|)$-approximation is straightforward by framing vehicle routing as a set-cover problem and using the Orienteering problem as a greedy subroutine [84]. Recently, the best known approximation ratio for general metrics was slightly improved to $O(\min\{cost(OPT), \log D/\log\log D\})$ [49]. Relaxing the distance constraint, Nagarajan and Ravi also gave an $(O(\log \frac{1}{\epsilon}), 1+\epsilon)$ bicriteria approximation for general metrics, that finds $O(\log \frac{1}{\epsilon})cost(OPT)$ tours of length at most $(1+\epsilon)D$ [84]. In this chapter we give a $(1, 1+\epsilon)$ bicriteria PTAS for trees, and note that the hardness results for trees described above [84] imply that without allowing this $(1+\epsilon)$ stretch in D, a PTAS is unlikely to exist.

The *regret* of a path is the difference between the path length and the distance between the path endpoints. The Min-Max Regret Routing problem is to cover all clients with k *paths* starting from the depot, such that the maximum regret is minimized. For trees, there is a known 13.5-approximation algorithm [21], which we improve to a PTAS in this thesis. For general graphs there is a $O(k^2)$-approximation algorithm [49].

In the related School Bus Routing problem, there is a bound R on the regret of each path and the goal is to find the minimum number of paths required to cover all client demand. For general graphs, [50] provides an LP-based 15-approximation algorithm, improving upon the authors' previous 28.86-approximation algorithm [49]. In trees, there exists a 3-approximation algorithm for the uncapacitated version of this problem and a 4-approximation algorithm for the capacitated version [21]. Additionally, there is a (3/2) inapproximability bound [21]. A PTAS is therefore

[1]We have contacted the authors, and they agree with our assessment.

unlikely to exist for trees. Instead, we give a $(1, 1+\epsilon)$ bicriteria PTAS which, as a corollary, improves the best-known approximation ratio to 2.

As we saw in Chapter 7, the best known approximation ratio for Capacitated Vehicle Routing in tree metrics is $4/3$ (as shown in this thesis) which improves over the previous approximation ratios of $(\sqrt{41}-1)/4$ [10] and 1.5 [59]. See Chapter 3 for related work on Capacitated Vehicle Routing in more general metrics. In this chapter we give a $(1, 1+\epsilon)$ bicriteria PTAS for Capacitated Vehicle Routing in trees with arbitrary capacities, in which capacities are allowed to be exceeded by a $(1+\epsilon)$ factor.

8.3 Preliminaries

Recall some of our notation for rooted trees (see Section 1.3 and Section 7.3 for a more comprehensive overview). We use $p(v)$ to denote the parent of vertex v and T_v to denote the subtree rooted at v. If $u = p(v)$, the v-*branch* at u consists of T_v (the subtree rooted at v) together with the *stem* edge (u, v). We define the *length* of a subgraph $A \subseteq E$ to be $\ell(A) = \sum_{(u,v)\in A} \ell(u, v)$. For vertices u, v, we use $d_T(u, v)$ to denote the shortest-path distance in T between u and v.

In this chapter, we assume every vertex has at most two children. If vertex v has $j > 2$ children $v_1, ..., v_j$, add vertex v' and edge (v, v') of length zero and replace edges (v, v_1),(v, v_2) with edges (v', v_1),(v', v_2) of the same lengths.

Our framework applies to vehicle-routing problems that can be framed as Min-Max Vehicle Load problems, in which the objective is to minimize the maximum vehicle load, subject to a fleet budget. Given a Min-Max Vehicle Load problem, a trivial n-approximation can be used to obtain an upper bound D_{high} for $cost(OPT)$. An overarching algorithm takes as input a load value $D \geq 0$ and provides the following guarantee: if there exists a solution with max load D, the algorithm will find a solution with max load at most $(1+\epsilon)D$. A PTAS follows from using binary search between $\frac{D_{high}}{n}$ and D_{high} for the smallest value D_{low} such that the algorithm returns a solution of max load at most $(1+\epsilon)D_{low}$. This implies $D_{low} \leq cost(OPT)$. For the rest of the chapter, we assume D is fixed.

8.4 Framework Overview

Optimization problems on trees are often well suited for dynamic programming algorithms. In fact, the following dynamic programming strategy can solve Min-Max Vehicle Load problems on trees exactly: at each vertex v, for each value $0 \leq i \leq D$, *guess* the number of solution route segments of load exactly i in the subtree rooted at v. Such an algorithm would be exponential in D. (See the DP for Capacitated Vehicle Routing in Section 3.3.2).

Instead of considering every possible load value, route segment loads can be *rounded* up to the nearest θD, for some value $\theta \in (0, 1]$ that depends only on ϵ, so that only $O(\theta^{-1})$ segment load values need to be considered. In order to achieve a PTAS, we must show that this rounding does not incur too much error. Rounding the load of a route at *every* vertex accumulates too much error, but if the number of times that any given route is rounded is at most ϵ/θ, then at most ϵD error accumulates, as desired.

One main insight underlying our algorithm is that a route only needs to incur rounding error when it branches. The challenge in bounding the rounding error then becomes bounding the number of times a route branches. While a route in the optimal solution may have an arbitrary amount of branching, we show that we can greatly limit the scope of candidate solutions to those with a specific structure while only incurring an ϵD error in the maximum load. Rather than having to make decisions for covering every leaf in the tree (of which there may be arbitrarily many−each with arbitrarily small load), we partition the tree into *clusters* and then address covering the clusters.

By *reassigning* small portions of routes within a cluster, we show that there exists a near-optimal solution in which all clients (leaves) within a given cluster are covered by only one or two vehicles. These clusters are chosen to be small enough that the error incurred by the reassignment is small, but large enough that any given route covers clients in a bounded number of clusters. This *coarsens* the solutions considered by the algorithm, as vehicles must commit to covering larger fractions of load at a time. A dynamic program then finds the optimal such coarse solution using these simple building blocks within each cluster.

8.4.1 Simplifying the Solution Structure

Let $\hat{\epsilon}$ and δ be problem-specific values that depend only on ϵ. Let $\mathcal{H}_{\mathcal{T}}$ denote the set of all subgraphs of T, and let $g : \mathcal{H} \to \mathbb{Z}^{\geq 0}$ be a problem-specific *load function*. We require g to be monotonic and

subadditive. Intuitively, for all $H \in \mathcal{H}_{\mathcal{T}}$, $g(H)$ is the load accumulated by a vehicle for covering H.

Condensing the Input Tree

The first step in the framework is to CONDENSE all small branches into leaf edges. Specifically, let $\mathcal{B}$ be the set of all maximal branches of load at most δD. That is, for every v-branch $b \in \mathcal{B}$, $g(b) \leq \delta D$ and for b's parent $p(v)$-branch, b_p, $g(b_p) > \delta D$. For convenience, if $b_1 \in \mathcal{B}$ is a v_1 branch at u and $b_2 \in \mathcal{B}$ is a *sibling* v_2 branch at u such that $g(b_1) + g(b_2) \leq \delta D$, we add a vertex u' and an edge (u, u') of length zero and replace (u, v_i) with edge (u', v_i) of length $\ell(u, v_i)$ for $i \in \{1, 2\}$. The u' branch at u then replaces the two branches b_1 and b_2 in $\mathcal{B}$. This ensures that any two branches in $\mathcal{B}$ with the same parent cannot be combined into a subtree of load $\leq \delta D$.

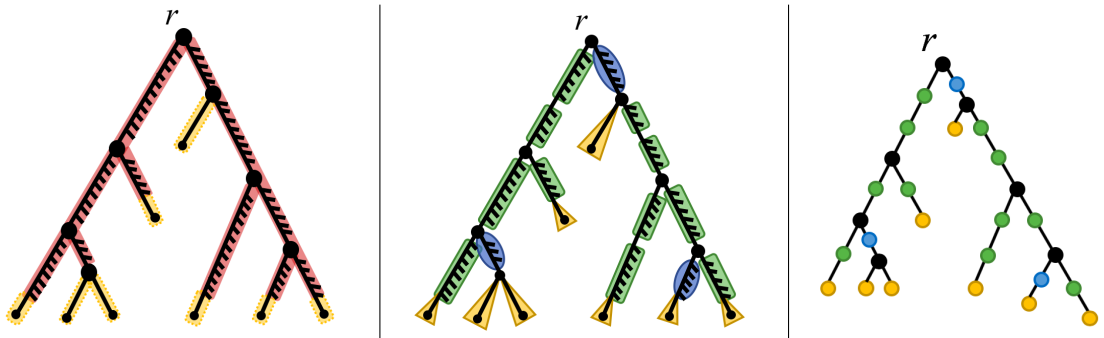
Then, for every $b \in \mathcal{B}$, we *condense* b by replacing it with a leaf edge of length $\ell(b)$ and load $g(b)$. All clients in b are now assumed to be co-located at the leaf. Though it is easier to think of these condensed branches as leaf edges, the algorithm need not actually modify the input tree; condensing a branch is equivalent to requiring the entire branch to be covered by a single vehicle.

Clustering the Condensed Tree

After condensing all small branches, we partition the condensed tree into clusters. We define every leaf edge whose load is at least $\frac{\delta}{2}D$ to be a *leaf cluster*. The leaf-cluster-to-root paths define what we call the *backbone* of T. By construction, every edge that is not on this backbone is either a leaf cluster (of load $\geq \frac{\delta}{2}D$) or a leaf edge (of load $< \frac{\delta}{2}D$). That is, every vertex is at most one edge away from the backbone (see Figure 8.1a).

We can think of the condensed tree as a binary tree whose root is the depot, whose leaves are the leaf clusters, and whose internal vertices are the branching points of the backbone. Each edge of this binary tree corresponds to a maximal path of the backbone between these vertices, together with the small leaf edges off of this path (see Figure 8.1a). To avoid confusion with tree edges, we call these path and leaf subgraphs *woolly edges*. A *woolly subedge* of a woolly edge consists of a subpath of the backbone and all incident leaf edges.

A woolly edge e whose load $g(e)$ is less than $\frac{\hat{\epsilon}\delta}{2}D$ is called a *small cluster*. The remaining woolly edges have load at least $\frac{\hat{\epsilon}\delta}{2}D$. We partition each such woolly edge into one or more woolly subedges, which we call *edge clusters*, each with load in $[\frac{\hat{\epsilon}\delta}{2}D, \frac{\delta}{2}D]$. Backbone edges do not contain clients and can be subdivided as needed to ensure enough granularity in the tree edge lengths so that such a



(a) Woolly edges (b) Clusters (c) T^*

Figure 8.1: (a) Leaf clusters in yellow and woolly edges in red; (b) The tree partitioned into leaf clusters (yellow triangles), small clusters (blue ovals), and edge clusters (green rectangles); (c) The corresponding T^* for clustering from (b).

partition is always possible (see Figure 8.1b).

For convenience, we label the components of edge clusters. Let $\mathcal{C}$ be the set of edge clusters. For any edge cluster $C \in \mathcal{C}$, let P_C denote the backbone path in C and let L_C denote the leaf edges in C. We order the backbone edges along P_C as $p_{C,1}, p_{C,2}, ..., p_{C,m}$ in increasing distance from the depot and similarly label the leaf edges $e_{C,1}, e_{C,2}, ..., e_{C,m-1}$ such that $e_{C,i}$ is the leaf incident to $p_{C,i}$ and $p_{C,i+1}$ for all $1 \leq i < m$ (see Figure 8.2). If no such incident leaf exists for some i, we can add a leaf of length zero. Likewise P_C can be *padded* with edges of length zero to ensure that each edge cluster 'starts' and 'ends' with a backbone edge.

Solution Structure

Consider the intersection of a solution with an edge cluster C. There are three different *types* of routes that visit C (see Figure 8.2). A *C-passing* route traverses C without covering any clients, and thus includes all of P_C but no leaf edges in L_C. A *C-collecting* route traverses *and* covers clients in C, and thus includes all of P_C and some edges in L_C. Last, a *C-ending* route covers clients in, but does not traverse C, and thus includes backbone edges $p_{C,1}, p_{C,2}, ..., p_{C,i}$ for some $i < m$ and some leaves in L_C, but does not include all of P_C. Note that any *C-ending* route can be assumed to cover some leaves in L_C, because otherwise removing any such redundancy would only improve a solution.

