CARNEGIE MELLON UNIVERSITY

DOCTORAL THESIS

Algorithms for Ranking and Routing Problems

Author:
Yang JIAO

Supervisor:
R. RAVI

Dissertation Committee:
R. Ravi (Chair)
Gérard Cornuéjols
Willem-Jan van Hoeve
Ojas Parekh
Cynthia Phillips

A thesis submitted in fulfillment of the requirements for the degree of Doctor of Philosophy

in the

Algorithms, Combinatorics, and Optimization
Tepper School of Business, Carnegie Mellon University

September 7, 2018

Declaration of Authorship

I, Yang JIAO, declare that this thesis titled, "Algorithms for Ranking and Routing Problems" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

CARNEGIE MELLON UNIVERSITY

Abstract

Tepper School of Business, Carnegie Mellon University

Doctor of Philosophy

Algorithms for Ranking and Routing Problems

by Yang JIAO

In this thesis, we develop new algorithms for two categories of problems–ranking and routing. In the first category, we classify the complexity of new problems for mutually ranking participants and tasks when a ranking of the participants “nearby" the optimal is given. In the second category, we give approximation algorithms for inventory routing on line metrics, two new variants of inventory routing, and provide fast heuristics for inventory routing on arbitrary metrics.

For the first category, we introduce and resolve the computational complexity of a set of new problems that require ranking participants and tasks by their strengths and difficulties, respectively, given the set of tasks that each participant completed successfully. The ideal ranking ensures that stronger participants succeed at all tasks that weaker participants performed successfully and easier tasks are performed successfully by all the participants who succeeded at harder tasks. The new variants we introduce and study account for recurring participants, by constraining the outcome of the current ranking to be close to an initial given ranking of the participants arrived from past contests. We provide a comprehensive study of the complexity of all the variants.

The second category involves three sets of routing problems. The first among them is the Inventory Routing Problem (IRP). Given clients on a metric, each with daily demands that must be delivered from a depot, and holding costs over the planning horizon, a solution selects a set of daily routes from the depot through a subset of clients to deliver all demands before they are due. The cost of the solution combines the costs of the routes with the holding cost of the demand that arrives earlier at clients. For Inventory Routing on line metrics, we give a constant approximation algorithms by LP rounding and a primal dual method. We also study the computational aspect of IRP on general metrics. We design fast combinatorial heuristics for IRP by connecting them to prize-collecting vehicle routing problems and evaluate their performance on randomly generated data sets. The second variant of IRP we study is called Deadline IRP. In this version, every client has a deadline within which it will run out if it starts at full capacity, and each visit to every client fills the client location to capacity. The goal is to determine an IRP solution so that no client ever runs out. We provide logarithmic approximations and show a class of instances for which our method cannot improve the approximation factor. The third set of routing problems we study are variants of Inventory Routing with Facility Location, which allows multiple depot locations to be opened for service at an extra cost. We provide a 12-approximation for Star Inventory Routing with Facility Location assuming that clients connect directly to the opened facilities.

Acknowledgements

First, I am deeply grateful to my advisor R. Ravi, whose continuous enthusiasm has been an inspiration to me for the past five years. I am indebted to the generous time and effort he has spent developing my research skills and communication skills. His sharp intuition for a wide range of problems has helped me develop better ways to visualize and solve new problems. I also appreciate Ravi's approachability and willingness to meet beyond our regularly scheduled hours whenever I needed. His abundant and prompt feedback on my many drafts and talks has significantly improved my ability to communicate technical ideas clearly. Ravi has encouraged and supported me to attend workshops/conferences and present my research at every opportunity. I cannot thank him enough for all he has done for my development.

Additionally, I would like to thank Ojas Parekh for hosting me at Sandia and introducing me to the emerging field of quantum computing. Ojas's excitement for quantum computing has positively influenced me to keep an eye out for new areas related to my research. I also appreciate him kindly taking me to the many hospital visits during the occurrence of my health problems at Sandia. During my internship with Ojas, I was fortunate to have the opportunity to have met Cindy Phillips. I would like to thank her for her professional advice and for inviting me to present my work at SIAM.

During my work on the ranking related problems, I had the opportunity to collaborate with Wolfgang Gatterbauer. I would like to thank him for the discussions and the detailed feedback he gave on my paper and presentation.

Also, I would like to thank all of my committee members, R. Ravi, Gérard Cornuéjols, Willem-Jan van Hoeve, Ojas Parekh, and Cindy Phillips, for providing valuable feedback on my drafts and talks.

Before I first came into the field, I had an excellent teacher, Rajiv Gandhi, who kindled my love for algorithms. I thank him for believing in the best of all his students and for inspiring me to delve deeper into theoretical computer science.

I extend my thanks to Lawrence Rapp and Laila Lee, who helped the PhD process run smoothly.

Finally, I would like to thank my family, friends, and peers for keeping me motivated to the end. In particular, thanks to Anuj, Arash, Christian, Dabeen, Eric, Gerdus, Jenny, Ji, John, Nam, Ryo, Sagnik, Shirley, Stelios, Thiago, Thomas, and Wenting for their helpful discussions and friendship. I thank my parents for their unwavering love and support throughout my life.

Contents

List of Figures

Dedicated to my parents

Chapter 1

Introduction

Motivation

We study a set of ranking problems and three types of routing problems that involve coordinating schedules with the routes. The ranking problems arise from the context of Massive Open Online Courses, where the large number of students make it challenging to create appropriate ways to evaluate students such that their results can all be graded efficiently for timely feedback. One way to speed up the process is to restrict the test questions to automatically gradable questions e.g. multiple choice. Furthermore, to quickly create a large number of test questions, we consider crowd-sourcing the creation of test questions to the students. However, giving role of question creation to the students means that the instructor no longer knows the difficulty of the questions. So we would like to find ways to quickly rank the difficulty of the newly created questions as well as the strength of the students after they attempt all the questions. Chapter 2 models variants of the this ranking problem and resolves their complexity.

Routing problems with scheduling components have been studied extensively on the computational side, but are fairly new within the theoretical realm. Although a rich history of theoretical methods exist for network design problems (the routing component), the integration of routing with scheduling create an interesting challenge to obtaining theoretical guarantees. We study variants of a popular inventory routing problem in this thesis [80]. We consider the deterministic inventory routing problem over a discrete finite time horizon. Given clients on a metric, each with daily demands that must be delivered from a depot and holding costs over the planning horizon, an optimal solution selects a set of daily tours through a subset of clients to deliver all demands before they are due and minimizes the total holding and tour routing costs over the horizon. In particular, the best approximation for Inventory Routing Problem (on general metrics) has yet to break a logarithmic factor. It is also unknown whether this problem can be shown to have a super constant hardness of approximation. Constant factor guarantees have only been shown for special cases such as when the metric is a tree or when the schedule is restricted to be periodic. In the subsequent chapters, we extend LP-based methods to variants of Inventory Routing and give fast heuristics that obtain near optimal solutions on randomly generated data sets.

Summary

Ranking Problems The first category of problems we study are the ranking problems of Chapter 2. Here, we introduce a new set of problems based on the *Chain Editing problem*. In our version of Chain Editing, we are given a set of participants and a set of tasks that every participant attempts. For each participant-task pair, we know whether the participant has succeeded at the task or not. We assume that participants vary in their ability to solve tasks, and that tasks vary in their difficulty to be solved. In an ideal world, stronger participants should succeed at a superset

of tasks that weaker participants succeed at. Similarly, easier tasks should be completed successfully by a superset of participants who succeed at harder tasks. In reality, it can happen that a stronger participant fails at a task that a weaker participants succeeds at. Our goal is to find a *perfect nesting of the participant-task relations* by flipping a minimum number of participant-task relations, implying such a "nearest perfect ordering" to be the one that is closest to the truth of participant strengths and task difficulties. Many variants of the problem are known to be NP-hard.

We propose six natural k-near versions of the Chain Editing problem and classify their complexity. The input to a k-near Chain Editing problem includes an initial ordering of the participants (or tasks) that the final solution is required to be "close" to, by moving each participant (or task) at most k positions from the initial ordering. We obtain surprising results on the complexity of the six k-near problems: Five of the problems are polynomial-time solvable using dynamic programming, but one of them is NP-hard.

Inventory Routing Problem The second category of problems consists of three types of routing problems. The first among them is inventory routing, introduced in Chapter 3. For the special case that the metric is a line, we obtain a 5-approximation using LP rounding and a 26-approximation using a primal dual algorithm.

Besides theoretical guarantees for the special case on line metrics, we study the computational side of IRP arbitrary metrics in Chapter 4. In particular, we provide fast heuristics for IRP on general metrics that uses a Prize-Collecting Steiner Tree subproblem to guide the inventory routing solution to near optimality. Our best heuristic solves instances of at least 160 clients over 6 days and 50 clients over 18 days to near optimality in a few seconds. It is three orders of magnitude faster than solving the single commodity flow MIP formulation using Gurobi cut off at 10% MIPGap.

Deadline Inventory Routing Problem The second type of routing problem, introduced in Chapter 5, is called Deadline Inventory Routing, motivated by the replenishment of ATMs. In *Deadline Inventory Routing*, every client has a deadline within which it will run out if it starts at full capacity, and each visit to every client fills the client location to capacity. The goal is to determine a set of routes, one for each day, such that no client ever runs out. We show a $\log(n)$-approximation, where n is the number of clients. To obtain the logarithmic approximation, we first reduce from arbitrary instances to a subclass of instances where each deadline is a power of 2 by losing a constant factor in the cost of the solution. Within powers-of-2 instances, a natural (though possibly costly) solution is one that visits on day l exactly those clients whose deadlines 2^i divide l (i.e., l is a multiple of 2^i). We call such a solution *synchronized* since it tries to group together as many clients as possible who are appropriate to visit per day. Next, we introduce *nondecreasing* solutions, which are those whose route per day must visit clients in nondecreasing order of their deadline values. We show that nondecreasing solutions have cost at most a logarithmic factor away from the cost of any arbitrary solution. Then, we show that synchronized solutions are derivable from nondecreasing solutions preserving the exact cost. So optimal synchronized solutions are also logarithmic factor from arbitrary solutions. Since the set of clients to visit per day is completely determined in synchronized solutions, optimal synchronized solutions are easy to approximate by approximating Steiner trees. Hence we obtain an $O(\log n)$-approximation for Stationary Deadline Inventory Routing. We show that the analysis for this method is tight on an infinite class of instances. Using an LP-based approach, we also obtain a $\log(T)$-approximation, where T is the number of days in the time horizon.

Inventory Routing Problem with Facility Location The third type of routing problem is called Star Inventory Routing Problem with Facility Location (SIRPFL), which we study in Chapter 6. As a stepping stone to solving SIRPFL, we first study the Inventory Access Problem (IAP), which is the single depot, single client special case of IRP. We provide a simple dynamic program for Uncapacitated IAP and an NP-hardness reduction for Capacitated IAP where each demand cannot be split among different trips. Next, we study SIRPFL, which involves the extra decisions of where to open facilities and which opened facility satisfies each demand. As is the case for Facility Location, we assume that the connections per day are directly built between opened facilities and clients, i.e., the client-facility connections form stars centered at facilities. For Uncapacitated Star SIRPFL, we provide a 12-approximation by rounding an LP relaxation.

Main Contributions

- In Chapter 2, we introduced a set of ranking problems motivated by the need to efficiently rank a large number of participants while having some history of how they have performed. We resolved their computational complexity, including a surprising NP-hardness result despite the similarity of the problems. Intuitively, the problem that turned out to be NP-hard could not be solved with similar methods due to its less constrained nature, which makes it harder to infer the correct ordering.

- For Inventory Routing on line metrics, we proved a constant factor guarantee by carefully pruning an initial solution from a primal dual method in Chapter 3. The initial solution from the primal dual phase has cost arbitrarily far from the optimal cost. By partitioning the line into regions at distances of powers of 2 away from the depot, we were able to modify the visits so that each region's LP values are charged only a constant number of times. On general metrics, the difficulty remains in finding a pruning procedure that facilitates a consistent charging scheme.

- Following the theoretical study, we provided near optimal, fast heuristics for general IRP by creatively using PCST to form portions of the final IRP solution in Chapter 4. In particular, we designed a set of local search heuristics where we reduce each large neighborhood search to solving a Prize-Collecting Steiner Tree (PCST) instance. Also, we provide a set-covering based greedy heuristic that chooses the sets by solving an appropriately constructed PCST. Finally, we give a primal dual heuristic that uses PCST to determine the visit sets during the growth of the dual variables. Our best heuristics solved the test instances within 1.08 factor of the optimal cost and performed three orders of magnitude faster than standard MIP solvers.

- In Chapter 5, we introduced a new inventory replenishment problem motivated by the delivery of cash to ATMs. We provided two logarithmic approximations and a class of instances where one of the methods cannot be improved.

- Finally, for SIRPFL, we extend rounding ideas for Facility Location, accounting for scheduling aspect of satisfying demands, to obtain a constant approximation in Chapter 6.

Road Map

The subsequent chapters are organized as follows. In Chapter 2, we introduce new problems involving ranking participants and tasks and classify the complexity of each problem. Chapter 3 discusses our approximation algorithms for IRP on

line metrics. In Chapter 4, we give three combinatorial heuristics that utilize Prize-Collecting Steiner Tree solutions to find fast near optimal solutions for IRP. In Chapter 5, we propose a new variant of IRP called Deadline IRP and state our approaches. Chapter 6 introduces another variant of IRP, Star IRP with Facility Location and provides a constant approximation by LP rounding.

Chapter 2

Algorithms for Automatic Ranking of Participants and Tasks in an Anonymized Contest

2.1 Introduction

2.1.1 Motivation

Consider a contest with a set S of participants who are required to complete a set Q of tasks. Every participant either succeeds or fails at completing each task. We aim to obtain rankings of the participants' strengths and the tasks' difficulties. This situation can be modeled by a bipartite graph with participants on one side, tasks on the other side, and edges present if a participant succeeded at the task. From the edges of the bipartite graph, we can infer that a participant a_2 is stronger than a_1 if the neighborhood of a_1 is strictly contained in (or is strictly "nested in") that of a_2. Similarly, we can infer that a task is easier than another if its neighborhood strictly contains that of the other. If two participants or tasks have the same neighborhood, then they are considered equally strong or equally easy. See Figure 2.1 for a visualization of strengths of participants and difficulties of tasks. If all neighborhoods are nested, then this nesting immediately implies a ranking of the participants and tasks. However, participants and tasks are not perfect in reality, which may result in a bipartite graph with "non-nested" neighborhoods. For such more realistic scenarios, we wish to determine a ranking of the participants and the tasks that is still "close" to the ideal case. In this chapter, we define several variants of this problem that are different in what changes can be made (adding, deleting, or adding and deleting edges) and prior knowledge of rankings (exact for one side, no prior knowledge, nearby starting values) that together give rise to varying problem complexities.

2.1.1.0.1 Relation to Truth Discovery.

A popular application of unbiased rankings is computational "truth discovery." *Truth discovery* is the determination of trustworthiness of conflicting pieces of information that are observed often from a variety of sources [78] and is motivated by the problem of extracting information from networks where the trustworthiness of the actors are uncertain [57]. The most basic model of the problem is to consider a bipartite graph where one side is made up of actors, the other side is made up of their claims, and edges denote associations between actors and claims. Furthermore, claims and actors are assumed to have "trustworthiness" and "believability" scores, respectively, with known a priori values. According to a number of recent

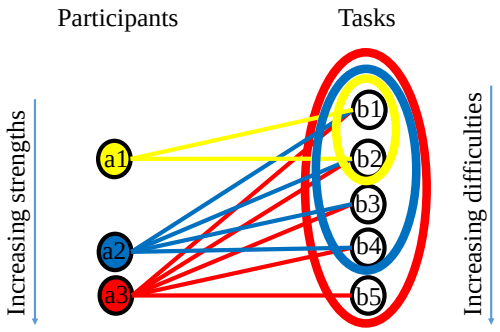


FIGURE 2.1: An "ideal" graph is shown. Participants and tasks may be interpreted as students and questions, or actors and claims. Participant a_1 succeeds at b_1 to b_2; a_2 succeeds at b_1 to b_4; a_3 succeeds at b_1 to b_5. The nesting of neighborhoods here indicate that participant a_1 is weaker than a_2, who is weaker than a_3, and task b_1 and b_2 are easier than b_3 and b_4, which in turn are easier than b_5.

surveys [57, 78, 70], common approaches for truth discovery include iterative procedures, optimization methods, and probabilistic graphic models. (1) Iterative methods [38, 50, 76, 84] update trust scores of actors to believability scores of claims, and vice versa, until convergence. Various variants of these methods (such as Hubs and Authorities (or Sums) [65], TruthFinder [84], AverageLog, Investment, and PooledInvestment [76]) have been extensively studied and proven in practice [3]. (2) Optimization methods [10, 69] aim to find truths that minimize the total distance between the provided claims and the output truths for some specified continuous distance function; coordinate descent [16] is often used to obtain the solution. (3) Probabilistic graphical models [77] of truth discovery are solved by expectation maximization. Other methods for truth discovery include those that leverage trust relationships between the sources [52]. Our study is conceptually closest to optimization approaches (we minimize the number of edge additions or edits), but we suggest a *discrete objective* for minimization, for which we need to develop new algorithms.

2.1.1.0.2 Our Motivation: Massively Open Online Courses.

Our interest in the problem arises from trying to model the problem of automatic grading of large number of students in the context of MOOCs (massively open online courses). Manual grading of assignments from many students is infeasible. In turn, creating many automatically gradable questions (that are also relevant to the topics of a class) is difficult. Our idea is to crowd-source the creation of automatically gradable questions (in particular, multiple choice items) to students, and have all the students take all questions. In this context, we do not know the difficulty of questions and would like to quickly compute a roughly accurate ordering of the difficulty of the crowd-sourced questions from the answers chosen by the students. Additionally, we also want to rank the strength of the students based on their performance. In an ideal world, stronger participants should succeed at a superset of tasks that weaker participants succeed at, which motivates our nesting property. In reality, it can happen that a stronger participant fails at a task that a weaker participants succeeds at. Our goal is to find a ranking of students and questions that "explains" our observations as much as possible and is thus a close to the ideal case as possible.

2.1.1.0.3 Our Model.

Henceforth, we refer to participants as students and tasks as questions in the rest of the chapter. We cast the ranking problem as a discrete optimization problem of minimizing the number of changes to a given record of the students' performance to obtain nested neighborhoods. This is called the *Chain Editing* problem. It is often possible that some information regarding the best ranking is already known. For instance, if the observed rankings of students on several previous assignments are consistent, then it is likely that the ranking on the next assignment will be similar. We model known information by imposing an additional constraint that the changes made to correct the errors to an ideal ranking must result in a ranking that is near a given base ranking. By near, we mean that the output position of each student should be within at most k positions from the position in the base ranking, where k is a parameter. Given a nearby ranking for the students, we consider all possible variants arising from how the question ranking is constrained. The question ranking may be constrained in one of the following three ways: (*i*) the exact question ranking is specified (which we term the "constrained" case), (*ii*) it must be near a given question ranking (the "both near" case), or (*iii*) the question ranking is unconstrained (the "unconstrained" case). We provide the formal definitions of these problems next.

2.1.2 Problem Formulations

Here, we define all variants of the ranking problem. The basic variants of Chain Editing are defined first and the k-near variants are defined afterward.

2.1.3 Basic Variants of Chain Editing

First, we introduce the problem of recognizing an "ideal" input. Assume that we are given a set S of students, and a set Q of questions. Every student attempts every question. Edges between S and Q indicate which questions the students answered correctly. Denote the resulting bipartite graph by $G = (S \cup Q, E)$. Let $n = |S| + |Q|$. For every pair $(s, q) \in S \times Q$, we are given an edge between s and q if and only if student s answered question q correctly.

For a graph (V, E), denote the neighborhood of a vertex x by $N(x) := \{y \in V : xy \in E\}$. In other words, the neighborhood of a question is the set of student who answered the question correctly. Similarly, the neighborhood of a student is the set of questions that the student answered correctly.

Strength and Difficulty We say that student s_1 is *stronger* than student s_2 if $N(s_1) \supset N(s_2)$, and student s_1 is *equivalent* to s_2 if $N(s_1) = N(s_2)$. We say that question q_1 is *harder* than question q_2 if $N(q_1) \subset N(q_2)$, and question q_1 is *equivalent* to question q_2 if $N(q_1) = N(q_2)$. Given an ordering α on the students and β on the questions, $\alpha(s_1) > \alpha(s_2)$ shall indicate that s_1 is stronger than s_2; $\beta(q_1) > \beta(q_2)$ shall indicate that q_1 is harder (more difficult) than q_2; $\alpha(s_1) = \alpha(s_2)$ and $\beta(q_1) = \beta(q_2)$ shall indicate that s_1 is equivalent to s_2 and q_1 is equivalent to q_2, respectively.

Interval and Nesting properties An ordering of the questions satisfies the *interval property* if for every student s, its neighborhood $N(s)$ consists of a block of consecutive questions (starting with the easiest question) with respect to the ordering of the questions. An ordering α of the students satisfies the *nesting property* if $\alpha(s_1) \geq \alpha(s_2) \Rightarrow N(s_1) \supseteq N(s_2)$.

Definition The objective of the *Ideal Mutual Orderings (IMO)* problem is to order the students and the questions so that they satisfy the interval and nesting properties respectively, or output NO if no such orderings exist.

Observe that IMO can be solved efficiently by comparing containment relation among the neighborhoods of the students and ordering the questions and students according to the containment order.