We say that a cluster C has *single coverage* if a single vehicle covers *all* clients in C. We say that an edge cluster C has *split coverage* if there is one *C-ending* route that covers leaf edges $e_{C,1}, e_{C,2}, ..., e_{C,i}$ for some $i < m - 1$ and one *C-collecting* route that covers leaf edges

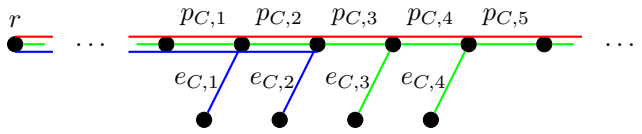


Figure 8.2: Three types of route within an edge cluster C; the red (top) tour is a C-passing route, the green (middle) tour is a C-collecting route, and the blue (bottom) tour is a C-ending route. (Figure by Alice Paul).

$e_{C,i+1}, e_{C,i+2}, ..., e_{C,m-1}$ (see Figure 8.2).

Finally, we say that a feasible solution has a *simple structure* if:

- Leaf clusters and small clusters have single coverage,
- Edge clusters have single or split coverage, and
- Each vehicle covers clients in $O(\frac{1}{\hat{\epsilon}^2\delta})$ clusters.

Customization of the framework must include proving a *structure theorem* stating that there exists a near-optimal solution (i.e. a feasible solution with maximum load at most $(1+\epsilon)D$) with simple structure. Such a theorem proves that it is safe to reduce the set of potential solutions to those with simple structure.

8.4.2 Dynamic Program

After proving a structure theorem, the framework uses a dynamic programming algorithm (DP) to actually find a near-optimal solution with simple structure. We define the *cluster tree* T^* to be the tree that results from contracting each cluster of T to a single vertex. That is, the cluster tree has a vertex for each cluster and each branching point of the backbone (See Figure 8.1c). The DP traverses T^* starting at the leaves and moving rootward, enumerating the possible route structures within each cluster. Namely, the DP considers all ways edge cluster coverage can be split and how routes are merged at branching points.

At each vertex in this tree the algorithm stores a set of *configurations*. A configuration is interpreted as a set of routes in T that cover all clusters in the subtree of T^* rooted at v. Let $\theta \in (0, 1]$ be a problem-specific value that depends only on ϵ. A configuration at a vertex v specifies, for each multiple i of θD between 0 and $(1+\epsilon)D$ the number of routes whose rounded load is i at the time they reach v. Because θ depends only on ϵ, the number of configurations and runtime of

the DP is polynomially bounded. After traversing the entire cluster tree, the solution is found at the root. If there exists a configuration at the root such that all of the rounded route loads are at most $(1+\epsilon)D$, the algorithm returns this solution.

To ensure that the DP actually finds a near-optimal solution, the number of times that a given route is rounded must be bounded by ϵ/θ, which gives a rounding error of at most ϵD. In particular, we design the DP so that the number of times that any one route is rounded is proportional to the number of clusters that it covers clients in. Then, using the structure theorem, there exists a near-optimal solution that covers clients in $O(\frac{1}{\hat{\epsilon}^2\delta})$ clusters and gets rounded by the DP $O(\frac{1}{\hat{\epsilon}^2\delta})$ many times. Finally, θ is set to $c_\theta\epsilon\hat{\epsilon}^2\delta$ for some constant c_θ.

For loads involving distance, C-passing routes pose a particular challenge for bounding rounding error. These routes may accumulate load while passing through clusters without covering any clients, yet the DP cannot afford to update the load at every such cluster. Instead, the DP *projects* routes to predetermined destinations up the tree, so that they accumulate rounding error only once while passing many clusters. The configuration then stores the (rounded) loads of the *projected* routes, and the DP need not update these load values for clusters passed through along the projection.

8.4.3 Reassignment Lemma

We now present a lemma that will serve as a general-purpose tool for our framework. This tool is used to *reassign* small route segments. That is, if some subgraph H is covered by several small route segments from distinct vehicles $h_1, h_2, ..., h_m$, then for some $1 \leq i \leq m$, the entire subgraph H is assigned to be covered by h_i. This *increases* load on h_i so as to cover all of H, and *decreases* load on h_j for all $j \neq i$ which are no longer required to cover H (see Figure 8.4). We show that this assignment process can be performed simultaneously for many such subgraphs such that the net load increase of any one route is small.

Let $G = (A, B, E)$ be an edge-weighted bipartite graph where A is a set of *facilities*, B is a set of *clients*, and $w(a, b) \geq 0$ is the weight of edge $(a, b) \in E$. For any vertex v, we use $N(v)$ to denote the *neighborhood* of v, namely the set of vertices u such that there is an edge $(u, v) \in E$. Each facility $a \in A$ has capacity $q(a) = \sum_{b \in N(a)} w(a, b)$ and each client $b \in B$ has weight $w(b) \leq \sum_{a \in N(b)} w(a, b)$. A feasible *assignment* is a function $f : B \to A$, such that each client b is assigned to an adjacent facility $f(b) \in N(b)$. We can think of the weights $w(a, b)$ representing fractional assignment costs while weight $w(b)$ corresponds to a "discounted" cost of wholly serving client b. Ideally, the total

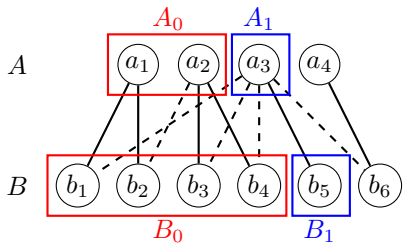


Figure 8.3: An example of the inductive subsets A_i and B_i. A solid edge (a, b) indicates that $f(b) = a$, and a dashed edge indicates that $(a, b) \in E$ but $f(b) \neq a$. Note that b_6's level is infinite. (Figure by Alice Paul).

weight of clients assigned to any facility a would not exceed the capacity $q(a)$; however, this is not always possible. We define the *overload* $h_f(a)$ of a facility a to be $w(f^{-1}(a)) - q(a) = \sum_{b|f(b)=a} w(b) - \sum_{b \in N(a)} w(a, b)$ and the *overload* h_f of an assignment to be $\max_{a \in A} h_f(a)$. The Bipartite Weight-Capacitated Assignment problem is to find an assignment with minimum overload.

In our application of Bipartite Weight-Capacitated Assignment, facilities represent vehicle routes and clients represent subgraphs of T. Assignment of a client b to a facility a represents assigning subgraph b to be covered by route a (see the proof of Lemma 45 for an example).

We now prove the following useful result for the problem.

Lemma 44. *Given an instance of the* Bipartite Weight-Capacitated Assignment *problem, an assignment with overload at most* $\max_{b \in B} w(b)$ *can be found efficiently.*

Proof. Let $w_{max} = \max_{b \in B} w(b)$. Consider an initial assignment f by arbitrarily assigning each client to an adjacent facility. Let $A_0 = \{a \in A | h_f(a) > w_{max}\}$ be the set of facilities whose capacities are exceeded by more than w_{max}. The lemma statement is satisfied if and only if $|A_0| = 0$.

Let $B_0 = f^{-1}(A_0)$ be the set of clients that are assigned to facilities in A_0. We inductively define A_i for $i \geq 1$ to be $N(B_{i-1}) \setminus \bigcup_{j<i} A_j$ and $B_i = f^{-1}(A_i)$ to be the set of clients that are assigned to facilities in A_i (see Figure 8.3). We say that a client b (resp. facility a) has *level* i if $b \in B_i$ (resp. $a \in A_i$), and we say that client b has infinite level if b does not appear in any B_i. By construction, each client appears in some B_i for at most one value i, so the level of a client is either infinite or at most $|B|$ (see Figure 8.3).

It suffices to show that if $|A_0| > 0$ then there is some client b, with some finite level, whose level can be increased without decreasing the level of any other client. After at most $|B|^2$ such improvements, B_0 must be empty, so $|A_0|$ must also be empty, proving the claim.

Suppose that there is some i and some facility $a \in A_i$ such that $h_f(a) \leq q(a)$. We say such a facility is *underloaded*, and therefore $i > 0$. By construction, there is some $b \in B_{i-1}$ adjacent to a and some $a' \in A_{i-1}$ such that $f(b) = a'$. We reassign b to a by setting $f(b) = a$. Note that since this adds $w(b) \leq w_{max}$ to the load of a, the resulting overload of a is at most w_{max}, so the level of a does not decrease. Further, b now has level at least i and the level of every other client is either unaffected or increased.

We now show that such an underloaded facility always exists. Let j be the largest value such that A_j is non-empty. If B_j is empty, then all facilities in A_j are underloaded. Otherwise B_j is non-empty and $N(\bigcup_{0\leq i\leq j} B_i) \subseteq \bigcup_{0\leq i\leq j} A_i$, so $\sum_{b\in\cup_i B_i} w(b) \leq \sum_{b\in\cup_i B_i} \sum_{a\in N(b)} w(a,b) \leq \sum_{a\in\cup_i A_i} q(a)$. Since no other clients are assigned to these facilities, at least one facility in $\bigcup_{0\leq i\leq j} A_i$ must be underloaded.

□

8.5 Customizing the Framework: Minimum Makespan Vehicle Routing

In this section, we demonstrate how to apply the general framework to a specific problem, Minimum Makespan Vehicle Routing. In particular, we use the framework to achieve the following:

Theorem 27. *For every $\epsilon > 0$, there is a polynomial-time algorithm that, given an instance of* Minimum Makespan Vehicle Routing *on a tree, finds a solution whose makespan is at most $1 + \epsilon$ times optimum.*

An identical result was claimed to be achieved in a recent paper by Chen and Marx [28]. In Section 8.5.3, we compare our approach to theirs and prove that their algorithm is incorrect.

Recall that the problem is to find k *tours* that serve all clients in T such that the maximum *length* of any tour is minimized. The vehicle routes are tours, and the vehicle load is tour length, so the load $g(H)$ of subgraph H is twice the length of edges in the subgraph. The condense operation is then applied to the input tree, with $\delta = \hat{\epsilon} = \epsilon/c$ for some constant c we will define later. Leaf clusters therefore correspond to branches of length at least $\frac{\hat{\epsilon}}{4}D$ (load at least $\frac{\hat{\epsilon}}{2}D$), small clusters have total length less than $\frac{\hat{\epsilon}^2}{4}D$, and edge clusters have total length in $[\frac{\hat{\epsilon}^2}{4}D, \frac{\hat{\epsilon}}{4}D]$ As described in Section 8.4, the two steps in applying the framework are proving a structure theorem and designing

a dynamic program.

8.5.1 Minimum Makespan Vehicle Routing Structure Theorem

We prove the following for Minimum Makespan Vehicle Routing.

Theorem 28. *If there exists a solution with makespan D, then there exists a solution with makespan at most $1 + O(\hat{\epsilon})D$ that has simple structure.*

We prove the above by starting with some optimal solution of makespan at most D and show that after a series of steps that transforms the solution into one with simple structure, the makespan is still near-optimal.

To ensure that each step maintains solution feasibility, we introduce the following notion of independence. Let T' be a connected subgraph of T containing the depot r, and let X be a set of subgraphs of T. We say that X is a *tour-independent* set with respect to T' if $T' \cup X'$ is connected for all $X' \subseteq X$. In particular, if T' is the subgraph covered by a single tour then adding any subgraphs in X' creates a new feasible tour.

Lemma 45. *The* condense *operation adds at most $\hat{\epsilon}D$ to the optimal makespan.*

Proof. The condense operation is equivalent to requiring every branch in $\mathcal{B}$ to be covered by a single tour. We show that there is such a solution of makespan at most $cost(OPT) + \hat{\epsilon}D$. Fix an optimal solution, and let A be the set of tours in the optimal solution that (at least partially) cover branches in $\mathcal{B}$. We define an edge-weighted bipartite graph $G = (A, \mathcal{B}, E)$ where there is an edge (a, b) if and only if tour a contains edges of branch b, and $w(a, b)$ is the length of the tour segment of a in branch b, namely twice the length of the edges covered by tour a. Note that $\forall a \in A, b \in \mathcal{B}$, $w(a, b) \leq \hat{\epsilon}D$. For each $b \in \mathcal{B}$, we define the weight $w(b)$ to be $2\ell(b)$, and for each $a \in A$, we define the capacity $q(a)$ to be the sum $\sum_{b:a\cap b\neq\emptyset} w(a, b)$ of all tour segments of a in branches of $\mathcal{B}$. Clearly, $w(b) \leq \sum_{a:a\cap b\neq\emptyset} w(a, b)$, since these tour segments collectively cover b.

Essentially, $q(a)$ represents tour a's *budget* for buying whole branches and is defined by the length of its tour segments in the branches that it partially covers. Further, we will only assign a branch to a tour that already covers some edges in the branch so there is no additional cost to connect the tour to the branch.