Proposition 2.1.1. *There is a polynomial time algorithm to solve IMO.*

Proof. Compare the neighborhood of every pair of students $\{s_1, s_2\} \subseteq S$ and check whether $N(s_1) \subseteq N(s_2)$ or $N(s_1) \supseteq N(s_2)$. If $N(s_1) \cap N(s_2)$ is a strict subset of $N(s_1)$ and $N(s_2)$, then output NO. Now, assuming that every pair $\{s_1, s_2\} \subseteq S$ satisfies $N(s_1) \subseteq N(s_2)$ or $N(s_1) \supseteq N(s_2)$, we know that there is an ordering $\alpha : S \to [|S|]$ such that $\alpha(s_1) \leq \alpha(s_2) \Rightarrow N(s_2) \subseteq N(s_2)$. We easily find such an ordering by sorting the students according to their degrees, i.e., from lowest to highest degree, the students will receive labels from the smallest to the largest. Denote the resulting ordering by π. Since all neighborhoods are subsets or supersets of any other neighborhood and π was sorted by degree, $\pi(s_1) \leq \pi(s_2) \Rightarrow N(s_1) \leq N(s_2)$. So we have satisfied the nesting property.

To satisfy the interval property, we order the questions according to the nesting of the neighborhoods. Recall that we have $N(\pi^{-1}(1)) \subseteq \cdots \subseteq N(\pi^{-1}(|S|))$. Now, we order the questions so that whenever $q_1 \in N(\pi^{-1}(i))$ and $q_2 \in N(\pi^{-1}(j))$ with $i < j$, we have q_1 labeled smaller q_2 according to the ordering. We can do so by labeling the questions in $N(\pi^{-1}(1))$ the smallest numbers (the ordering within the set does not matter), then the questions in $N(\pi^{-1}(2))$ the next smallest, and so on. Call the resulting ordering β. Note that for all $s \in S, s = \pi^{-1}(i)$ for some i. So $N(s) = N(\pi^{-1}(i)) \supseteq N(\pi^{-1}(1))$, i.e., s correctly answers the easiest question according to β. Furthermore, $N(s)$ is a block of questions that are consecutive according to the ordering β. So the interval property is also satisfied.

To determine the run time, note that we made $O(n^2)$ comparisons of neighborhoods. Each set intersection of two neighborhoods took $O(n)$ time assuming that each neighborhood was stored as a sorted list of the questions (sorted by any fixed labeling of the questions). Ordering the students by degree took $O(n \log n)$ time and ordering the questions took $O(n)$ time. So the total run time is $O(n^2)$. □

Next, observe that the nesting property on one side is satisfiable if and only if the interval property on the other side is satisfiable. Hence, we will require only the nesting property in subsequent variants of the problem.

Proposition 2.1.2. *A bipartite graph has an ordering of all vertices so that the questions satisfy the interval property if and only if it has an ordering with the students satisfying the nesting property.*

Proof. First, we prove the forward direction. Assume that $G = (S \cup Q, E)$ satisfies the interval property with respect to the ordering β on Q. By definition of interval property, for every $u \in S$, we have $N(u) = \{\beta^{-1}(1), \ldots, \beta^{-1}(j)\}$ for some $j \in [|Q|]$. Then for every $u_1, u_2 \in S$, we have $N(u_1) \subseteq N(u_2)$ or $N(u_2) \subseteq N(u_1)$. Let α be an ordering of S by degree of each $u \in S$. Then the nesting property holds with respect to α.

Second, we prove the backward direction. Assume that $G = (S \cup Q, E)$ satisfies the nesting property with respect to α on S. Then $N(\alpha^{-1}(1)) \subseteq \cdots \subseteq N(\alpha^{-1}(|S|))$. Using the algorithm in the proof of Proposition 2.1.1 for IMO, we obtain an ordering β on Q so that the interval property holds with respect to β. □

Next, we define three variants of IMO, which model the possible ways we would allow changes to the edges in the graph in order to achieve the nesting property: allowing edges to be added, or deleted, or both.

Chain Editing (CE) In the Chain Editing (CE) problem, we are given a bipartite graph representing student-question relations and asked to find a minimum set of edge edits that admits an ordering of the students satisfying the nesting property.

A more restrictive problem than Chain Editing is Chain Addition. Chain Addition is variant of Chain Editing that allows only edge additions and no deletions. Chain Addition models situations where students sometimes accidentally give wrong answers on questions that they know how to solve but never answer a hard problem correctly by luck, e.g., in numerical entry questions.

Chain Addition (CA) In the Chain Addition (CA) problem, we are given a bipartite graph representing student-question relations and asked to find a minimum set of edge additions that admits an ordering of the students satisfying the nesting property.

On the other hand, weak students may accidentally solve hard questions correctly when the questions are multiple choice or true/false. Chain Deletion models such situations.

Chain Deletion (CD) In the Chain Deletion (CD) problem, we are given a bipartite graph representing student-question relations and asked to find a minimum set of edge deletions that admits an ordering of the students satisfying the nesting property.

Among the three problems, Chain Addition and Chain Deletion are isomorphic, i.e., solving one enables us to solve the other. The key property that connects Chain Addition with Chain Deletion is that a graph satisfies the nesting property if and only if its complement satisfies the nesting property. To solve Chain Deletion on a graph G, consider the complement $\overline{G}$ of G and solve Chain Addition on $\overline{G}$. Let F be the set of edges in an optimal solution for Chain Addition on $\overline{G}$. By definition of complement, F must have been a subset of the edges in G. Since $\overline{G} \cup F$ satisfies the nesting property, its complement $\overline{\overline{G} \cup F} = G \setminus F$ must also satisfy the nesting property. So F is an optimal solution for Chain Deletion on G. A symmetric argument applies to solve Chain Addition from Chain Deletion. Since the addition and the deletion cases are isomorphic, we consider only the addition and the more general edition, which – together with the three constraint variants from subsection 2.1.1.0.3 – give rise to our 6 problem formulations.

Analogous to needing only to satisfy one of the two properties, it suffices to find an optimal ordering for only one side. Once one side is fixed, it is easy to find an optimal ordering of the other side respecting the fixed ordering.

Proposition 2.1.3. *In Chain Editing, if the best ordering (that minimizes the number of edge edits) for either students or questions is known, then the edge edits and ordering of the other side can be found in polynomial time.*

Proof. Consider the special case that one side of the correct ordering is given to us, say the questions are given in hardest to easiest order $v_1 \geq \cdots \geq v_q$. Then we can find the minimum number of errors needed to satisfy the required conditions by correcting the edges incident to each student u individually.

We know by the interval property that every student u must correctly answer either a set of consecutive questions starting from v_1 or no questions at all. For each $u \in S$, and for each v_j, simply compute the number of edge edits required so that the neighborhood of u becomes $\{v_1, \ldots, v_j\}$. Select the question v_u that minimizes the cost of enforcing $\{v_1, \ldots, v_j\}$ to be the neighborhood of u. Once the edges are corrected, order the students by the containment relation of their neighborhoods.

The algorithm correctly calculates the minimum edge edits since the interval property was satisfied at the minimum cost possible per student. The algorithm finds the neighborhood of each student by trying at most $|Q| < n$ difficulty thresholds v_j, and the cost of calculation for each threshold takes $O(1)$, by using the value calculated from the previous thresholds tried. Summing over the $|S| < n$ students gives a total running time no more than $O(n^2)$. □

2.1.3.0.1 *k*-near Variants of Chain Editing or Addition

We introduce and study the nearby versions of Chain Editing or Chain Addition. Our problem formulations are inspired by Balas and Simonetti's [11] work on *k*-near versions of the TSP.

***k*-near CE or CA** In the *k-near* problem, we are given an initial ordering $\alpha : S \to [|S|]$ and a nonnegative integer k. A *feasible* solution exhibits a set of edge edits (additions) attaining the nesting property so that the associated ordering π, induced by the neighborhood nestings, of the students satisfies $\pi(s) \in [\alpha(s) - k, \alpha(s) + k]$.

Next, we define three types of *k*-near problems. In the subsequent problem formulations, we bring back the interval property to our constraints since we consider problems where the question side is not allowed to be arbitrarily ordered.

Unconstrained *k*-near CE or CA In *Unconstrained k-near* Chain Editing (Addition), the student ordering must be *k*-near but the question side may be ordered any way. The objective is to minimize the number of edge edits (additions) so that there is a *k*-near ordering of the students that satisfies the nesting property.

Constrained *k*-near CE or CA In *Constrained k-near* Chain Editing (Addition), the student ordering must be *k*-near while the questions have a fixed initial ordering that must be kept. The objective is to minimize the number of edge edits (additions) so that there is *k*-near ordering of the students that satisfies the nesting property and respects the interval property according to the given question ordering.

Both *k*-near CE or CA In *Both k-near* Chain Editing (Addition), both sides must be *k*-near with respect to two given initial orderings on their respective sides. The objective is to minimize the number of edge edits (additions) so that there is a *k*-near ordering of the students that satisfies the nesting property and a *k*-near ordering of the questions that satisfies the interval property.

2.1.4 Main Results

In this chapter, we introduce *k*-near models to the Chain Editing problem and present surprising complexity results. Our *k*-near model captures realistic scenarios of MOOCs, where information from past tests is usually known and can be used to arrive at a reliable initial nearby ordering.

We find that five of the *k*-near Editing and Addition problems have polynomial time algorithms while the Unconstrained *k*-near Editing problem is NP-hard. Additionally, we provide an $O(kn)$ additive approximation algorithm for the NP-hard

<table>
<tr><th rowspan="2">Questions \ Students</th><th rowspan="2">Unconstrained</th><th colspan="2">k-near</th><th rowspan="2">Constrained</th></tr>
<tr><th>Editing</th><th>Addition</th></tr>
<tr><td>Unconstrained</td><td>NP-hard [83, 39]</td><td>NP-hard Thm 2.3.4, $\mathcal{O}(kn)$-approx Thm 2.3.5</td><td>$\mathcal{O}(n^3 2^{4k} k^{4k})$ Thm 2.3.3</td><td>$\mathcal{O}(n^2)$ Prop 2.1.3</td></tr>
<tr><td>k-near Editing</td><td>NP-hard Thm 2.3.4, $\mathcal{O}(kn)$-approx Thm 2.3.5</td><td>$\mathcal{O}(n^3 2^{8k} k^{8k+4})$ Thm 2.3.6</td><td></td><td rowspan="2">$\mathcal{O}(n^3 2^{4k} k^{4k+2})$ Thm 2.3.2</td></tr>
<tr><td>Addition</td><td>$\mathcal{O}(n^3 2^{4k} k^{4k})$ Thm 2.3.3</td><td></td><td>$\mathcal{O}(n^3 2^{8k} k^{8k+4})$ Thm 2.3.7</td></tr>
<tr><td>Constrained</td><td>$\mathcal{O}(n^2)$ Prop 2.1.3</td><td colspan="2">$\mathcal{O}(n^3 2^{4k} k^{4k+2})$ Thm 2.3.2</td><td>$\mathcal{O}(n^2)$</td></tr>
</table>

FIGURE 2.2: All variants of the decision version of the problems are shown with their respective complexities. The complexity of Unconstrained/Unconstrained Addition [83] and Editing [39] were derived before. More detailed results for these cases will be shown in Figure 2.3. All other results are given in this chapter. Most of the problems have the same complexity for both Addition and Editing versions. The only exception is the Unconstrained k-near version where Editing is NP-hard while Addition has a polynomial time algorithm.

case. Our intuition is that the Constrained k-near and Both k-near problems are considerably restrictive on the ordering of the questions, which make it easy to derive the best k-near student ordering. The Unconstrained k-near Addition problem is easier than the corresponding Editing problem because the correct neighborhood of the students can be inferred from the neighborhoods of all weaker students in the Addition problem, but not for the Editing version.

Aside from restricting the students to be k-near, we may consider all possible combinations of whether the students and questions are each k-near, fixed, or unconstrained. The remaining (non-symmetric) combinations not covered by the above k-near problems are both fixed, one side fixed and the other side unconstrained, and both unconstrained. The both fixed problem is easy as both orderings are given in the input and one only needs to check whether the orderings are consistent with the nesting of the neighborhoods. When one side is fixed and the other is unconstrained, we have already shown that the ordering of the unconstrained side is easily derivable from the ordering of the fixed side via Proposition 2.1.3. If both sides are unconstrained, this is exactly the Chain Editing (or Addition) problem, which are both known to be NP-hard (see below). Figure 2.2 summarizes the complexity of each problem, including our results for the k-near variants, which are starred. Note that the role of the students and questions are symmetric up to flipping the orderings.

To avoid any potential confusion, we emphasize that our algorithms are not fixed-parameter tractable algorithms, as our parameter k is not a property of problem instances, but rather is part of the constraints that are specified for the outputs to satisfy.

The remaining sections are organized as follows. Section 2.2 discusses existing work on variants of Chain Editing that have been studied before. Section 2.3 shows the exact algorithms for five of the k-near problems, and includes the NP-hardness proof and an $O(kn)$ additive approximation for the last k-near problem. Section 2.4 summarizes our main contributions.

2.2 Related Work

The earliest known results on hardness and algorithms tackled Chain Addition. Since many results parameterize in terms of the value of an optimal solution to their problem, we use OPT to denote the optimal value, where the problem solved depends on the context. Before stating the results, we define a couple of problems closely related to Chain Addition. The *Minimum Linear Arrangement* problem considers as input a graph $G = (V, E)$ and asks for an ordering $\pi : V \to [|V|]$ minimizing $\sum_{vw \in E} |\pi(v) - \pi(w)|$. The *Chordal Completion* problem, also known as the *Minimum Fill-In* problem, considers as input a graph $G = (V, E)$ and asks for the minimum size set of edges F to add to G so that $(V, E \cup F)$ has no chordless cycles. A *chordless* cycle is a cycle $(v_1, \ldots, v_r, v_1)$ such that for every i, j with $|i - j| > 1$ and $\{i, j\} \neq \{1, r\}$, we have $v_i v_j \notin E$. Yannakakis [83] proved that Chain Addition is NP-hard by a reduction from Linear Arrangement. He also showed that Chain Addition is a special case of Chordal Completion on graphs of the form $(G = U \cup V, E)$ where U and V are cliques. Recently, Chain Editing was shown to be NP-hard by Drange et al. [39].

Another problem called *Total Chain Addition* is essentially identical to Chain Addition, except that the objective function counts the number of total edges in the output graph rather than the number of edges added. For Total Chain Addition, Feder et al. [44] gave a 2-approximation. The total edge addition version of Chordal Completion has an $O(\sqrt{\Delta} \log^4(n))$-approximation algorithm [1] where Δ is the maximum degree of the input graph. For Chain Addition, Feder et al. [44] claimed an $8d + 2$-approximation, where d is the smallest number such that every vertex-induced subgraph of the original graph has some vertex of degree at most d. Natanzon et al. [74] gave an $8OPT$-approximation for Chain Addition by approximating Chordal Completion. However, no approximation algorithms are known for Chain Editing.

Modification to chordless graphs and to chain graphs have also been studied from a fixed-parameter point of view. A *fixed-parameter tractable (FPT)* algorithm for a problem of input size n and parameter p bounding the value of the optimal solution, is an algorithm that outputs an optimal solution in time $O(f(p)n^c)$ for some constant c and some function f dependent on p. For Chordal Completion, Kaplan et al. [63] gave an FPT in time $O(2^{O(OPT)} + OPT^2 nm)$. Fomin and Villanger [48] showed the first subexponential FPT for Chordal Completion, in time $O(2^{O(\sqrt{OPT} \log OPT)} + OPT^2 nm)$. Cao and Marx [27] studied a generalization of Chordal Completion, where three operations are allowed: vertex deletion, edge addition, and edge deletion. There, they gave an FPT in time $2^{O(OPT \log OPT)} n^{O(1)}$, where OPT is now the minimum total number of the three operations needed to obtain a chordless graph. For the special case of Chain Editing, Drange et al. [39] showed an FPT in time $2^{O(\sqrt{OPT} \log OPT)} + \text{poly}(n)$, where $\text{poly}(n)$ represents a polynomial function with respect to n. They also showed the same result holds for a related problem called Threshold Editing.

On the other side, Drange et al. [39] showed that Chain Editing and Threshold Editing do not admit $2^{o(\sqrt{OPT})}\text{poly}(n)$ time algorithms assuming the Exponential Time Hypothesis (ETH). For Chain Completion and Chordal Completion, Bliznets et al. [19] excluded the possibility of $2^{O(\sqrt{n}/\log n)}$ and $2^{O(OPT^{\frac{1}{4}}/\log^c OPT)} n^{O(1)}$ time algorithms assuming ETH, where c is a constant. For Chordal Completion, Cao and Sandeep [28] showed that no algorithms in time $2^{O(\sqrt{OPT}-\delta)} n^{O(1)}$ exist for any positive δ, assuming ETH. They also excluded the possibility of a PTAS for Chordal Completion assuming $P \neq NP$. Wu et al. [82] showed that no constant approximation

	Chordal	Chain
Editing	Unknown approximation, FPT [38]	Unknown approximation, FPT [38]
Addition	$8OPT$-approx [74], FPT [38]	$8OPT$-approx [74], $8d+2$-approx [44], FPT [38]
Total Addition	$O(\sqrt{\Delta}\log^4(n))$-approx [1], FPT [38]	2-approx [44], FPT [38]

FIGURE 2.3: This table shows existing results for the case that both sides are unconstrained, which are all known to be NP-hard from the upper left block of Figure 2.2.

is possible for Chordal Completion assuming the Small Set Expansion Conjecture. Figure 2.3 summarizes the known results for the aforementioned graph modification problems.

For the k-near problems, we show that the Unconstrained k-near Editing problem is NP-hard by adapting the NP-hardness proof for Threshold Editing from Drange et al. [38]. The remaining k-near problems have not been studied. An abbreviated version of this chapter appeared in the proceedings of the 11th International Conference and Workshops on Algorithms and Computation [62].

2.3 Polynomial Time Algorithms for k-near Orderings

We present our polynomial time algorithm for the Constrained k-near Addition and Editing problems, the Both k-near Addition and Editing problems, and the Unconstrained k-near Addition problem. We also show the NP-hardness of the Unconstrained k-near Editing problem and provide a $O(kn)$ additive approximation algorithm for it.

We assume correct orderings label the students from weakest (smallest label) to strongest (largest label) and label the questions from easiest (smallest label) to hardest (largest label). We associate each student with its initial label given by the k-near ordering. For ease of reading, we boldface the definitions essential to the analysis of our algorithm.

2.3.1 Constrained k-near

We will solve the Constrained k-near Editing and Addition problems in time $O(n^3 2^{4k} k^{4k+2})$ by dynamic programs. First, we will solve the Constrained k-near Editing problem. Then we modify the algorithm to solve the Constrained k-near Addition problem.

2.3.1.0.1 Constrained k-near Editing

Theorem 2.3.1 (Constrained k-near Editing)**.** *Constrained k-near Editing can be solved in time $O(n^3 2^{4k} k^{4k+2})$.*

Proof. Assume that the students are given in k-near order $1, \ldots, |S|$ and that the questions are given in exact order $1 \leq \cdots \leq |Q|$. We construct a dynamic program for Constrained k-near Editing. First, we introduce the subproblems that we will consider. Define $\boldsymbol{C(i, u_i, U_i, v_{j_i})}$ to be the smallest number of edges incident to the weakest i positions that must be edited such that u_i is in position i, U_i is the set of students in the weakest $i-1$ positions, and v_{j_i} is the hardest question correctly answered by

the i weakest students. Before deriving the recurrence, we will define several sets that bound our search space within polynomial size of $n = |S| + |Q|$.

Search Space for U_i. Given position i and student u_i, define $\boldsymbol{P_{i,u_i}}$ to be the set of permutations on the elements in $\big[\max\{1, i-k\}, \min\{|S|, i+k-1\}\big] \setminus \{u_i\}$. Let $\boldsymbol{F_{i,u_i}} := \Big\{\{\pi^{-1}(1), \ldots, \pi^{-1}(k)\} : \pi \in P_{i,u_i}, \pi(a) \in [a-k, a+k], \forall a \in \big[\max\{1, i-k\}, \min\{|S|, i+k-1\}\big] \setminus \{u_i\}\Big\}$. The set P_{i,u_i} includes all possible permutations of the $2k$ students centered at position i, and the set F_{i,u_i} enforces that no student moves more than k positions from its label. We claim that every element of F_{i,u_i} is a candidate for $U_i \setminus \big[1, \max\{1, i-k-1\}\big]$ given that u_i is assigned to position i. To understand the search space for U_i given i and u_i, observe that for all $i \geq 2$, U_i already must include all of $\big[1, \max\{1, i-k-1\}\big]$ since any student initially at position $\leq i-k-1$ cannot move beyond position $i-1$ in a feasible solution. If $i = 1$, we have $U_1 = \emptyset$. From now on, we assume $i \geq 2$ and treat the base case $i = 1$ at the end. So the set $U_i \setminus \big[1, \max\{1, i-k-1\}\big]$ will uniquely determine U_i. We know that U_i cannot include any students with initial label $[k+i, |S|]$ since students of labels $\geq k+i$ must be assigned to positions i or later. So the only uncertainty remaining is which elements in $\big[\max\{1, i-k\}, \min\{|S|, i+k-1\}\big] \setminus \{u_i\}$ make up the set $U_i \setminus \big[1, \max\{1, i-k-1\}\big]$. We may determine all possible candidates for $U_i \setminus \big[1, \max\{1, i-k-1\}\big]$ by trying all permutations of $\big[\max\{1, i-k\}, \min\{|S|, i+k-1\}\big] \setminus \{u_i\}$ that move each student no more than k positions from its input label, which is exactly the set F_{i,u_i}.