Applying Lemma 44 to G, we can achieve an assignment of branches to tours such that each branch is assigned to one tour and the capacity of each tour is exceeded by at most $\max_{b\in B} w(b) \leq$

$\hat{\epsilon}D$. Further, for any tour $a \in A$, let T'_a be the corresponding subgraph visited by a excluding any branches in B. T'_a contains r and is connected, so $N_G(a) \subseteq \mathcal{B}$ is a tour-independent set with respect to T'_a. Thus, the reassignment of branches creates a feasible solution in which the extra distance traveled by each tour is at most $\hat{\epsilon}D$. □

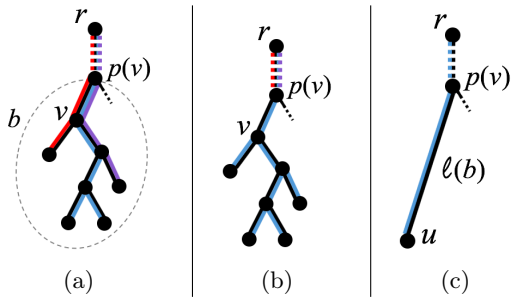


Figure 8.4: Figure (a) depicts a branch $b \in \mathcal{B}$ covered by several small tour segments; (b) shows the entire branch b being assigned to the blue tour; (c) shows the result of the condense operation.

Lemma 46. *Requiring all leaf clusters and small clusters to have single coverage increases the makespan by at most* $4\hat{\epsilon}D$.

Proof. After condensing the tree, all leaf clusters have single coverage, and the effect on makespan was covered in Lemma 45. Because of the binary tree structure, we can *assign* each small cluster to a descendant leaf cluster in such a way that each leaf cluster is assigned at most two small clusters. Since each leaf cluster is covered by a single tour, we can require this tour to also cover the clients of the small cluster(s) assigned to that leaf cluster. This is feasible since small clusters are only assigned to *descendant* leaf clusters. Furthermore, since leaf clusters have length at least $\frac{\hat{\epsilon}}{4}D$, we can *charge* this error to the length of the leaf clusters. In particular, since any given tour covers at most $D/(2 \cdot \frac{\hat{\epsilon}}{4}D) = \frac{2}{\hat{\epsilon}}$ leaf clusters, this assignment adds at most $2 \cdot \frac{2}{\hat{\epsilon}} \cdot (2 \cdot \frac{\hat{\epsilon}^2}{2}D) = 4\hat{\epsilon}D$ to the makespan. □

Lemma 47. *Requiring every edge cluster to have single or split coverage adds at most* $3\hat{\epsilon}D$ *to the optimal makespan.*

Proof. Fix an optimal solution, SOL_0. We begin by limiting the number of C-ending tours. For every edge cluster C, let $\gamma(C)$ be the subcluster of C covered by C-ending tour segments. Let $\mathcal{C}_1 = \{\gamma(C) | C \in \mathcal{C}\}$ and let A_1 be the set of tours in SOL_0 that are C-ending tours for at least one edge cluster $C \in \mathcal{C}$. We define a bipartite graph $G_1 = (A_1, \mathcal{C}_1, E_1)$ where there is an edge

$(a, \gamma(C))$ if and only if tour a is a C-ending tour, and $w_{G_1}(a, \gamma(C))$ is the length of the tour segment of a in edge cluster C. Note that $\forall a \in A_1, \gamma(C) \in \mathcal{C}_1$, $w_{G_1}(a, \gamma(C)) \leq 2\ell(C) \leq \hat{\epsilon}D$. We define the weight $w_{G_1}(\gamma(C)) = 2\ell(\gamma(C))$, and for each $a \in A$, we define the capacity $q(a)$ to be the sum $\sum_{C:\gamma(C)\in N_{G_1}(a)} w_{G_1}(a, \gamma(C))$. Clearly, $w_{G_1}(\gamma(C)) \leq \sum_{a\in N_{G_1}(\gamma(C))} w_{G_1}(a, \gamma(C))$, since these tour segments collectively cover $\gamma(C)$. Therefore we can apply Lemma 44 to G_1 to achieve an assignment of subclusters in $\mathcal{C}_1$ to tours such that each subcluster is assigned to one tour and the capacity of each tour is exceeded by at most $\max_{\gamma(C)\in\mathcal{C}_1} w_{G_1}(\gamma(C)) \leq \hat{\epsilon}D$. For any tour $a \in A_1$, let T'_a be the corresponding subgraph visited by a, excluding any edge clusters C for which a is a C-ending tour. T'_a contains r and is connected. Since, for each tour a, $N_{G_1}(a)$ is a tour-independent set with respect to T'_a, this assignment forms a new feasible solution, SOL_1.

At this point each edge cluster C has at most one C-ending tour. We now address C-collecting tours. For every edge cluster C, let $\gamma(L_C) \subseteq L_C$ denote the set of leaf edges of C that are covered by C-collecting tours. Let $\mathcal{L}_2 = \{\gamma(L_C) | C \in \mathcal{C}\}$, and let A_2 be the set of tours in SOL_1 that are C-collecting tours for at least one cluster $C \in \mathcal{C}$. We define a bipartite graph $G_2 = (A_2, \mathcal{L}_2, E_2)$ where there is an edge $(a, \gamma(L_C))$ if and only if tour a is a C-collecting tour, and $w_{G_2}(a, \gamma(L_C))$ is twice the length of the leaves in $\gamma(L_C)$ covered by a. Note that $\forall a \in A_2, C \in \mathcal{C}$, $w_{G_2}(a, \gamma(L_C)) \leq 2\ell(C) \leq \hat{\epsilon}D$. We define the weight $w_{G_2}(\gamma(L_C)) = 2\ell(\gamma(L_C))$, and for each $a \in A$, we define the capacity $q(a)$ to be the sum $\sum_{C:\gamma(C)\in N_{G_2}(a)} w_{G_2}(a, \gamma(L_C))$ of tour segments covering leaves in C-collecting tour segments of a. Clearly, $w_{G_2}(\gamma(L_C)) \leq \sum_{a\in N_{G_2}(\gamma(C))} w_{G_2}(a, C)$, since these tour segments collectively cover $\gamma(L_C)$. Therefore we can apply Lemma 44 to G_2 to achieve an assignment of $\mathcal{L}_2$ leaf sets to tours such that each leaf set is assigned to a single tour and the capacity of each tour is exceeded by at most $\max_{C\in\mathcal{C}} w_{G_2}(\gamma(L_C)) \leq \hat{\epsilon}D$. For any tour $a \in A_2$, let T''_a be the corresponding subgraph visited by a, excluding all leaf sets in L_C. T''_a contains r and is connected. Since, for each tour a, $N_{G_2}(a)$ is a tour-independent set with respect to T''_a, this assignment forms a new feasible solution, SOL_2. Note that if some C-collecting tour a' is not assigned $\gamma(L_C)$, it becomes a C-passing tour.

At this point every edge cluster C has at most one C-collecting tour and at most one C-ending tour. If C has single or split coverage, we are done. Otherwise, let a_1 be the C-ending tour and a_2 be the C-collecting tour. Let $j < m$ be the largest index such that a_1 covers leaf $e_{C,j}$. Since j is the largest such index, then a_2 covers all leaves $e_{C,i}$ for $j < i < m$. If a_1 covers *all* leaves $e_{C,1}, e_{C,2}, ..., e_{C,j}$, then C has split coverage, and we are done. Otherwise, both a_1 and a_2 contain

the backbone edges $P'_C = \{p_{C,1}, \ldots, p_{C,j}\}$ and a subset of the leaf edges $L'_C = \{e_{C,1}, \ldots, e_{C,j}\}$. Using the same argument as above for C-collecting tours, we can assign the leaves L'_C to exactly one of these tours while adding at most $\max_{C \in \mathcal{C}} 2\ell(L'_C) \leq \hat{\epsilon}D$. If the leaves are assigned to a_1, then C ends up having split coverage, and if the leaves are assigned to a_2, then the remaining tour segment of a_1 along the backbone is redundant and can be removed, resulting in every edge cluster C having single coverage.

The resulting solution SOL_3 satisfies the lemma statement. □

We now prove Theorem 28.

Proof. Given Lemma 45, Lemma 46, and Lemma 47, all that remains in proving Theorem 28 is to bound the number of clusters that a single vehicle covers clients in. To do so, we first show that requiring the length of every C-ending or C-collecting tour segment to be at least $\frac{\hat{\epsilon}^3}{2}D$ adds a stretch factor of at most $1 + \hat{\epsilon}$ to the makespan.

Consider any edge cluster C with split coverage. Let a_0 be the C-ending tour, and let a_1 be the C-collecting tour. For $i = 0, 1$ let L_i be the length of a_i within C. Suppose that some L_i is less than $\frac{\hat{\epsilon}^3}{2}D$. Since $L_0 + L_1 \geq 2\ell(C) \geq 2\frac{\hat{\epsilon}^2}{2}D = \hat{\epsilon}^2 D$, then $L_{1-i} \geq \hat{\epsilon}^2(1 - \frac{\hat{\epsilon}}{2})D$ and assigning a_{1-i} to cover the clients served by a_i increases the length of a_{1-i} within C by at most a factor of $1 + \hat{\epsilon}$ since $L_i / L_{i-1} \leq \hat{\epsilon}$. Reassigning these small tour segments within all edge clusters with split coverage increases the length of any given tour by at most a factor of $1 + \hat{\epsilon}$.

After all of the above steps, there exists a solution with makespan at most $(1 + \hat{\epsilon})(1 + 8\hat{\epsilon})D$. Consider some such solution and a single tour a. Let X_1 be the set of leaf clusters that a covers. Since any such leaf cluster has length at least $\frac{\hat{\epsilon}}{4}D$ and is entirely covered by a, $|X_1|$ is $O(\frac{1}{\hat{\epsilon}})$. Let X_2 be the set of small clusters that a covers. Since a covers exactly those small clusters that are assigned to leaf clusters in X_1, $|X_2| \leq 2|X_1|$ and is also $O(\frac{1}{\hat{\epsilon}})$, since each leaf cluster is assigned at most two small clusters. Last, let X_3 be the set of edge clusters C for which a is either a C-ending tour or a C-collecting tour. Since the length of tour a is at most $(1 + O(\hat{\epsilon}))D$, and each of t's tour segments in X_3 has length at least $\frac{\hat{\epsilon}^3}{2}D$, $|X_3|$ is $O(\frac{1}{\hat{\epsilon}^3})$.

□

8.5.2 Minimum Makespan Vehicle Routing Dynamic Program

Having proven a structure theorem, we now present a dynamic programming algorithm (DP) that actually finds a near-optimal solution with simple structure.

Recall, the DP traverses cluster tree T^* starting at the leaves and moving rootward. A configuration is a vector in $\{0, 1, 2, ..., k\}^{2\hat{\epsilon}^{-4}}$. A configuration $\vec{x}$ at a vertex v is interpreted as a set of tours *projected up to* r in T that cover all clusters in the subtree of T^* rooted at v. For $i \in \{1, 2, ..., 2\hat{\epsilon}^{-4}\}$, $\vec{x}[i]$ is the number of tours in the set that have *rounded* length $i\hat{\epsilon}^4 D$. That is, the actual tours that correspond to the $\vec{x}[i]$ tours represented in the vector each have length that may be less than $i\hat{\epsilon}^4 D$.

The algorithm categorizes the vertices into three different cases and handles them separately. The *base cases* are the leaves of T^*. Let $v \in T^*$ be such a leaf, let L_v be the corresponding leaf cluster in T, and let u be the vertex at which L_v meets the backbone. When the algorithm determines the configuration for v it addresses covering both L_v as well as covering any small clusters $C_1, ..., C_h$ that are assigned to L_v. Let ℓ_{small} be the length of all of the *leaves* of these small clusters, namely $\ell_{small} = \ell(\bigcup_{1 \le i \le h} C_i \setminus \text{backbone})$. Let ℓ_0 be $2(\ell(L_v) + \ell_{small} + d_T(u, r))$ rounded up to the nearest $\hat{\epsilon}^4 D$. The only configuration stored at v is $\vec{x}$ such that $\vec{x}[\ell_0] = 1$ and $\vec{x}[j] = 0, \forall j \neq \ell_0$. All cluster lengths and distances to the depot can be precomputed in linear time, after which each base case can be computed in constant time.