Feasible and Compatible Subproblems. Next, we define $\boldsymbol{S_i} = \Big\{(u_i, U_i, v_{j_i}) : u_i \in \big[\max\{1, i-k\}, \min\{|S|, i+k\}\big], U_i \setminus \big[1, \max\{1, i-k-1\}\big] \in F_{i,u_i}, v_{j_i} \in Q \cup \{0\}\Big\}$. The set S_i represents the search space for all possible vectors (u_i, U_i, v_{j_i}) given that u_i is assigned to position i. Note that u_i is required to be within k positions of i by the k-near constraint. So we encoded this constraint into S_i. To account for the possibility that the i weakest students answer no questions correctly, we allow v_{j_i} to be in position 0, which we take to mean that $U_i \cup \{u_i\}$ gave wrong answers to all questions.

Now, we define $\boldsymbol{R_{i-1,u_i,U_i,v_{j_i}}} := \{(u_{i-1}, U_{i-1}, v_{j_{i-1}}) \in S_{i-1} : v_{j_{i-1}} \leq v_{j_i}, U_i = \{u_{i-1}\} \cup U_{i-1}\}$. The set $R_{i-1,u_i,U_i,v_{j_i}}$ represents the search space for smaller subproblems that are compatible with the subproblem (i, u_i, U_i, v_{j_i}). More precisely, given that u_i is assigned to position i, U_i is the set of students assigned to the weakest $i-1$ positions, and v_{j_i} is the hardest question correctly answered by $U_i \cup u_i$, the set of subproblems of the form $(i-1, u_{i-1}, U_{i-1}, v_{j_{i-1}})$ which do not contradict the aforementioned assumptions encoded by (i, u_i, U_i, v_{j_i}) are exactly those whose $(u_{i-1}, U_{i-1}, v_{j_{i-1}})$ belongs to $R_{i-1,u_i,U_i,v_{j_i}}$. We illustrate compatibility in Figure 2.4.

The Dynamic Program. Finally, we define $\boldsymbol{c_{u_i,v_{j_i}}}$ to be the number of edge edits incident to u_i so that the neighborhood of u_i becomes exactly $\{1, \ldots, v_{j_i}\}$, i.e., $c_{u_i,v_{j_i}} := |N_G(u_i) \triangle \{1, \ldots, v_{j_i}\}|$. We know that $c_{u_i,v_{j_i}}$ is part of the cost within $C(i, u_i, U_i, v_{j_i})$ since v_{j_i} is the hardest question that $U_i \cup \{u_i\}$ is assumed to answer correctly and u_i is a stronger student than those in U_i who are in the positions before i. We obtain the following recurrence.

$$C(i, u_i, U_i, v_{j_i}) = \min_{(u_{i-1}, U_{i-1}, v_{j_{i-1}}) \in R_{i-1,u_i,U_i,v_{j_i}}} \{C(i-1, u_{i-1}, U_{i-1}, v_{j_{i-1}})\} + c_{u_i,v_{j_i}}$$

The base cases are $C(1, u_1, U_1, v_{j_1}) = |N_G(u_1) \triangle \{1, \ldots, v_{j_1}\}|$ if $v_{j_1} > 0$, and $C(1, u_1, U_1, v_{j_1}) = |N_G(u_1)|$ if $v_{j_1} = 0$ for all $u_1 \in [1, 1+k], v_{j_1} \in Q \cup \{0\}$.

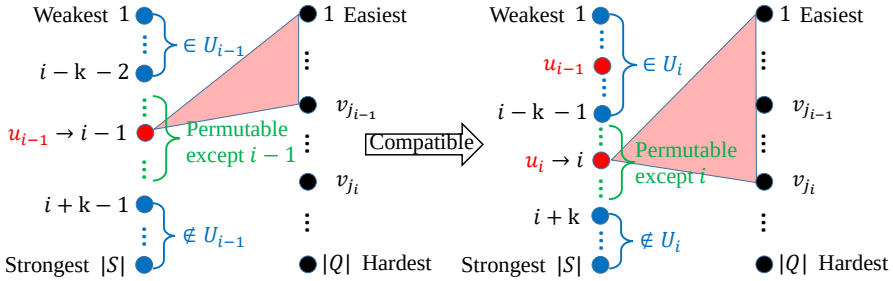


FIGURE 2.4: Subproblem $(i-1, u_{i-1}, U_{i-1}, v_{j_{i-1}})$ is compatible with subproblem (i, u_i, U_i, v_{j_i}) if and only if $v_{j_{i-1}}$ is no harder than v_{j_i} and $U_i = \{u_{i-1}\} \cup U_{i-1}$. The cost of (i, u_i, U_i, v_{j_i}) is the sum of the minimum cost among feasible compatible subproblems of the form $(i-1, u_{i-1}, U_{i-1}, v_{j_{i-1}})$ and the number of edits incident to u_i to make its neighborhood exactly $\{1, \ldots, v_{j_i}\}$.

By definition of our subproblems, the final solution we seek is $\min_{(u_{|S|}, U_{|S|}, v_{j_{|S|}}) \in S_{|S|}} C(|S|, u_{|S|}, U_{|S|}, v_{j_{|S|}})$.

Running Time. Now, we bound the run time of the dynamic program. Note that before running the dynamic program, we build the sets P_{i,u_i}, F_{i,u_i}, S_i, $R_{i-1,u_i,U_i,v_{j_i}}$ to ensure that our solution obeys the k-near constraint and that the smaller subproblem per recurrence is compatible with the bigger subproblem it came from. Generating the set P_{i,u_i} takes $(2k)! = O(2^{2k}k^{2k})$ time per (i, u_i). Checking the k-near condition to obtain the set F_{i,u_i} while building P_{i,u_i} takes k^2 time per (i, u_i). So generating S_i takes $O(k \cdot 2^{2k}k^{2k}k^2 \cdot |Q|)$ time per i. Knowing S_{i-1}, generating $R_{i-1,u_i,U_i,v_{j_i}}$ takes $O(|S|)$ time. Hence, generating all of the sets is dominated by the time to build $\cup_{i \le |S|} S_i$, which is $O(|S|k^3 2^{2k}k^{2k}|Q|) = O(n^2 2^{2k}k^{2k+3})$.

After generating the necessary sets, we solve the dynamic program. Each subproblem (i, u_i, U_i, v_{j_i}) takes $O(|R_{i-1,u_i,U_i,v_{j_i}}|)$ time. So the total time to solve the dynamic program is $O(\sum_{i \in S, (u_i,U_i,v_{j_i}) \in S_i} |R_{i-1,u_i,U_i,v_{j_i}}|) = O(|S||S_i||S_{i-1}|) = O(n(k \cdot 2^{2k}k^{2k} \cdot n)^2) = O(n^3 2^{4k}k^{4k+2})$. □

2.3.1.0.2 Constrained k-near Addition

We use the same framework as Constrained k-near Editing to solve the Constrained k-near Addition. We change the definitions of the subproblem, the relevant sets, and the costs appropriately to adapt to the Addition problem.

Theorem 2.3.2 (Constrained k-near Addition)**.** *Constrained k-near Addition can be solved in time $O(n^3 2^{4k}k^{4k+2})$.*

Proof. First, redefine $\boldsymbol{C(i, u_i, U_i, v_{j_i})}$ to be the smallest cost of adding edges incident to the weakest i positions so that u_i is in position i, U_i is the set of students in the weakest $i-1$ positions, and v_{j_i} is the hardest question correctly answered by the i weakest students.

The sets $\boldsymbol{P_{i,u_i}}$ and $\boldsymbol{F_{i,u_i}}$ will stay the same as before. We redefine $\boldsymbol{S_i} := \Big\{(u_i, U_i, v_{j_i}) : u_i \in [\max\{1, i-k\}, \min\{|S|, i+k\}], U_i \setminus [1, \max\{1, i-k-1\}] \in F_{i,u_i}, v_{j_i} \in Q \cup \{0\}, v_{j_i} \ge \max N_G(\{u_i\} \cup U_i)\Big\}$. Requiring that v_{j_i} is at least as hard as $N_G(\{u_i\} \cup U_i)$ ensures that the final solution will satisfy the interval property with respect to the given question order. It was not needed in the Editing problem

because wherever v_{j_i} landed, the edges that reach questions harder than v_{j_i} were deleted. The definition of $\boldsymbol{R_{i-1,u_i,U_i,v_{j_i}}}$ will stay the same as before, but using the new definition of S_{i-1} from this section. Finally, the cost $\boldsymbol{c_{u_i,v_{j_i}}}$ will become the number of edge additions incident to u_i so that the neighborhood of u_i becomes $\{1,\ldots,v_{j_i}\}$, i.e., $c_{u_i,v_{j_i}} := |\{1,\ldots,v_{j_i}\} \setminus N_G(u_i)|$.

The recurrence relation from Constrained k-near Editing still applies here. However, the base cases become $C(1,u_1,U_1,v_{j_1}) = |\{1,\ldots,v_{j_1}\} \setminus N_G(u_1)|$ if $v_{j_1} > 0$, and $C(1,u_1,U_1,v_{j_1}) = 0$ if $v_{j_1} = 0$.

The run time is still dominated by the dynamic program since the time to construct S_i becomes only $|Q|$ times larger (to enforce the additional constraint that v_{j_i} is hard enough). Hence the total time to solve this problem remains $O(n^3 2^{4k} k^{4k+2})$. □

2.3.2 Unconstrained k-near

First, we solve the Unconstrained k-near Addition problem in time $O(n^3 2^{4k} k^{4k})$. Second, we show that the Unconstrained k-near Editing problem is NP-hard.

Assume that the students are given in k-near order $1,\ldots,|S|$. The questions are allowed to be ordered arbitrarily in the final solution.

2.3.2.0.1 Unconstrained k-near Addition

Theorem 2.3.3 (Unconstrained k-near Addition)**.** *Unconstrained k-near Addition can be solved in time $O(n^3 2^{4k} k^{4k})$.*

Proof. We introduce subproblems of the form (i,u_i,U_i). Define $\boldsymbol{C(i,u_i,U_i)}$ to be the smallest number of edges incident to the weakest i positions that must be added so that u_i is in position i and U_i is the set of the $i-1$ weakest students.

We use the same $\boldsymbol{P_{i,u_i}}$ and $\boldsymbol{F_{i,u_i}}$ as defined for Constrained k-near Editing to bound the search space for U_i given that u_i is in position i. Define $\boldsymbol{S_i} := \Big\{(u_i,U_i) : u_i \in \big[\max\{1,i-k\},\min\{|S|,i+k\}\big], U_i \setminus [1,\max\{1,i-k-1\} \in F_{i,u_i}\Big\}$.

Next, define $\boldsymbol{R_{i-1,u_i,U_i}} := \Big\{(u_{i-1},U_{i-1}) \in S_{i-1} : U_i = \{u_{i-1}\} \cup U_{i-1}\Big\}$. The set R_{i-1,u_i,U_i} ensures that the smaller subproblems have prefixes that are compatible with those assigned in the bigger subproblems they came from. Compatibility is illustrated in Figure 2.5.

Lastly, define $\boldsymbol{c_{u_i,U_i}}$ to be the number of edge additions incident to u_i so that the neighborhood of u_i becomes the smallest set of questions containing $N_G(U_i \cup \{u_i\})$, i.e., $c_{u_i,U_i} := |N_G(U_i \cup \{u_i\}) \setminus N_G(u_i)|$.

Using the above definitions, we have the following recurrence:

$$C(i,u_i,U_i) = \min_{(u_{i-1},U_{i-1}) \in R_{i-1,u_i,U_i}} \{C(i-1,u_{i-1},U_{i-1})\} + c_{u_i,U_i}$$

The base cases are $C(1,u_1,U_1) = |N_G(U_1) \setminus N_G(u_1)|$ for all $(u_1,U_1) \in S_1$, since u_1 must add edges to the questions that the weaker students correctly answered.

The final solution to Unconstrained k-near Addition is $\min_{(u_{|S|},U_{|S|}) \in S_{|S|}} C(|S|,u_{|S|},U_{|S|})$.

To bound the run time, note that generating S_i takes $O(n \cdot 2^{2k} k^{2k} k^2)$ time. The dynamic program will dominate the run time again. In the dynamic program, each subproblem (i,u_i,U_i) takes $O(|R_{i-1,u_i,U_i}|)$ time. So the total time is $O(\sum_{i \in S,(u_i,U_i) \in S_i} |R_{i-1,u_i,U_i}|) = O(|S||S_i||S_{i-1}|) = O(n(n2^{2k}k^{2k})^2) = O(n^3 2^{4k} k^{4k})$. □

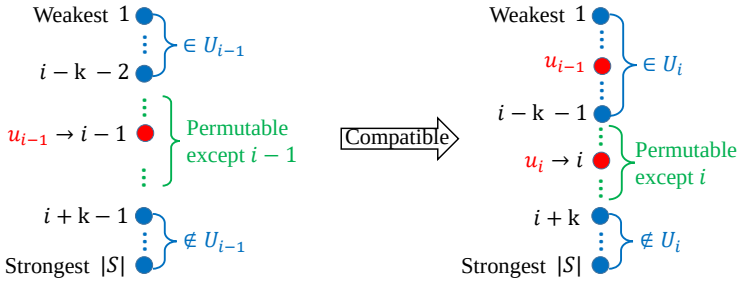


FIGURE 2.5: Subproblem $(i-1, u_{i-1}, U_{i-1})$ is compatible with subproblem (i, u_i, U_i) if and only if $U_i = \{u_{i-1}\} \cup U_{i-1}$. The cost of (i, u_i, U_i) is sum of the minimum cost among feasible compatible subproblems of the form $(i-1, u_{i-1}, U_{i-1})$ and the number of additions incident to u_i to make its neighborhood the smallest set of questions containing the existing neighbors of U_i.

2.3.2.0.2 Unconstrained k-near Editing

The Unconstrained k-near Editing problem is NP-hard even for $k = 1$. We closely follow the proof of Drange et al. [38] for the NP-hardness of Threshold Editing to show that Unconstrained k-near Editing is NP-hard. In Drange et al.'s construction, they specified a partial order for which the cost of Threshold Editing can only worsen if the output ordering deviates from it. We crucially use this property to prove NP-hardness for Unconstrained 1-near Editing.

Theorem 2.3.4 (Unconstrained k-near Editing)**.** *Unconstrained k-near Editing is NP-hard.*

Proof. Let $G = (S, Q, E)$ be a bipartite graph with initial student ordering π. Consider the decision problem Π of determining whether there is a 1-near unconstrained editing of at most t edges for the instance (G, π). We reduce from 3-SAT to Π. Let Φ be an instance for 3-SAT with clauses $C = \{c_1, \ldots, c_m\}$ and variables $V = \{v_1, \ldots, v_n\}$. We construct the corresponding instance $\Pi = (G_\Phi, \pi_\Phi, t_\Phi)$ for 1-near unconstrained editing as follows. First we order the variables in an arbitrary order and use this order to define π. For each variable v_i, create six students $s^i_a, s^i_b, s^i_f, s^i_t, s^i_c, s^i_d$. Next, we define a partial ordering P that the initial order π_Φ shall obey. Define P to be the partial order satisfying $s^i_a > s^i_b > s^i_f, s^i_t > s^i_c > s^i_d$ for all $i \in [n]$ and $s^i_\alpha > s^j_\beta$ for all $i < j$, $\alpha, \beta \in \{a, b, c, d, f, t\}$. Define π_Φ to be the linear ordering satisfying all relations of P for the variables in the initial arbitrary order, and additionally $s^i_f > s^i_t$. We remark that the proof works regardless of whether we set $s^i_f > s^i_t$ or $s^i_f < s^i_t$ in π_Φ. We shall impose that optimal solutions satisfy all of the relations of P. To do so, for every $s > s'$, we add $t_\Phi + 1$ new questions each with edges to s and no edges to s', and with edges to all $r > s$ in π_Φ. Then whenever an editing solution switches the order of s and s', it must edit at least $t_\Phi + 1$ edges. After adding the necessary questions to ensure feasible solutions must preserve the partial order P, we create a question q_{c_l} for each clause c_l. If a variable v_i appears positively in c_l, then add the edge $q_{c_l} s^i_t$. If v_i appears negatively in c_l, then add the edge $q_{c_l} s^i_f$. If v_i does not occur in c_l, then add the edge $q_{c_l} s^i_c$. For all variables v_i and clauses c_l, add the edges $q_{c_l} s^i_b$ and $q_{c_l} s^i_d$. Finally, define $t_\Phi = |C|(3|V| - 1)$. Refer to Figure 2.6 for an illustration of the construction.

Now, we show that there is a satisfying assignment if and only if there is a 1-near editing of at most t_Φ edges. First, we prove the forward direction. Assume there is a

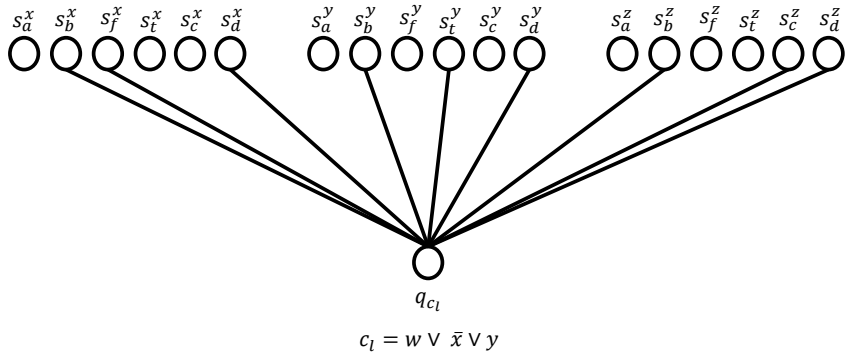


FIGURE 2.6: Each set of six vertices represents the students corresponding to a variable x, y, or z. The bottom vertex represents a question corresponding to the clause $c_l = w \vee \bar{x} \vee y$.

satisfying assignment $f : V \to \{T, F\}$. Let c_l be a clause. One of the literals v_i in c_l is set to T under the assignment f. If v_i occurs positively, then edit the neighborhood of q_{c_l} to be all students s such that $s \geq s_t^i$ according to P and impose $s_t^i > s_f^i$ in the solution. If v_i occurs negatively in q_{c_l}, then edit the neighborhood of q_{c_l} to be all students s such that $s \geq s_f^i$ and keep the initial order that $s_f^i > s_t^i$. In both cases, the neighborhood of q_{c_l} changed by 2 among the six students corresponding the variable v_i and changed by 3 for the remaining groups of six students. So the number of edge edits incident to each (clause) question is $3|V| - 1$. Note that the neighborhoods of the extra questions we added to impose P are already nested because each time a new question was added, it received edges to all students who are stronger than a particular student according to P. So only the questions that came from clauses potentially need to edit their neighborhoods to achieve nesting. Hence, the total number of edge edits is $|C|(3|V| - 1) = t_\Phi$.

Second, we prove the backward direction. Assume there is an unconstrained 1-near editing of $|C|(3|V| - 1)$ edges to obtain a chain graph. Let c_l be a clause. For any variable v_j not occurring in c_l, the original edges that q_{c_l} has to the six students corresponding to v_j are to s_b^j, s_c^j, s_d^j. If the cut-off point of the edited neighborhood of q_{c_l} is among $s_a^j, s_b^j, s_f^j, s_t^j, s_c^j, s_d^j$, then the edges incident to q_{c_l} must change by at least three among those six, which means that q_{c_l} would have at least $3|V|$ edges incident to it. If the cut-off point of the edited neighborhood of q_{c_l} is among the six students corresponding to a variable v_i that occurs in c_l, then the edges incident to q_{c_l} must change by at least two (by switching the order of s_f^i and s_t^i when needed) among those six students and at least three for the students corresponding to the remaining variables. Thus q_{c_l} has at least $3|V| - 1$ edges edits incident to it for every c_l. So the smallest number of edge edits possible is at least $|C|(3|V| - 1)$. By the assumption, G_Φ has a feasible editing of at most $|C|(3|V| - 1)$ edges. Then each q_{c_l} must have exactly $3|V| - 1$ edits incident to it. So the cut-off point for the edited neighborhood of each q_{c_l} must occur among the six students corresponding to a variable v_i occurring inside c_l. If the occurring variable v_i is positive, then the cut-off point must have been at s_t^i and required $s_t^i > s_f^i$ since all other cut-offs incur at least three edits. Similarly, if v_i is negative, then the cut-off point must have been at s_f^i and required $s_f^i > s_t^i$. All clauses must be consistent in their choice of the ordering between s_f^i and s_t^i for all $i \in [n]$ since the editing solution was feasible. Hence, we obtain a satisfying assignment by setting each variable v_i true if and only if $s_t^i > s_f^i$. □

Next, we show a simple $O(kn)$ additive approximation algorithm for Unconstrained k-near Editing.

Theorem 2.3.5 (Approximation for Unconstrained k-near Editing)**.** *Unconstrained k-near Editing has an $O(kn)$ additive approximation algorithm.*

Proof. Fix the student side to the initial ordering $\sigma : S \to [|S|]$ given for the k-near condition and solve the corresponding Constrained Unconstrained Editing problem exactly. Denote by F the edge edits found from solving the Constrained Unconstrained Editing problem. Let σ^* be the ordering for S in an optimal solution to the original k-near Unconstrained problem. Let H be the minimum size edge edits corresponding to σ^*. It suffices to show that for each $q \in Q$, $|N_F(q)| - |N_H(q)| \leq 2k-2$, since this inequality would imply that $|F| - |H| \leq (2k-2)|Q| \leq 2kn$.