The *grow cases* are the vertices in T^* that correspond to edge clusters in T. Let $v \in T^*$ be such a vertex, and let C_v be the corresponding edge cluster in T. Let u be the root-most vertex in C_v, and let $v' \in T^*$ be the lone child vertex of v. Note that v' may correspond to a branching backbone vertex, a leaf cluster or another edge cluster, but by construction, v has exactly one child. Since C_v has single or split coverage, at most two tours in any configuration at v are involved in covering the leaves of C_v: all other tours in the configuration are C_v-passing tours, and their representation in the configuration remains unchanged. The algorithm considers all possible rounded tour lengths ℓ_1 for a C_v-ending tour t_1 for the configuration (including not having such a tour) and for each such t_1, the algorithm considers all possible (rounded) lengths ℓ_2 for an incoming C_v-collecting tour t_2, *before* the remaining length from covering leaves in C_v is added to the tour. Given ℓ_1 and ℓ_2, the algorithm can easily compute the resulting rounded length ℓ_3 of t_2 *after* covering its share of C_v leaves. For each configuration $\vec{x}'$ for child vertex v', the algorithm determines configuration $\vec{x}$ for v such that $\vec{x}[\ell_1] = \vec{x}'[\ell_1] + 1$, $\vec{x}[\ell_2] = \vec{x}'[\ell_2] - 1$, $\vec{x}[\ell_3] = \vec{x}'[\ell_3] + 1$, and $\vec{x}[i] = \vec{x}'[i]$ otherwise. If the

resulting $\vec{x}$ is feasible, it is stored at v. Since there are at most $2\hat{\epsilon}^{-4}$ options for ℓ_1 and ℓ_2 and at most $k^{2\hat{\epsilon}^{-4}}$ configurations at v', the runtime for each grow case is $k^{O(\hat{\epsilon}^{-4})}$.

Finally, the *merge cases* are the vertices in T^* that correspond to branching backbone vertices in T as well as the depot. Let $v_0 \in T^*$ be such a vertex, and let u be the corresponding vertex in T. Let $v_1, v_2 \in T^*$ be the two children of v_0 in T^*. Every tour t_0 in a configuration at v_0 will either be a directly inherited tour t_i of rounded length ℓ_i from a configuration at v_i for $i \in \{1, 2\}$, or will be a *merging* of some tour t_1 from v_1 and some t_2 from v_2 with resulting length $\ell_1 + \ell_2 - 2\ell(u, r)$ rounded up to the nearest $\hat{\epsilon}^4 D$ (recall that t_1 and t_2 are tours from the depot so the subtracted amount addresses over-counting the path to the depot). For every possible (ℓ_1, ℓ_2) (including lengths of zero to account for tours inherited by children), the algorithm considers how many tours at v_0 could have resulted from merging a tour of length ℓ_1 from v_1 with a tour of length ℓ_2 from v_2. Each of these possibilities corresponds to a configuration $\vec{x}_i$ at v_i for $i \in \{1, 2\}$ and to a merged configuration $\vec{x}$ at v_0. If $\vec{x}_1$ and $\vec{x}_2$ are valid configurations stored at v_1 and v_2, respectively, then the algorithm stores $\vec{x}$ at v_0. There are $k^{4\hat{\epsilon}^{-8}}$ such possibilities, so the runtime of each merge case is $k^{O(\hat{\epsilon}^{-8})}$. Note that the dynamic program only considers storing feasible configurations $\vec{x}$ at vertex v_0 so the algorithm maintains that there are at most k total tours.

Since for any $\epsilon > 0$ the DP has a polynomial runtime, the following lemma completes the proof of Theorem 27.

Lemma 48. *The dynamic program described above finds a tour with maximum makespan at most* $(1 + \epsilon)D$.

Proof. By Theorem 28, each tour covers clients in $O(1/\hat{\epsilon}^3)$ clusters and has accumulated $c_1\hat{\epsilon}D$ error during the simplification steps, for some constant c_1. Notably, each tour only covers the backbone edges that lead to clusters in which it covers clients, since otherwise these edges can be removed. Therefore, the number of backbone branching vertices that a tour branches at is also $O(1/\hat{\epsilon}^3)$. Note that the tour might pass other backbone branching vertices without the tour itself actually branching there. The dynamic program rounds tour lengths up to the nearest $\hat{\epsilon}^4 D$ exactly at the clusters in which the tour covers clients and at the backbone branching vertices at which the tour branches. Therefore each tour incurs an error of at most $\hat{\epsilon}^4 D$ at each of the $O(1/\hat{\epsilon}^3)$ times that the dynamic program rounds the tour, for a total rounding error of $c_2\hat{\epsilon}D$ for some constant c_2. The total increase in the makespan is $(c_1 + c_2)\hat{\epsilon}D$. Setting $\hat{\epsilon}$ to $\frac{\epsilon}{c_1+c_2}$ completes the proof. □

8.5.3 Previous Claim to a Minimum Makespan PTAS

The intuition behind our result is similar to that of [28]: since the main challenge arises from having to account for small tour segments, modify the instance so that only large tour segments need to be accounted for. Because each tour can only have a small number of these larger tour segments, the algorithm can essentially *guess* all possible ways that the segments can be joined to form tours. The difficulty with this approach is in showing that this type of grouping of small segments into large segments can be done without incurring too much error. As did [28], we also observed that assuming each subtree with length less than $\epsilon cost(OPT)$ to be covered by a single tour increases the makespan by $O(\epsilon cost(OPT))$, where $cost(OPT)$ is the optimum makespan.

The presented argument of Theorem 2.1 [28] (which their PTAS requires) depends on the following argument, though they use different terminology: Let T_v denote the subtree rooted at v. The input can be *safely* modified[2] so that there exists a set of vertices CR such that the subtrees rooted at vertices in CR collectively *partition* the leaves of the tree and such that CR has the following properties:

1. For each $v \in CR$, T_v is *small*: for each child u of v, the length of T_u is less than $\epsilon cost(OPT)$.
2. For each $v \in CR$, T_v is *large*: the length of T_v is at least $\epsilon cost(OPT)$.
3. The vertices of CR are *independent*: no two distinct vertices in CR have an ancestor-descendant relationship.

Each of these properties is vital to the proof of Theorem 2.1 [28]: Property 1 bounds the solution error, Property 2 ensures polynomial runtime, and Property 3 ensures solution feasibility.

Given such a set CR, Chen and Marx [28] prove that there exists a near-optimal solution in which each *branch* of a subtree rooted at a vertex in CR is covered by a single tour. They do so by showing that an optimal solution SOL can be transformed into a near-optimal solution SOL' in which each such branch is wholly covered by one of the tours that partially covers it in SOL. This proof depends on Property 1 to bound the resulting error of this transformation and on Property 3 to maintain solution feasibility while independently reassigning branch coverage. A dynamic programming strategy can then be used to find such a near-optimal solution, which requires Property 2 in order to achieve a polynomial runtime.

[2] *Safe* graph modifications are those that result in an equivalent instance and consist of subdividing a single edge e_0 into two smaller edges e_1 and e_2 such that $\ell(e_1) + \ell(e_2) = \ell(e_0)$.

The problem, though, is that no such set CR is guaranteed to exist. In the process of modifying the input graph to ensure that their proposed set CR satisfies Property 3, they end up with a set that no longer satisfies Property 2.

In fact, it is not difficult to show the following.

Lemma 49. *There exist instances in which no such set CR as described above exists.*

Proof. We describe an instance that neither admits such a set CR nor can be modified using *safe* graph modifications into an equivalent instance that admits such a set. Let T be a tree with root r that consists of a *central* path $P = r, v_1, v_2, ..., v_l$, *side* subtrees T_i for $i \in \{1, 2, ..., l\}$ such that the root of T_i is a child of v_i, and *main* subtree T_{l+1} whose root v_{l+1} is a child of v_l. Let edges (v_i, v_{i+1}) along P have small nonzero lengths, let each side subtree T_i have total length less than $\frac{\epsilon cost(OPT)}{2}$ and let the main subtree T_{l+1} have total length at least $\frac{\epsilon cost(OPT)}{2}$ (see Figure 8.5). In order to satisfy Property 2, no vertex v in any side subtree T_i for $0 < i \leq l$ can be in CR, since the subtree rooted at such a v is not large enough. So, since the subtrees rooted at vertices in CR partition the leaves of T, at *least* one vertex in P must be in CR to cover the leaves of trees T_i. At the same time, in order to satisfy Property 3, at *most* one vertex in P can be in CR. But, since main subtree T_{l+1} is large, no choice of vertex in P can satisfy Property 1.

Note that the side and main subtrees are general, and can be assumed to be defined after any safe modifications occur. Additionally, if any edges of P are subdivided, the argument above clearly still holds.

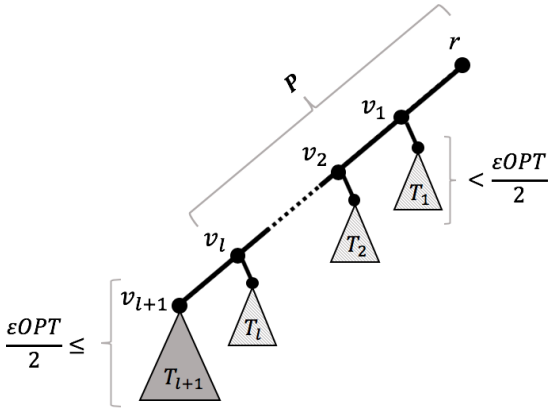


Figure 8.5: Counterexample to [28].

□

The counterexample in the above proof allows an arbitrarily complex and heavy main subtree T_{l+1}, so it cannot be argued that only trivial examples fail to admit a set CR. In particular, this pattern of light side subtrees can continue throughout the main subtree, so these *difficult areas* cannot be easily dealt with separately.

To the best of our knowledge, there is no straightforward way to salvage the theorem used in [28]. In fact, the crux of the problem is highlighted by the *absence* of such a set CR: How can the input be simplified so as to balance error with runtime given that coverage choices are *not* independent? Our approach uses similar intuition to that of [28], but manages to successfully address the critical challenges of the problem where prior attempts have fallen short.

Rather than partition the tree uniformly into subtrees, we recognize that the behavior of solutions *looks* different near the leaves than it does along internal vertices. As such, our framework partitions the *entire* tree into three different types of *clusters* (see Figure 8.1b). *Leaf clusters* are defined greedily so as to satisfy properties similar to 1-3 above and serve to anchor the cluster structure. *Small clusters* are internal clusters with weight (and frequency) small enough to be effectively ignored while incurring only a small error that can be *charged* to the leaf clusters. The remainder of the tree is grouped into *edge clusters*, with properties similar to 1 and 2 above: their weight is *small* enough to assume, without incurring too much error, a *simple* structure to the way that tours cover them, yet they are *large* enough so that any tour can only cover a bounded number of them. Though the edge clusters themselves do *not* maintain independence, we show a weaker, sufficient property exhibited by tour segments in these clusters that ensures tour connectivity while treating edge clusters independently.

8.5.4 Distance-Constrained Vehicle Routing

Recall that the Distance-Constrained Vehicle Routing problem is to minimize the number of tours of length at most D required to cover all clients. Since it is the Minimum Fleet Budget problem associated with Minimum Makespan Vehicle Routing, the following bicriteria PTAS follows as a corollary to Theorem 27.

Theorem 29. *Given an instance of* Distance-Constrained Vehicle Routing *on a tree, if there exists a solution with k tours of length at most D, then for any $\epsilon > 0$, there is a polynomial-time algorithm that finds a solution with k tours of length at most $(1+\epsilon)D$.*

8.6 Applications and Extensions

So far, we have introduced a framework and shown how to customize it to a specific problem (Minimum Makespan Vehicle Routing). In this section we demonstrate the flexibility and usefulness of our framework by showing how to customize it to seemingly very different problems as well as to the multiple-depot setting. We give a few examples in this section, but we believe the framework can serve as a tool for applications beyond those we mention.

We first use Capacitated Vehicle Routing as an example of a problem whose customization is very similar to that of Minimum Makespan Vehicle Routing. We then use School Bus Routing as an example of customizing the framework to a very different type of problem. Finally, we describe how the framework can be extended to accommodate the multiple-depot setting.

8.6.1 Capacitated Vehicle Routing

Recall that in Capacitated Vehicle Routing, each tour can cover at most Q clients and the objective is to minimize the *sum* of the tour lengths. This problem is quite different from Minimum Makespan Vehicle Routing: there is no longer a constraint on the number of tours or the length of each tour, and the solution cost depends on all tours rather than just the largest one. Yet we show in this section that these two problems have surprisingly similar framework customizations.

Because Capacitated Vehicle Routing is a Minimum Fleet Budget problem, we must apply the framework directly to the associated Min-Max Vehicle Load problem, Min-Max Client Capacity that seeks to minimize the maximum client capacity, Q, required of a vehicle given a bound k on the sum of tour lengths.

Customizing the framework to Min-Max Client Capacity is nearly identical to Minimum Makespan Vehicle Routing, except that vehicle load is the number of clients, rather than length, and the fleet budget is the sum of the tour lengths, rather than the number of vehicles. Correspondingly, we define the load $g(H)$ of a subgraph H to be the *number* of clients in the subgraph. Otherwise, the solution simplification and analysis proceed as in Minimum Makespan Vehicle Routing, noting that because the objective function is the sum of tour lengths, assigning an entire branch to be covered by a single tour never increases the fleet budget.

The Min-Max Client Capacity DP configuration is a vector in $\{0, 1, 2, ..., n\}^{\hat{\epsilon}^{-4}}$, where a configuration $\vec{x}$ at a vertex v is interpreted as a set of tours in T that collectively cover all clusters

in the subtree of T^* rooted at v. Now, however, for $i \in \{1, 2, ..., \hat{\epsilon}^{-4}\}$, $\vec{x}[i]$ is the number of tours in the set that have rounded *number of clients* $i\hat{\epsilon}^4 Q$. That is, the actual tours that correspond to the $\vec{x}[i]$ tours represented in the vector each cover at most $i\hat{\epsilon}^4 Q$ clients. For each of these configurations, the DP stores the minimum total length of a solution that is consistent with the configuration.

The resulting PTAS for Min-Max Client Capacity gives the following bicriteria result for Capacitated Vehicle Routing as a corollary.

Theorem 30. *Given an instance of* Capacitated Vehicle Routing *on a tree, if there exists a solution of total length k and capacity Q, then for any $\epsilon > 0$, there is a polynomial-time algorithm that finds a solution of total length k and capacity at most $(1 + \epsilon)Q$.*

8.6.2 School Bus Routing

A rather different vehicle load measure is maximum client *regret*: the maximum difference between the time a client is *actually* visited and the earliest possible time that client *could have* been visited. Note that maximum client regret is always achieved for the *final* client visited by a vehicle. It is therefore convenient to assume the routes to be *paths*. Recall, the regret of a u-to-v path, P, is the difference between the length of the path and the distance from u to v, namely $\ell(P) - d_T(u, v)$.

The Min-Max Regret Routing problem, given a number of vehicles, k, is to cover all clients with k paths starting from the depot, such that the maximum regret of a path is minimized. Here, vehicle load is the regret of the vehicle's path and fleet budget is the number of vehicles. The related Minimum Fleet Budget problem, the School Bus Routing problem, given a regret bound R, is to find the smallest number of paths of regret at most R that each start at the depot, r, and collectively cover all clients.

In this section, we sketch how to customize our framework to the Min-Max Regret Routing and achieve the following.

Theorem 31. *For every $\epsilon > 0$, there is a polynomial-time algorithm that, given an instance of* Min-Max Regret Routing *on a tree, finds a solution whose maximum regret is at most $1 + \epsilon$ times optimum.*

Both a bicriteria approximation scheme and a 2-approximation for the School Bus Routing problem follow as corollaries.

Theorem 32. *Given an instance of the* School Bus Routing *problem on a tree, if there exists a solution consisting of k paths of regret at most R, then for any $\epsilon > 0$, there is polynomial-time algorithm that finds a solution consisting of k paths of regret at most $(1+\epsilon)R$.*

Theorem 33. *There is a polynomial-time 2-approximation for the* School Bus Routing *problem in trees.*

Proof. [49] show that k paths of regret at most αR can be replaced with at most $\lceil \alpha \rceil k$ paths each of regret at most R. Applying this to Theorem 32 completes the proof. □

The ability of our framework to capture a regret-based load function demonstrates the flexibility of the framework and gives evidence for its potential application to problems outside those mentioned in this chapter.

It is useful (and equivalent) to assume that each vehicle path *ends* at the depot rather than begins there; we assume each vehicle starts at some origin vertex and travels to the depot r, possibly taking some detours on branches along the way. Note that in tree instances, these *regret detours* exactly correspond to the regret accumulated by the vehicle, whereas traveling along the origin-to-depot path is free. The biggest challenge with applying the framework to Min-Max Regret Routing is the asymmetry of these two aspects of the path.

Structure Theorem

For Min-Max Regret Routing, defining a load function for subgraphs is not as straightforward as for other problems, as the regret accumulated by a path covering subgraph H is highly dependent on where it starts and ends. If the vehicle is traversing H on its way to the depot, then some of H can be covered for free, namely the intersection of H with the shortest path from the vehicle's origin to the depot. For this reason, we require $g(\cdot)$ to take two additional parameters, vertices $u, v \in H$, and define $g(H, u, v)$ to be the minimum regret required for a path to cover H, given that u and v are respectively the *first* and *last* vertex in H on the path. This means that u and v are each either end points of the path or on the *boundary* of H.

Let $\hat{\epsilon} = \epsilon/c$ for some constant c we will define later, and $\delta = \hat{\epsilon}^2$. We first update the condensing and clustering steps to use this extended load function. For a branch b that is a u-branch at v, the branch load is measured as $g(b, v, v)$ when applying the condense operation. That is, the regret accumulated by a vehicle covering b and starting *outside of* b, or equivalently, twice the length of

b. Leaf clusters therefore correspond to branches of length at least $\frac{\hat{\epsilon}^2}{4}D$ (i.e. load at least $\frac{\hat{\epsilon}^2}{2}D$). Let the leaf edges of a woolly subedge be called the *wool*. Every woolly subedge can therefore be partitioned into backbone and wool. The load of a woolly subedge $e = (u, v)$ is defined as $g(e, u, v)$, which is twice the length of the wool of e (i.e. twice the sum of the leaf edge lengths). Therefore, small clusters are the woolly edges that have total wool length less than $\frac{\hat{\epsilon}^3}{4}D$. Edge clusters have total wool length in $[\frac{\hat{\epsilon}^3}{4}D, \frac{\hat{\epsilon}^2}{4}D]$ and partition the remaining woolly edges.

The condense operation increases the regret of any path by at most $2\hat{\epsilon}^2 D$. If any path starts in a branch b that is condensed, it can be assumed instead that the branch starts at the root-most vertex of b. This adds at most $\hat{\epsilon}^2 D$ regret to any path, since each path has exactly one starting point. The condensed branches are now all covered by *detours*, so the reassignment argument is equivalent to Minimum Makespan Vehicle Routing, adding at most an additional $\hat{\epsilon}^2 D$ regret. Without loss of generality, we assume that *all* paths start at the *top* (rootmost vertex) of some leaf cluster: paths are already assumed to start on the backbone, and the start point can always be extended to a descendant vertex without increasing the path's regret.

As in Minimum Makespan Vehicle Routing, small clusters can be assigned (and charged) to descendant leaf clusters such that the vehicle covering a given leaf cluster also covers at most two small clusters. We have now accounted for the single coverage of leaf and small clusters.

Next, consider some edge cluster C. C-passing and C-collecting routes here consist both of *through paths* that start in a descendant of C as well as *through detours* that do not start in a descendant of C but extend through C to descendant clusters (see Figure 8.6). As before, regret can be reassigned so that each edge cluster is covered by at most one C-collecting route, adding at most $\frac{\hat{\epsilon}^2}{2}D$ regret to any path, since edge clusters have total wool length at most $\frac{\hat{\epsilon}^2}{4}D$.

Furthermore, if the largest amount of regret accumulated in C by a C-ending route is at most $\hat{\epsilon}D$, then regret can be reassigned to a single C-ending route. Alternatively, if this amount is more than $\hat{\epsilon}D$ and is accumulated by C-ending route t, then all regret from other C-ending routes can be *charged* to t's regret in C, since this adds at most $\frac{\hat{\epsilon}^2}{2}D$ to the regret of t.

Using proofs analogous to those in Section 8.5 we can show that there exists a near-optimal solution such that small clusters and leaf clusters have single coverage, and edge clusters have split coverage. All that remains to proving the structure theorem is to show that there exists such a solution such that each vehicle covers clients in a bounded number of clusters.

Bounding the number of leaf clusters and small clusters a vehicle covers is straightforward: leaf

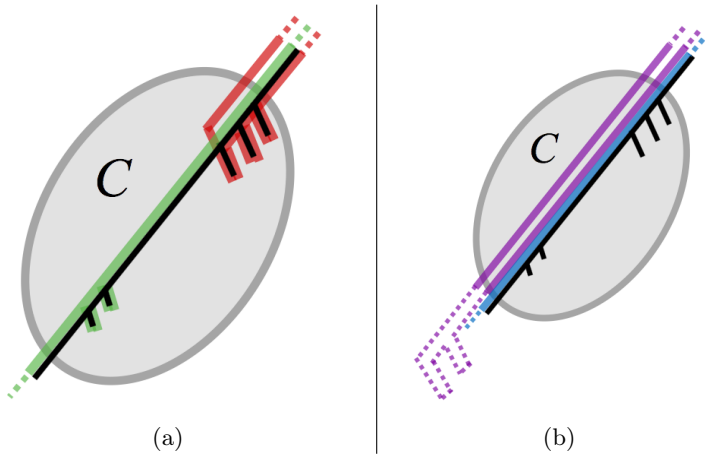


Figure 8.6: (a) The red path is a C-ending path and the green path is a C-collecting through path (b) The blue path is a C-passing through path and the purple path is a C-passing through detour.

clusters have large regret so a vehicle can only afford to cover a small number of them, and each vehicle covers at most 2 small clusters for every leaf cluster. Edge clusters with either single coverage, or *balanced* split coverage (in which both vehicles cover a large fraction of the cluster's regret) are similarly straightforward to handle (see proof of Theorem 28). Moreover, if C has split coverage and the C-ending path has very small regret in C (e.g. less than $\hat{\epsilon}^4 D$), its clients can be reassigned and charged to the C-collecting route, which must have a relatively large regret and can afford to singly cover C.

The difficult case is when C has split coverage and the C-collecting path has very small regret in C. Unlike in Theorem 28, even though the C-ending path has relatively large regret, it cannot afford to singly cover C because the *backbone* may be arbitrarily long. We say that C has *dispersed* coverage in this case and bound the number of times that any given vehicle is the C-collecting route for a cluster with *dispersed* coverage.

Consider some woolly edge e, and let $\{C_1, C_2, ..., C_l\}$ be the clusters along e that have dispersed coverage, ordered such that C_{i+1} is rootward of C_i for all i. Though the C_1-ending path cannot afford to cover the clients covered by the C_1-collecting path in C_1, it *can* afford to cover the clients covered by the C_2-collecting path in C_2, since it is necessarily passing through C_2 on the way to the depot. In general, the C_i-ending path can cover the clients covered by the C_{i+1}-collecting path in C_{i+1}. Finally, we can assign woolly edge e to a descendant leaf cluster C_L in such a way that each leaf cluster is assigned at most two woolly edges, and charge the regret of the C_1-collecting path in C_1 to the vehicle that covers C_L. This *propagation reassignment* only increases the maximum

regret of a vehicle by a $(1 + \hat{\epsilon})$ factor. This bounds the number of times a given vehicle serves as a C-collecting path in a cluster C with dispersed coverage: the small regret accumulated in C is charged to the much larger regret the vehicle covers in a descendant cluster, and the vehicle can only cover such a large amount of regret a bounded number of times.

Dynamic Program

The dynamic program for the School Bus Routing problem must account separately for the through paths and regret detours. For through paths, a configuration stores the (rounded) regret already accumulated, whereas for regret detours a configuration stores the (rounded) total detour length. Of note, at branching points a regret detour can be merged with a through path or with another regret detour, but two through paths cannot be merged together.

The bigger challenge with adapting the DP is with route projection: a regret detour can be arbitrarily far from the root, so the DP cannot project the detour all the way to the root. We address this by setting *breakpoints* along the tree such that for every vertex v, there is an ancestor breakpoint within a distance R, and such that no two breakpoints are within ϵR of each other. Breakpoints can be designated greedily. The DP then stores rounded regret detours projected to the next rootward breakpoint. By construction, when two detours are merged, they agree on breakpoint, and twice the distance to the breakpoint can be subtracted, as with merging a detour and a through path. When a breakpoint is encountered by the DP, all regret detours are projected to the next rootward breakpoint. This adds $O(\epsilon R)$ rounding operations to any given path, and is therefore negligible.

8.6.3 Generalizing to Multiple Depots

In this section, we sketch how our framework can be extended to allow for multiple depots. We illustrate this extension using Minimum Makespan Vehicle Routing, but note that it can be applied to multiple-depot variants of other problems too. Here, each tour must start and end at the same depot, and a client can be covered by a tour from any depot.

Theorem 34. *For every $\epsilon > 0$ and $\rho > 0$, there is a polynomial-time algorithm that, given an instance of ρ-Depot Minimum Makespan Vehicle Routing on a tree, finds a solution whose makespan is at most $1 + \epsilon$ times optimum.*

Structure Theorem

Label the depots $\{r_1, r_2, ..., r_\rho\}$, and, without loss of generality, assume each depot is a leaf and that the tree is rooted at some arbitrary vertex r_0, rather than at a depot. The clustering is then done, with respect to r_0, identically to the single-depot setting, with $\delta = \hat{\epsilon}$ to be defined later, and with the load $g(H)$ of subgraph H again being defined to be twice the length of edges in H.