For $q \in Q$, let $p(q)$ be the position of the weakest student who answers q correctly according to the ordering σ^*. By the k-near condition, any student more than $k-1$ positions after $p(q)$ cannot be ordered before $p(q)$ and vice versa. If $p(q)$ remains the position of the weakest student who correctly answers q according to the ordering σ, then the edge edits required would be the same as H, except for possibly those edges from q to students who are within $k-1$ positions of $p(q)$. For each q, F is determined by choosing the cut-off position for the neighborhood of q that minimizes the number of edits needed. Then $N_F(q)$ should differ from $N_H(q)$ no more than the case where the cut-off point for q stays the same position as $p(q)$. So $|N_F(q)| - |N_H(q)| \leq 2(k-1)$. Hence $|F| - |H| = O(kn)$. □

2.3.3 Both k-near

We will solve the Both k-near Editing and Addition problems in time $O(n^3 2^{8k} k^{8k+4})$. We first show our solution for the Editing problem and then adapt it to the Addition problem.

Assume that the students and questions are both given in k-near order with student labels $1, \ldots, |S|$, and question labels $1, \ldots, |Q|$.

2.3.3.0.1 Both k-near Editing

Theorem 2.3.6 (Both k-near Editing)**.** *Both k-near Editing can be solved in time $O(n^3 2^{8k} k^{8k+4})$.*

Proof. We consider subproblems of the form $(i, u_i, U_i, j_i, v_{j_i}, V_{j_i})$. Define $\boldsymbol{C(i, u_i, U_i, j_i, v_{j_i}, V_{j_i})}$ to be the smallest number of edges incident to the weakest i students that must be edited so that student u_i is in position i, U_i is the set of the $i-1$ weakest students, j_i is the position of the hardest question correctly answered by $U_i \cup \{u_i\}$, v_{j_i} is the question in position j_i, and V_{j_i} is the set of the $j_i - 1$ easiest questions.

Feasible and Compatible Subproblems. Next, we define the search space for $(u_i, U_i, j_i, v_{j_i}, V_{j_i})$ given that u_i is in position i. We use the same $\boldsymbol{P_{i,u_i}}$ and $\boldsymbol{F_{i,u_i}}$ defined in the proof for Constrained k-near Editing. Define $\boldsymbol{S_i} := \Big\{(u_i, U_i, j_i, v_{j_i}, V_{j_i}) : u_i \in \big[\max\{1, i-k\}, \min\{|S|, i+k\}\big], U_i \setminus \big[1, \max\{1, i-k-1\}\big] \in F_{i,u_i}, v_{j_i} \in \big[\max\{1, j_i - k\}, \min\{|Q|, j_i + k\}\big], V_{j_i} \setminus \big[1, \max\{1, j_i - k - 1\}\big] \in F_{j_i, v_{j_i}}\Big\}$. Here, we need to constrain both the student side and the question side to make sure that all elements are k-near as opposed to only enforcing the k-nearness on the students in Constrained k-near Editing.

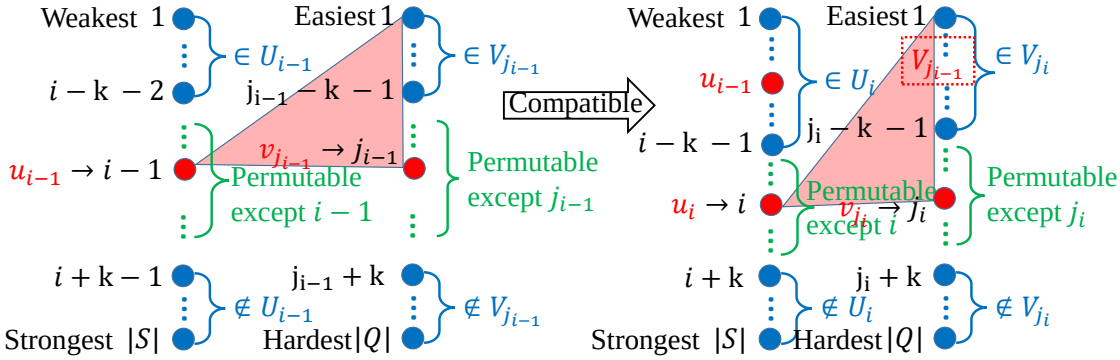


FIGURE 2.7: Subproblem $(i-1, u_{i-1}, U_{i-1}, j_{i-1}, v_{j_{i-1}}, V_{j_{i-1}})$ is compatible with subproblem $(i, u_i, U_i, j_i, v_{j_i}, V_{j_i})$ if and only if $U_i = \{u_i\} \cup U_{i-1}$, j_{i-1} represents a position no harder than j_i, $V_{j_i} \cup \{v_{j_i}\}$ contains $V_{j_{i-1}} \cup \{v_{j_{i-1}}\}$, and j_{i-1} strictly easier than j_i implies that V_{j_i} contains $V_{j_{i-1}} \cup \{v_{j_{i-1}}\}$. The cost of $(i, u_i, U_i, j_i, v_{j_i}, V_{j_i})$ is the sum of the minimum cost among feasible compatible states of the form $(i-1, u_{i-1}, U_{i-1}, j_{i-1}, v_{j_{i-1}}, V_{j_{i-1}})$ and the number of edits incident to u_i that makes its neighborhood $V_{j_i} \cup \{v_{j_i}\}$.

To bound the search space for subproblems to be compatible with the bigger subproblems they came from, we define $\boldsymbol{R_{i-1,u_i,U_i,j_i,v_{j_i},V_{j_i}}} := \Big\{(u_{i-1}, U_{i-1}, j_{i-1}, v_{j_{i-1}}, V_{j_{i-1}}) \in S_{i-1} : U_i = U_{i-1} \cup \{u_{i-1}\}, j_i \geq j_{i-1}, V_{j_i} \cup \{v_{j_i}\} \supseteq V_{j_{i-1}} \cup \{v_{j_{i-1}}\}, j_i > j_{i-1} \Rightarrow V_{j_i} \supseteq V_{j_{i-1}} \cup \{v_{j_{i-1}}\}\Big\}$. The constraints in the set $R_{i-1,u_i,U_i,j_i,v_{j_i},V_{j_i}}$ ensure that the prefixes of position i and position j_i in the smaller subproblem will be compatible with the bigger subproblem that it came from. Furthermore, $j_i \geq j_{i-1}$ ensures that stronger students correctly answer all questions that weaker students correctly answered. We demonstrate compatibility in Figure 2.7.

The Dynamic Program. Finally, define $\boldsymbol{c_{u_i,v_{j_i},V_{j_i}}}$ to be the number of edge edits incident to u_i so that the neighborhood of u_i becomes exactly $V_{j_i} \cup \{v_{j_i}\}$, i.e., $c_{u_i,v_{j_i},V_{j_i}} := |N_G(u_i) \triangle V_{j_i} \cup \{v_{j_i}\}|$.

Using the above definitions, we obtain the following recurrence.

$$C(i, u_i, U_i, j_i, v_{j_i}, V_{j_i}) = \min_{(u_{i-1}, U_{i-1}, j_{i-1}, v_{j_{i-1}}, V_{j_{i-1}}) \in R_{i-1,u_i,U_i,j_i,v_{j_i},V_{j_i}}} \{C(i-1, u_{i-1}, U_{i-1}, j_{i-1}, v_{j_{i-1}}, V_{j_{i-1}})\} + c_{u_i,v_{j_i},V_{j_i}}$$

The base cases are $C(1, u_1, U_1, j_1, v_{j_1}, V_{j_1}) = |N_G(u_1) \triangle \{v_{j_1}\} \cup V_{j_1}|$ for all $(u_1, U_1, j_1, v_{j_1}, V_{j_1}) \in S_1$.

The final solution is $\min_{(u_{|S|}, U_{|S|}, j_{|S|}, v_{j_{|S|}}, V_{j_{|S|}}) \in S_{|S|}} C(|S|, u_{|S|}, U_{|S|}, j_{|S|}, v_{j_{|S|}}, V_{j_{|S|}})$.

Running Time. First, observe that $|S_i| = O(k^2 2^{4k} k^{4k} |Q|)$, since there are $O(k)$ choices for u_i and v_i, $O(2^{2k}k^{2k})$ choices for U_i and V_{j_i}, and $|Q|$ choices for j_i. To build S_i, we need to build F_{i,u_i} and $F_{j_i,v_{j_i}}$. In Section 2.3, we saw that each of the F_{i,u_i} takes $O(k^2 2^{2k} k^{2k})$ time to build. Then building the set S_i is upper bounded by $O(k \cdot 2^{2k}k^{2k}k^2 \cdot |Q| \cdot k \cdot 2^{2k}k^{2k}k^2)$ per i, where we are over-counting the time to generate all possible U_i and V_{j_i} by the time it takes to build F_{i,u_i} and $F_{j_i,v_{j_i}}$. Building the set $R_{i-1,u_i,U_i,j_i,v_{j_i},V_{j_i}}$ while building S_i will take $O(|S| + |Q|)$ to check the conditions

that restrict S_{i-1} to $R_{i-1,u_i,U_i,j_i,v_{j_i},V_{j_i}}$. Due to the size of S_i, the construction of sets will still be dominated by the time to solve the dynamic program. Specifically, each subproblem $(i, u_i, U_i, j_i, v_{j_i}, V_{j_i})$ takes $O(|R_{i-1,u_i,U_i,j_i,v_{j_i},V_{j_i}}|)$ time. So the total time is $O(\sum_{i\in S,(u_i,U_i,j_i,v_{j_i},V_{j_i})\in S_i} |R_{i-1,u_i,U_i,j_i,v_{j_i},V_{j_i}}|) = O(|S||S_i||S_{i-1}|) = O(n(k^2 \cdot 2^{4k}k^{4k}n)^2) = O(n^3 2^{8k}k^{8k+4})$. □

2.3.3.0.2 Both k-near Addition

To solve the Addition version, we apply the method from the solution for Both k-near Editing.

Theorem 2.3.7 (Both k-near Addition)**.** *Both k-near Addition can be solved in time $O(n^3 2^{8k}k^{8k+4})$.*

Proof. We redefine $\boldsymbol{C(i, u_i, U_i, j_i, v_{j_i}, V_{j_i})}$ to be the smallest number of edges incident to the weakest i students that must be added so that student u_i is in position i, U_i is the set of the $i-1$ weakest students, j_i is the position of the hardest question correctly answered by $U_i \cup \{u_i\}$, v_{j_i} is the question in position j_i, and V_{j_i} is the set of the $j_i - 1$ easiest questions.