It is more convenient to assume that the depots are *outside of* the clusters. During the clustering process, if a depot r_i is in a cluster C with root-most vertex v, then a leaf edge (v, r_i') of length $d_T(r_i, v)$ is added to v, padded with zero length edges, to maintain a binary tree. The depot is then assumed to be located at r_i', and thus outside of C (see Figure 8.7). This modification adds at most $\hat{\epsilon}D$ to the optimal makespan, to account for the potential extra distance that tours from r_i must now travel to reach r_i'. Otherwise, the increase to makespan caused by condensing is equivalent to Lemma 45 of the single-depot setting.

To show that requiring all small clusters to have single coverage does not increase the optimal makespan by too much is more challenging than in the single-depot setting. Recall that with only one depot, the load from a small cluster C_S could be charged to a descendant leaf cluster C_L, because the tour t_L that covers C_L *must* pass through C_S on the way to the root depot. This is not true for the multi-depot setting.

Observe though that if C_S is incident to a *woolly* branch b (a branch of the condensed tree) that contains no depots, then the load from C_S *can* be charged to a leaf cluster C_L in b, since the tour covering C_L must leave branch b to reach its depot, and therefore either pass through or abut C_S. Such small clusters can be assigned to such leaf clusters so that each leaf cluster is charged for at most three small clusters, adding at most $4\hat{\epsilon}D$ to the makespan, as in Lemma 46.

What remain are the small clusters *not* incident to such depot-free branches. We claim that there are at most 2ρ such clusters. Note that since we moved depots to be outside of clusters, the cluster tree T^*, resulting from contracting clusters, now has vertices for each cluster, depot, branching point, and the root. Consider the *meta* tree structure T^*_{meta} induced by the depots and the root (see Figure 8.8b). There are fewer than 2ρ edges of T^*_{meta} since it is a binary tree with depot leaves. Consider some edge e_{meta} of T^*_{meta}. If e_{meta} *contains* more than one small cluster (i.e. corresponds to a subgraph of T^* with more than one small cluster vertex), those small clusters must be incident to depot-free branches. Otherwise, e_{meta} contains at most one small cluster. Therefore,

there are at most 2ρ small clusters not incident to depot-free branches, and requiring them to have single coverages increases the optimal makespan by at most $\rho\hat{\epsilon}^2 D$.

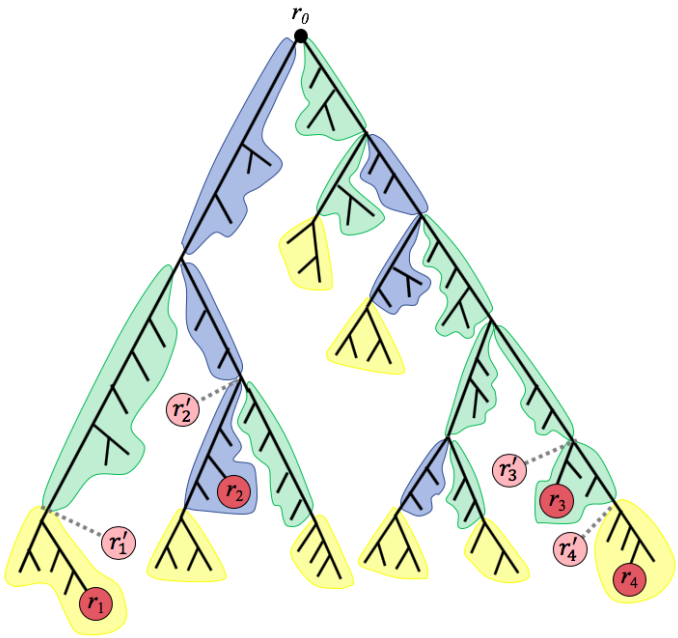


Figure 8.7: A four-depot clustering example showing depots moved outside of clusters. Leaf clusters are yellow, small clusters are blue, and edge clusters are green.

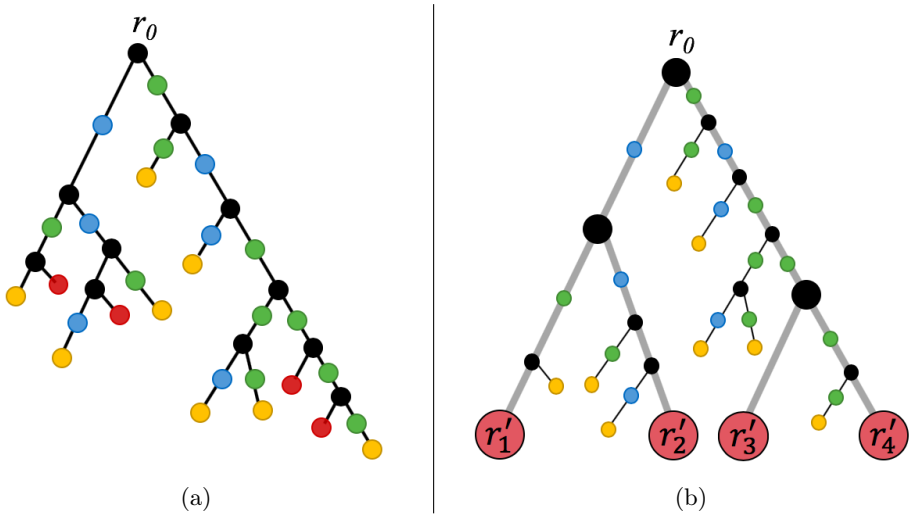


Figure 8.8: (a) Cluster tree T^* corresponding to Figure 8.1; (b) Corresponding meta tree T^*_{meta}. Meta edges are shown as thick gray lines.

Last, the definition of split coverage needs to be adjusted to accommodate multiple depots. In particular an edge cluster C may have C-ending tours from each end. We let *top* C-ending tours

denote those that enter C from the root-ward end and *bottom* C-ending tours denote those that enter C from the leaf-ward end. In the multi-depot setting, the analogous definition of split coverage for an edge cluster C is that the leaves of C along the backbone can be partitioned into three continuous intervals, such that if the root-most interval is nonempty it is covered by a single top C-ending tour, if the middle interval is nonempty it is covered by a single C-collecting tour, and if the leaf-most interval is nonempty it is covered by a single bottom C-ending tour. Using analogous analysis to Lemma 47 shows that requiring edge clusters to have split coverage adds at most $5\hat{\epsilon}D$ to the optimal makespan.

It can then be shown that there is a solution of makespan $(1 + O(\hat{\epsilon} + \rho\hat{\epsilon}^2))D$ such that each tour covers clients in $O(\frac{1}{\hat{\epsilon}^3} + \rho)$ clusters. To prove the structure theorem, it then suffices to set $\hat{\epsilon} = \frac{\epsilon}{c\rho}$ for an appropriate constant c.

Dynamic Program

To adapt the dynamic program for multiple depots, we first note that the configurations must now account for tours *to each depot*. The challenge is that our single-depot dynamic program depends on projecting vehicle routes *up* the tree, toward a common depot. In particular, this was the same direction as the DP traversal itself. In the multiple-depot setting, the DP must move *away from* depots, but it cannot afford to guess which clients to project a tour toward. To address this issue, for each depot r_i we assign a *sub-root* $\bar{r}_i$ to be the root-most vertex that is within a distance $D/2$ of r_i. In particular, this sub-root is the root-most vertex that can possibly be included in any tour from r_i.

When the DP introduces a new tour to depot r_i, it is then projected *up* to $\bar{r}_i$ instead of down to r_i. This includes when the DP processes r_i itself: a configuration describes how many tours start (and end) at r_i, and they are all projected from r_i up to $\bar{r}_i$ (see Figure 8.9a). Two depot-r_i tour segments t_1 and t_2 can only be merged at some vertex v if $\ell(t_1) + \ell(t_2) - 4d_T(v, \bar{r}_i) \leq (1+\epsilon)D$. After merging, they are stored as a depot-r_i tour segment of length $\ell(t_1) + \ell(t_2) - 2d_T(v, \bar{r}_i)$ rounded up to the nearest $\hat{\epsilon}^4 D$ (see Figure 8.9b). If at some vertex v, the (rounded) projected tour length reaches $2d_T(v, \bar{r}_i) + (1 + \epsilon)D$, the projected subtour from v to $\bar{r}_i$ is removed and the tour ends at v (see Figure 8.9c). In other words, v is the root-most vertex reached by that tour. The overall runtime is $n^{O(\rho\epsilon^{-8})}$.

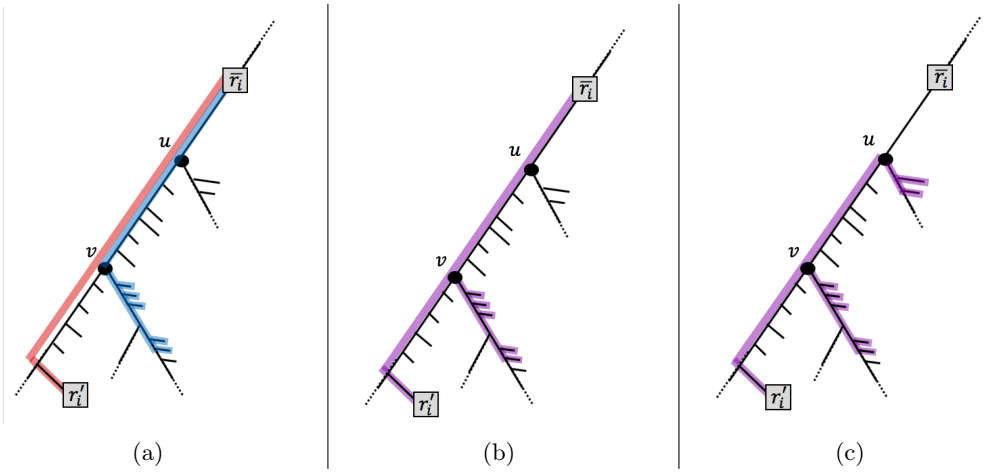


Figure 8.9: (a) Tours to r_i are projected to $\bar{r}_i$; (b) Tours to r_i from (a) merged at vertex v and projected to $\bar{r}_i$; (c) Tour to r_i capped at vertex u once tour length reaches $(1+\epsilon)D$.

Chapter 9

Conclusion

To conclude this thesis we will first discuss some questions that remain unanswered by our work that may be candidates for future research in this area. We then give some closing remarks.

9.1 Open Problems

Here we describe opportunities for future work in this line of research. The results presented in this thesis contribute to the understanding of approximation algorithms for facility-placement and vehicle-routing problems in metrics that model transportation networks. But many open problems remain, particularly in the space between bounded and general parameters as well as the space between simple and general metrics.

9.1.1 Capacitated ℓ-Domination

We showed that when distance ℓ and maximum capacity $\hat{q}$ are bounded, there is a PTAS for Capacitated ℓ-Domination in planar graphs. This raises the question as to whether such a PTAS can be found for general capacities and distances. The runtime of our algorithm is exponential in both ℓ and $\hat{q}$, so it is unlikely that a similar approach would yield such a PTAS. In fact, even when $\ell = 1$, Capacitated Domination is W[1]-hard for planar graphs when parameterized by treewidth and solution size [38]. Additionally, lower bound results for both Dominating Set [78] and ℓ-Domination [24] show that under the Strong Exponential-Time Hypothesis (SETH), the runtime for these problems has an exponential dependence on branchwidth. For a Baker-like approach to

work, the branchwidth of each slab would have to be independent of the maximum capacity.

Furthermore, it has been shown that under the Exponential-Time Hypothesis (ETH), (uncapacitated) ℓ-Domination in planar graphs cannot be solved in $f(k)n^{o(\sqrt{k})}$ time, where k is the solution size [80]. This implies that an EPTAS for even the uncapacitated version of the problem would refute ETH [48]. A recent bicriteria EPTAS was given for ℓ-Domination in planar graphs that, if there exists a set of k vertices that ℓ-dominates V, finds a set of $(1+\epsilon)k$ vertices that $(1+\epsilon)\,\ell$-dominates V [48]. A natural question is whether such a bicriteria approximation scheme exists for the capacitated version.

Another open question is whether a PTAS exists for Capacitated ℓ-Domination (or even Capacitated Domination) with arbitrary vertex weights in planar graphs. Our algorithm can be straightforwardly extended to accommodate bounded vertex weights. To achieve this, we pay a factor of $\hat{w}$ in the treewidth so that we can afford to replace an overloaded vertex of weight one with an underloaded vertex of weight $\hat{w}$. The challenge with extending this strategy to arbitrary vertex weights is that the underloaded vertices may be arbitrarily more expensive than the overloaded vertices, and it is unclear how to charge for the smoothing process. One idea is to design a more nuanced smoothing process that considers the weights of the vertices chosen in the search. Alternatively a careful rounding strategy may lead to a bicriteria result in which either the capacities can be exceeded or the demands can be under-covered by a small (bounded) amount.

More generally, it remains to explore what other (hard)-capacitated problems can be addressed with extensions to Baker's framework.

9.1.2 Capacitated Vehicle Routing

Fixed Capacities

For fixed capacity Q, we presented a QPTAS for Capacitated Vehicle Routing in planar graphs, which we then improved to a PTAS. We also gave a PTAS for Capacitated Vehicle Routing in graphs of bounded highway dimension. A natural question is, what other graph families have such approximation schemes. As shown in Chapter 3, we have reduced this question to determining which graph families can be embedded into graphs of bounded treewidth while approximately preserving distances, up to a given error allowance.

Our approach, however, depends on a dynamic program in which the degree of the runtime

polynomial depends on Q. One question is whether, for some non-trivial family of graphs, there exists a fixed-parameter approximation scheme when the problem is parameterized by Q. That is, can a solution that is within a $(1+\epsilon)$ factor of optimal be found in $f(Q)n^c$ time where c is independent of Q (but can depend on ϵ) and f is an arbitrary function independent of n. This is not known, even for trees.

Additionally, although the approximation schemes we present take polynomial time, they are not *efficient*. That is, the runtime of our algorithm is bounded by a polynomial whose degree depends on ϵ. Can this dependence on ϵ be avoided? It is an open question as to whether efficient PTASs exist.

General Capacities

In Chapter 7 we presented a 4/3-approximation algorithm for Capacitated Vehicle Routing with arbitrary capacities and splittable demands in trees. This is the best known approximation ratio and, as shown, is tight with respect to the best-known lower bound. Improving on this ratio would require finding a tighter lower bound. Alternatively, it is unknown whether there is a lower bound in the approximation ratio for the splittable-demand setting. As for the unsplittable-demand setting, recall that there is a 2-approximation known for trees [74] and it is NP-hard to approximate to better than a 3/2 factor [52]. This leaves the question, does there exist an α approximation for some $\alpha \in [3/2, 2)$?

Even though it is not known whether there exits a PTAS for splittable-demand Capacitated Vehicle Routing in trees, in Chapter 8 we present a *bicriteria* PTAS for trees, in which capacities are allowed to be exceeded by a $(1+\epsilon)$ factor. One question is whether such bicriteria polynomial-time approximation schemes can be found for other families of graphs. Even extending to graphs of bounded treewidth would be significant.

Recall that for Capacitated Vehicle Routing in the Euclidean plane, there is a QPTAS known for general capacities [36]. Can an analogous QPTAS (or even bicriteria QPTAS) be designed for other metrics such as planar or bounded-treewidth graphs? The challenge with extending the approach used for $\mathbb{R}^2$ is that it depends on polylogarithmically bounding the number of times a solution is rounded by bounding the recursion depth (or depth of the dynamic programming tree). It is not straightforward to do so for other metrics. In Chapter 8, we were able to bound the number of times a solution is rounded in tree instances by instead bounding the complexity of the solution

structure. Perhaps such a strategy could extend to other metrics.

9.1.3 Multiple Constraints

In Chapter 8 we presented a framework that can be applied to a wide variety of vehicle and fleet constraints. But what if multple constraints are imposed? For example, what if *both* vehicle capacities *and* vehicle distance constraints are considered? Nagarajan and Ravi showed that in general metrics, given an α-approximation for Distance-Constrained Vehicle Routing, an $(\alpha + 1)$-approximation for the splittable-demand setting and an $(\alpha + 2)$-approximation for the unsplittable-demand setting of Distance-Constrained Capacitated Vehicle Routing are straightforward [84]. That is to say that the current best-known approximations for Distance-Constrained Vehicle Routing are so large that the extra cost of adding capacity constraints is negligible. But, as these approximation ratios are improved, thoughtful strategies for combining such constraints will become necessary.

9.2 Closing Remarks

Finally, recall that the goal of this thesis is to design provable approximation guarantees for facility placement and vehicle-routing problems with real-world constraints in metrics that model real-world transportation metrics. With this in mind, we would like our work to have real-world applicability.

As is, our results are of theoretical importance, but the runtimes are not necessarily practical. We believe, however, that the ideas introduced here can be used to strengthen real-world solutions. In particular, careful consideration of the specific, often simple, structure of transportation networks can greatly improve algorithm performance.

Finally, our methods can also be used in conjunction with heuristic approaches. Our approximation algorithms offer provable performance guarantees with which to better evaluate heuristics, and the insights from our algorithms can be used to inform heuristic design.

Bibliography

[1] Ittai Abraham, Daniel Delling, Amos Fiat, Andrew V. Goldberg, and Renato F. Werneck. VC-Dimension and Shortest Path Algorithms. In *International Colloquium on Automata, Languages, and Programming*, pages 690–699. Springer, 2011.

[2] Ittai Abraham, Daniel Delling, Amos Fiat, Andrew V. Goldberg, and Renato F. Werneck. Highway Dimension and Provably Efficient Shortest Path Algorithms. *Journal of the ACM*, 63(5):41:1–41:26, 2016.

[3] Ittai Abraham, Amos Fiat, Andrew V. Goldberg, and Renato F. Werneck. Highway Dimension, Shortest Paths, and Provably Efficient Algorithms. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 782–793. Society for Industrial and Applied Mathematics, 2010.

[4] Anna Adamaszek, Artur Czumaj, and Andrzej Lingas. PTAS for k-Tour Cover Problem on the Plane for Moderately Large Values of k. In *International Symposium on Algorithms and Computation*, pages 994–1003. Springer, 2009.

[5] Kemal Altinkemer and Bezalel Gavish. Heuristics for Unequal Weight Delivery Problems with a Fixed Error Guarantee. *Operations Research Letters*, 6(4):149–158, 1987.

[6] Sanjeev Arora. Polynomial Time Approximation Schemes for Euclidean TSP and Other Geometric Problems. In *37th Annual Symposium on Foundations of Computer Science*, pages 2–11. IEEE, 1996.

[7] Sanjeev Arora, Michelangelo Grigni, David R. Karger, Philip N. Klein, and Andrzej Woloszyn. A Polynomial-Time Approximation Scheme for Weighted Planar Graph TSP. In *Symposium on Discrete Algorithms*, volume 98, pages 33–41, 1998.

[8] Sanjeev Arora, Prabhakar Raghavan, and Satish Rao. Approximation Schemes for Euclidean k-Medians and Related Problems. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing (STOC)*, pages 106–113, 1998.

[9] Vijay Arya, Naveen Garg, Rohit Khandekar, Adam Meyerson, Kamesh Munagala, and Vinayaka Pandit. Local Search Heuristics for k-Median and Facility Location Problems. *SIAM Journal on Computing*, 33(3):544–562, 2004.

[10] Tetsuo Asano, Naoki Katoh, and Kazuhiro Kawashima. A New Approximation Algorithm for the Capacitated Vehicle Routing Problem on a Tree. *Journal of Combinatorial Optimization*, 5(2):213–231, 2001.

[11] 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 *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 275–283. ACM, 1997.

[12] Mihai Bādoiu, Sariel Har-Peled, and Piotr Indyk. Approximate Clustering via Core-sets. In *Proceedings of the Thiry-Fourth Annual ACM Symposium on Theory of Computing*, pages 250–257. ACM, 2002.

[13] Brenda S Baker. Approximation Algorithms for NP-Complete Problems on Planar Graphs. *Journal of the ACM (JACM)*, 41(1):153–180, 1994.

[14] Roberto Baldacci, Aristide Mingozzi, and Roberto Roberti. Recent Exact Algorithms for Solving the Vehicle Routing Problem Under Capacity and Time Window Constraints. *European Journal of Operational Research*, 218(1):1–6, 2012.

[15] Roberto Baldacci, Paolo Toth, and Daniele Vigo. Exact Algorithms for Routing Problems under Vehicle Capacity Constraints. *Annals of Operations Research*, 175(1):213–245, 2010.

[16] Judit Barilan, Guy Kortsarz, and David Peleg. How to Allocate Network Centers. *Journal of Algorithms*, 15(3):385–415, 1993.

[17] Yair Bartal. Probabilistic Approximations of Metric Spaces and Its Algorithmic Applications. In *37th Annual Symposium on Foundations of Computer Science*, pages 184–193, 1996.

[18] Holger Bast, Stefan Funke, and Domagoj Matijevic. Ultrafast Shortest-Path Queries via Transit Nodes. In *The Shortest Path Problem*, pages 175–192, 2006.

[19] Holger Bast, Stefan Funke, Domagoj Matijevic, Peter Sanders, and Dominik Schultes. In Transit to Constant Time Shortest-Path Queries in Road Networks. In *Proceedings of the Meeting on Algorithm Engineering and Experiments*, pages 46–59. Society for Industrial and Applied Mathematics, 2007.

[20] Amariah Becker, Philip N. Klein, and David Saulpic. A Quasi-Polynomial-Time Approximation Scheme for Vehicle Routing on Planar and Bounded-Genus Graphs. In *25th Annual European Symposium on Algorithms (ESA 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[21] Adrian Bock, Elyot Grant, Jochen Könemann, and Laura Sanità. The School Bus Problem on Trees. *Algorithmica*, 67(1):49–64, Sep 2013. URL: `https://doi.org/10.1007/s00453-012-9711-x`, `doi:10.1007/s00453-012-9711-x`.

[22] Hans L. Bodlaender, Daniel Lokshtanov, and Eelko Penninkx. Planar Capacitated Dominating Set is W[1]-Hard. In *International Workshop on Parameterized and Exact Computation*, pages 50–60. Springer, 2009.

[23] Agustín Bompadre, Moshe Dror, and James B. Orlin. Improved Bounds for Vehicle Routing Solutions. *Discrete Optimization*, 3(4):299–316, 2006.

[24] Glencora Borradaile and Hung Le. Optimal Dynamic Program for r-Domination Problems over Tree Decompositions. In *11th International Symposium on Parameterized and Exact Computation*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[25] Jose Caceres-Cruz, Pol Arias, Daniel Guimarans, Daniel Riera, and Angel A. Juan. Rich Vehicle Routing Problem: Survey. *ACM Computing Surveys (CSUR)*, 47(2):32, 2015.

[26] Stefan Cardon, Sander Dommers, Ceyhun Eksin, René Sitters, André Stougie, Leen Stougie, et al. A PTAS for the Multiple Depot Vehicle Routing Problem. *SPOR Report: Reports in Statistics, Probability and Operations Research*, 3, 2008.

[27] A. Chakrabarti, A. Jaffe, J. R. Lee, and J. Vincent. Embeddings of Topological Graphs: Lossy Invariants, Linearization, and 2-Sums. In *49th Annual IEEE Symposium on Foundations of Computer Science*, pages 761–770, 2008.

[28] Lin Chen and Daniel Marx. Covering a Tree with Rooted Subtrees–Parameterized and Approximation Algorithms. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2801–2820. SIAM, 2018.

[29] Miroslav Chlebík and Janka Chlebíková. Approximation Hardness of Dominating Set Problems in Bounded Degree Graphs. *Information and Computation*, 206(11):1264–1275, 2008.

[30] Nicos Christofides. Worst-Case Analysis of a New Heuristic for the Travelling Salesman Problem. Technical report, Carnegie-Mellon University Pittsburgh PA Management Sciences Research Group, 1976.

[31] Vincent Cohen-Addad, Éric Colin de Verdière, Philip N. Klein, Claire Mathieu, and David Meierfrankenfeld. Approximating Connectivity Domination in Weighted Bounded-Genus Graphs. In *Symposium on Theory of Computing*, pages 584–597, 2016.

[32] Vincent Cohen-Addad, Philip N. Klein, and Claire Mathieu. Local Search Yields Approximation Schemes for k-Means and k-Median in Euclidean and Minor-Free Metrics. In *Proceedings of the 57th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2016.

[33] William Cook and Paul Seymour. Tour Merging via Branch-Decomposition. *INFORMS Journal on Computing*, 15(3):233–248, 2003.

[34] Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*, volume 3, chapter 7: Treewidth. Springer, 2015.

[35] George B. Dantzig and John H. Ramser. The Truck Dispatching Problem. *Management Science*, 6(1):80–91, 1959.

[36] Aparna Das and Claire Mathieu. A Quasi-Polynomial Time Approximation Scheme for Euclidean Capacitated Vehicle Routing. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 390–403. SIAM, 2010.

[37] Erik D. Demaine, Fedor V. Fomin, Mohammadtaghi Hajiaghayi, and Dimitrios M. Thilikos. Fixed-Parameter Algorithms for (k,r)-Center in Planar Graphs and Map Graphs. *ACM Transactions on Algorithms (TALG)*, 1(1):33–47, 2005.

[38] Michael Dom, Daniel Lokshtanov, Saket Saurabh, and Yngve Villanger. Capacitated Domination and Covering: A Parameterized Perspective. In *International Workshop on Parameterized and Exact Computation*, pages 78–90. Springer, 2008.

[39] Frederic Dorn, Eelko Penninkx, Hans L. Bodlaender, and Fedor V. Fomin. Efficient Exact Algorithms on Planar Graphs: Exploiting Sphere Cut Branch Decompositions. In *European Symposium on Algorithms*, pages 95–106. Springer, 2005.

[40] David Eisenstat, Philip N. Klein, and Claire Mathieu. Approximating k-Center in Planar Graphs. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 617–627. Society for Industrial and Applied Mathematics, 2014.

[41] David Eppstein. Dynamic Generators of Topologically Embedded Graphs. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 599–608. Society for Industrial and Applied Mathematics, 2003.

[42] Guy Even, Naveen Garg, Jochen Könemann, R. Ravi, and Amitabh Sinha. Min-max Tree Covers of Graphs. *Operations Research Letters*, 32(4):309–315, July 2004. URL: `http://dx.doi.org/10.1016/j.orl.2003.11.010`, `doi:10.1016/j.orl.2003.11.010`.

[43] Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A Tight Bound on Approximating Arbitrary Metrics by Tree Metrics. *Journal of Computer and System Sciences*, 69(3):485–497, 2004.

[44] Tomás Feder and Daniel Greene. Optimal Algorithms for Approximate Clustering. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing (STOC)*, 1988.

[45] Uriel Feige. A Threshold of $\ln n$ for Approximating Set Cover. *Journal of the ACM (JACM)*, 45(4):634–652, 1998.

[46] Andreas Emil Feldmann. Fixed Parameter Approximations for k-Center Problems in Low Highway Dimension Graphs. In *Proceedings, Part II, of the 42nd International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 588–600. Springer-Verlag, 2015.

[47] Andreas Emil Feldmann, Wai Shing Fung, Jochen Könemann, and Ian Post. A (1+ ε)-Embedding of Low Highway Dimension Graphs into Bounded Treewidth Graphs. In *International Colloquium on Automata, Languages, and Programming*, pages 469–480. Springer, 2015.

[48] Eli Fox-Epstein, Philip N. Klein, and Aaron Schild. Embedding Planar Graphs into Low-Treewidth Graphs with Applications to Efficient Approximation Schemes for Metric Problems. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1069–1088. SIAM, 2019.

[49] Zachary Friggstad and Chaitanya Swamy. Approximation Algorithms for Regret-Bounded Vehicle Routing and Applications to Distance-Constrained Vehicle Routing. In *Proceedings of the Forty-Sixth Annual ACM Symposium on Theory of Computing*, STOC '14, pages 744–753, New York, NY, USA, 2014. ACM. URL: `http://doi.acm.org/10.1145/2591796.2591840`, `doi:10.1145/2591796.2591840`.

[50] Zachary Friggstad and Chaitanya Swamy. Compact, Provably-Good LPs for Orienteering and Regret-Bounded Vehicle Routing. In Friedrich Eisenbrand and Jochen Koenemann, editors, *Integer Programming and Combinatorial Optimization*, pages 199–211, Cham, 2017. Springer International Publishing.

[51] Satoshi Fujita. Vertex Domination in Dynamic Networks. In *WALCOM: Algorithms and Computation*, pages 1–12. Springer, 2008.

[52] Bruce L. Golden and Richard T. Wong. Capacitated Arc Routing Problems. *Networks*, 11(3):305–315, 1981.

[53] Teofilo F. Gonzalez. Clustering to Minimize the Maximum Intercluster Distance. *Theoretical Computer Science*, 38:293–306, 1985.

[54] Michelangelo Grigni, Elias Koutsoupias, and Christos Papadimitriou. An Approximation Scheme for Planar Graph TSP. In *36th Annual Symposium on Foundations of Computer Science*, pages 640–645. IEEE, 1995.

[55] Sudipto Guha and Samir Khuller. Greedy Strikes Back: Improved Facility Location Algorithms. *Journal of Algorithms*, 31(1):228–248, 1999.

[56] Anupam Gupta, Robert Krauthgamer, and James R. Lee. Bounded Geometries, Fractals, and Low-Distortion Embeddings. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 534–543. IEEE, 2003.

[57] Venkatesan Guruswami and Piotr Indyk. Embeddings and Non-Approximability of Geometric Problems. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, 2003.

[58] Mark Haimovich and A.H.G. Rinnooy Kan. Bounds and Heuristics for Capacitated Routing Problems. *Mathematics of Operations Research*, 10(4):527–542, 1985.

[59] Shin-ya Hamaguchi and Naoki Katoh. A Capacitated Vehicle Routing Problem on a Tree. In *International Symposium on Algorithms and Computation*, pages 399–407. Springer, 1998.

[60] Dorit S. Hochbaum and David B. Shmoys. A Best Possible Heuristic for the k-Center Problem. *Mathematics of Operations Research*, 10(2):180–184, 1985.

[61] Dorit S. Hochbaum and David B. Shmoys. Using Dual Approximation Algorithms for Scheduling Problems Theoretical and Practical Results. *Journal of the ACM (JACM)*, 34(1):144–162, 1987.

[62] Stefan Irnich, Paolo Toth, and Daniele Vigo. Chapter 1: The Family of Vehicle Routing Problems. In Paolo Toh and Daniele Vigo, editors, *Vehicle Routing: Problems, Methods, and Applications*. SIAM, 2014.

[63] Kamal Jain and Vijay V. Vazirani. Approximation Algorithms for Metric Facility Location and k-Median Problems Using the Primal-Dual Schema and Lagrangian Relaxation. *Journal of the ACM*, 48(2):274–296, 2001.

[64] David S. Johnson. Approximation Algorithms for Combinatorial Problems. *Journal of Computer and System Sciences*, 9(3):256–278, 1974.

[65] Mong-Jen Kao and Han-Lin Chen. Approximation Algorithms for the Capacitated Domination Problem. In *International Workshop on Frontiers in Algorithmics*, pages 185–196. Springer, 2010.

[66] Mong-Jen Kao and DT Lee. Capacitated Domination: Constant Factor Approximations for Planar Graphs. In *International Symposium on Algorithms and Computation*, pages 494–503. Springer, 2011.

[67] Mong-Jen Kao, Chung-Shou Liao, and DT Lee. Capacitated Domination Problem. *Algorithmica*, 60(2):274–300, 2011.

[68] Marek Karpinski, Michael Lampis, and Richard Schmied. New Inapproximability Bounds for TSP. *Journal of Computer and System Sciences*, 81(8):1665–1677, 2015.

[69] Yoshiyuki Karuno, Hiroshi Nagamochi, and Toshihide Ibaraki. Vehicle Scheduling on a Tree with Release and Handling Times. *Annals of Operations Research*, 69:193–207, 1997.

[70] Ioannis Katsikarelis, Michael Lampis, and Vangelis Th. Paschos. Structural Parameters, Tight Bounds, and Approximation for (k, r)-Center. In *28th International Symposium on Algorithms and Computation (ISAAC)*, pages 50:1–50:13, 2017. URL: `https://doi.org/10.4230/LIPIcs.ISAAC.2017.50`, `doi:10.4230/LIPIcs.ISAAC.2017.50`.

[71] Michael Khachay and Roman Dubinin. PTAS for the Euclidean Capacitated Vehicle Routing Problem in $\mathbb{R}^d$. In *International Conference on Discrete Optimization and Operations Research*, pages 193–205. Springer, 2016.

[72] Michael Khachay and Helen Zaytseva. Polynomial Time Approximation Scheme for Single-Depot Euclidean Capacitated Vehicle Routing Problem. In *Combinatorial Optimization and Applications*, pages 178–190. Springer, 2015.

[73] Samir Khuller and Yoram J. Sussmann. The Capacitated k-Center Problem. *SIAM Journal on Discrete Mathematics*, 13(3):403–418, 2000.

[74] Martine Labbé, Gilbert Laporte, and Hélene Mercure. Capacitated Vehicle Routing on Trees. *Operations Research*, 39(4):616–622, 1991.

[75] Gilbert Laporte. What You Should Know About the Vehicle Routing Problem. *Naval Research Logistics (NRL)*, 54(8):811–819, 2007.

[76] Gilbert Laporte, Michel Gendreau, Jean-Yves Potvin, and Frédéric Semet. Classical and Modern Heuristics for the Vehicle Routing Problem. *International Transactions in Operational Research*, 7(4-5):285–300, 2000.

[77] Shi Li and Ola Svensson. Approximating k-Median via Pseudo-Approximation. *SIAM Journal on Computing*, 45(2):530–547, 2016.

[78] Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Known Algorithms on Graphs of Bounded Treewidth Are Probably Optimal. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 777–789. Society for Industrial and Applied Mathematics, 2011.

[79] Robert Lupton, F. Miller Maley, and Neal Young. Data Collection for the Sloan Digital Sky Survey—A Network-Flow Heuristic. *Journal of Algorithms*, 27(2):339–356, 1998.

[80] Dániel Marx and Michał Pilipczuk. Optimal Parameterized Algorithms for Planar Facility Location Problems Using Voronoi Diagrams. In *European Symposium on Algorithms*, pages 865–877. Springer, 2015.

[81] Marjan Marzban and Qian-Ping Gu. Computational Study on a PTAS for Planar Dominating Set Problem. *Algorithms*, 6(1):43–59, 2013.

[82] Hiroshi Nagamochi. Approximating the Minmax Rooted-Subtree Cover Problem. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 88(5):1335–1338, 2005.

[83] Hiroshi Nagamochi and Kohei Okada. Approximating the Minmax Rooted-Tree Cover in a Tree. *Information Processing Letters*, 104(5):173–178, 2007.

[84] Viswanath Nagarajan and R. Ravi. Approximation Algorithms for Distance Constrained Vehicle Routing Problems. *Networks*, 59(2):209–214, 2012.

[85] Tim Nieberg and Johann Hurink. A PTAS for the Minimum Dominating Set Problem in Unit Disk Graphs. In *Approximation and Online Algorithms*, pages 296–306. Springer, 2005.

[86] Christos H. Papadimitriou and Mihalis Yannakakis. The Traveling Salesman Problem with Distances One and Two. *Mathematics of Operations Research*, 18(1):1–11, 1993.

[87] Ján Plesník. On the Computational Complexity of Centers Located in a Graph. *Aplikace matematiky*, 25(6):445–452, 1980.

[88] Neil Robertson and Paul D. Seymour. Graph Minors. X. Obstructions to Tree-Decomposition. *Journal of Combinatorial Theory, Series B*, 52(2):153–190, 1991.

[89] Sartaj K. Sahni. Algorithms for Scheduling Independent Tasks. *Journal of the ACM (JACM)*, 23(1):116–127, 1976.

[90] Boston Public Schools. Transportation challenge: Solving routing and bell times. URL: `https://www.bostonpublicschools.org/transportationchallenge`.

[91] David B. Shmoys, Éva Tardos, and Karen Aardal. Approximation Algorithms for Facility Location Problems. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 265–274. ACM, 1997.

[92] K. Talwar. Bypassing the Embedding: Algorithms for Low Dimensional Metrics. In *36th Annual ACM Symposium on Theory of Computing*, pages 281–290, 2004.

[93] Mikkel Thorup. Compact Oracles for Reachability and Approximate Distances in Planar Digraphs. *Journal of the ACM*, 51(6):993–1024, 2004.

[94] Eduardo Uchoa, Diego Pecin, Artur Pessoa, Marcus Poggi, Thibaut Vidal, and Anand Subramanian. New Benchmark Instances for the Capacitated Vehicle Routing Problem. *European Journal of Operational Research*, 257(3):845–858, 2017.

[95] Feng Wang, Erika Camacho, and Kuai Xu. Positive Influence Dominating Set in Online Social Networks. In *Combinatorial Optimization and Applications*, pages 313–321. Springer, 2009.

[96] Liang Xu, Zhou Xu, and Dongsheng Xu. Exact and Approximation Algorithms for the Min–Max k-Traveling Salesmen Problem on a Tree. *European Journal of Operational Research*, 227(2):284–292, 2013.