We keep $\boldsymbol{P_{i,u_i}}$ and $\boldsymbol{F_{i,u_i}}$ the same as in the proof for Constrained k-near Editing. Redefine $\boldsymbol{S_i} := \Big\{(u_i, U_i, j_i, v_{j_i}, V_{j_i}) : u_i \in [\max\{1, i-k\}, \min\{|S|, i+k\}], U_i \setminus [1, \max\{1, i-k-1\}] \in F_{i,u_i}, v_{j_i} \in [\max\{1, j_i - k\}, \min\{|Q|, j_i + k\}], V_{j_i} \setminus [1, \max\{1, j_i - k - 1\}] \in F_{j_i,v_{j_i}}, V_{j_i} \cup \{v_{j_i}\} \supseteq N_G(\{u_i\} \cup U_i)\Big\}$. The addition constraint $V_{j_i} \cup \{v_{j_i}\} \supseteq N_G(\{u_i\} \cup U_i)$ is added here to ensure that the interval property induced by the current student ordering is satisfied every step. It was not needed in section 2.3.3.0.1 because existing edges to questions outside $V_{j_i} \cup \{v_{j_i}\}$ could be deleted. The definition of $\boldsymbol{R_{i-1,u_i,U_i,j_i,v_{j_i},V_{j_i}}}$ remains the same as section 2.3.3.0.1, but using the newly defined S_{i-1}. Lastly, redefine $\boldsymbol{c_{u_i,v_{j_i},V_{j_i}}}$ to be the number of edge additions incident to u_i so that the neighborhood of u_i becomes exactly $V_{j_i} \cup \{v_{j_i}\}$, i.e., $c_{u_i,v_{j_i},V_{j_i}} := |V_{j_i} \cup \{v_{j_i}\} \setminus N_G(u_i)|$.

The general recurrence relation of Section 2.3.3.0.1 stays the same. The base cases change to $C(1, u_1, U_1, j_1, v_{j_1}, V_{j_1}) = |\{v_{j_1}\} \cup V_{j_1} \setminus N_G(u_1)|$, with the convention that $j_1 = 0$ means $V_{j_1} = \emptyset$ and v_{j_1} is omitted from the count $|\{v_{j_1}\} \cup V_{j_1}|$.

Although the time to construct S_i is larger by a factor of $|Q|$, the total run time is dominated by the dynamic program, which takes $O(n^3 2^{8k}k^{8k+4})$. □

It is possible that the above running times for the five "easy" problems could improve. Our dynamic programs are designed based on the intuitiveness of the states and not necessarily optimized for time complexity.

2.4 Conclusion

We proposed a new set of problems that arise naturally from ranking participants and tasks in competitive settings and classified the complexity of each problem. First, we introduced six k-near variants of the Chain Editing problem, which capture a common scenario of having partial information about the final orderings from past rankings. Second, we provided polynomial time algorithms for five of the problems and showed NP-hardness and an $O(kn)$ additive approximation for the remaining one.

Some open questions still remain for the NP-hard problems in Figure 2.2. For Chain Editing when both sides are unconstrained, there are no known approximation algorithms. For the corresponding Chain Addition problem, can a constant approximation can be achieved? For the Unconstrained k-near Editing problem, can the $O(kn)$ additive approximation be improved?

Acknowledgments
This work was supported in part by the US National Science Foundation under award numbers CCF-1527032, CCF-1655442, and IIS-1553547.

Chapter 3

Inventory Routing on Line Metrics

3.1 Introduction

The inventory routing problem (IRP) has been studied extensively in supply chain optimization [23, 29, 25, 26]. It models the delivering decisions that a supplier must make to satisfy demands for its product at various locations over a time horizon. For instance, a chain store may have different levels of demand for a product depending on the location of the store. As the product sells, each location needs to be replenished to satisfy future demands. The supplier is asked by the chain store to decide when and how much of its product to ship to each location. Storing extra amounts of the product over time incurs holding cost while each delivery to different locations from the supply center incurs routing cost. The supplier would like to minimize its total shipping cost and storage cost over the time horizon. Visiting frequently incurs high routing cost while visiting sparsely incurs high holding cost. Minimizing the total cost requires finding the right trade off between the two costs.

We now give the formal definition of IRP. In *IRP*, we are given a complete graph $K_n = (V, E)$ whose vertices are potential locations of clients and whose edge weights are determined by a *metric* $w : E \to \mathbb{R}_{\geq 0}$ ($w_{xz} \leq w_{xy} + w_{yz}$ for all $x, y, z \in V$). In a *graph metric*, the distance between every pair of vertices is the length of the shortest path between them in a given weighted graph. There is a depot $r \in V$ from which a vehicle loads supply to drop off to clients. The vehicle may carry any amount of supply from the depot at each time. We have a discrete time horizon $1, \ldots, T$ over which client $v \in V$ demands $d_t^v \geq 0$ units of supply to be delivered to it by time t. For each client $v \in V$, demand time $t \in [T]$, and potential serving time $s \leq t$, storing one unit of supply at v during time $[s, t]$ incurs a holding cost of $h_{s,t}^v$. We assume that the holding costs are *monotone*, i.e., $h_{s_1,t}^v \geq h_{s_2,t}^v$ for every client $v \in V$, $t \in [T]$, and $s_1 \leq s_2$. We denote by $D(V \times [T])$ the set points (v, t) such that $d_t^v > 0$. When the context of V and $[T]$ are clear, we use D and $D(V \times [T])$ interchangeably. We call such points *demand points*. The objective is to select a tour from r through a subset of clients per time $t \in [T]$ to satisfy every demand point (no late delivery allowed) so that the total routing cost and holding cost over $[T]$ is minimized. For simplicity, we assume that the route is a Steiner tree (instead of a tour) through the selected subset of clients each day since previous work in approximation algorithms for IRP assume the same. Observe that a Steiner tree can be converted into a tour of twice its cost by doubling the tree's edges and short-cutting. Denote by $H_{s,t}^v$ the holding cost incurred if d_t^v is served at time s, i.e., $H_{s,t}^v = h_{s,t}^v d_t^v$. We remark that there is always an optimal solution such that each d_t^v is served at a single time, for if d_t^v is delivered in separate portions at times $s_1 < \ldots < s_l$, then the total cost does not increase if we move all of d_t^v to be delivered at time s_l.

In this chapter, we construct constant factor *approximation algorithms* (polynomial

time algorithms producing solution within a guaranteed factor of the cost of an optimal solution) for IRP restricted to line metrics. First, we show a 5-approximation for IRP on line metrics with a simple rounding method. Second, we provide a primal dual algorithm for IRP on line metrics, proving a 26-approximation. The *line metric* induces the distance between each pair of vertices to be the total weight of the edges between them on a given weighted line. Although there is a dynamic program [18] for the line metric case of IRP, we study LP based methods as they are more generalizable to harder metrics. Our primal dual algorithm extends the method [67] for the closely related joint replenishment problem (JRP) to IRP on the line to obtain a 26-approximation.

The remaining sections are organized as follows. In Section 4.2, we summarize known approximation algorithms, heuristics, and policies for IRP. In Section 3.3, we provide a 5-approximation for IRP on line metrics via LP rounding. In Section 3.4, we show a 26-approximation by a primal dual method for IRP on line metrics. In Section 3.5, we state related open questions.

3.2 Related Work

Approximation algorithms for special cases of IRP have been studied while the general problem has no known constant factor approximation. IRP on general metrics has a $O(\frac{\log T}{\log \log T})$-approximation by Nagarajan and Shi [73] and an $O(\log N)$-approximation by Fukunaga et al [49]. For IRP restricted to nested periodic schedules, Fukunaga et al. [49] provide a 2.55-approximation. The joint replenishment problem is equivalent to IRP on a two-level tree metric, where the first level has only one edge. For JRP, Levi et al. [67] give a 2-approximation via a primal dual approach. They reduced the approximation factor to 1.8 in [68] by LP rounding. Recently, Bienkowski et al. [17] improve the factor to 1.791 by randomized rounding. For the online version of JRP, Buchbinder et al. [22] give a 3-approximation by a different primal dual method. A generalization of the JRP is IRP on arbitary trees, for which Cheung et el. [32] provide a 3-approximation.

Heuristics for IRP and its variants have been extensively studied. For instance, Federgruen and Zipkin [45] study both stochastic and deterministic IRP with capacitated vehicles using a single day approach to initialize a feasible solution and switch clients between routes to reduce the cost. In a different single day approach, Golden et al. [56] determine the client set per day by the urgency level of each client. Chien et al. [33] study a single day approach that passes information between consecutive days. They seek to maximize daily profit, where the profit is determined by the revenue gained from the quantity delivered and the revenue lost from not satisfying demands, but unsatisfied demand for one day is recalculated as revenue gain for the next day. In a short term planning approach for IRP with capacitated storage locations, Dror et al. [40, 41] find the next best replenishment day per client for each short term period based on the routing cost and stockout cost. For stochastic IRP, Jaillet et al. [12, 13, 61] find the best replenishment day per client within every two week period, assigning only the first week's solution, and re-solving the next two weeks.

Researchers have also developed policies for IRP by restricting the structure of the routes. Kleywegt et al. [66] consider the stochastic IRP with stockout penalties under the restriction that the vehicle must visit the depot after each client visit. For deterministic IRP, Anily and Federgruen [5] break the client set into regions and

study the restricted version where every region's entire client set must be visited whenever one of its client is visited.

3.3 Primal Rounding for Line

Here, we give a 5-approximation for IRP on the line by rounding an optimal primal solution. We use the edge-based LP specialized to the line. Given a line metric, let w_e be the weight of edge e in the line. For each $v \in V$, let E_{rv} be the set of edges between r and v in the line. The LP relaxation is:

$$\min \quad \sum_{e\in E}\sum_{s=1}^{T} w_e y_s^e + \sum_{(v,t)\in D(V\times[T])}\sum_{s=1}^{t} H_{s,t}^v x_{s,t}^v$$

$$\text{s.t.} \quad \sum_{s=1}^{t} x_{s,t}^v \geq 1 \qquad \forall (v,t) \in D(V\times[T]) \tag{3.1}$$

$$y_s^e \geq x_{s,t}^v \qquad \forall (v,t) \in D(V\times[T]), s \in [t], e \in E_{rv} \tag{3.2}$$

$$x_{s,t}^v \geq 0 \qquad \forall (v,t) \in D(V\times[T]), s \in [t] \tag{3.3}$$

$$y_s^e \geq 0 \qquad \forall e \in E, s \in [T]. \tag{3.4}$$

Observe that in an optimal solution, if e_1 is left of e_2, then $y_s^{e_1} \geq y_s^{e_2}$ for all s by the second constraint.

The idea of our rounding algorithm is to space out the visits at doubling distances from r so that we can pay for each visit with a disjoint portions of y_s^e. For each visit distance, we decide when to visit these locations based on the amount y_s^e accumulated over time.

Label the clients $1, \ldots, N$ as before. Denote the edge incident to r by e_1. For each i such that $w_{e_1}2^i \leq D_N$, if there is no client at distance $w_{e_1}2^i$ from r, insert a dummy client v_i here with $d_t^{v_i} = 0$ for all $t \in [T]$. We label any existing client at distance $w_{e_1}2^i$ from r by v_i as well. Let L be the last i such that $w_{e_1}2^i \leq D_N$. For consistency of notation, v_{L+1} shall denote the a dummy client (of zero demand) at distance $w_{e_1}2^{L+1}$ from r. We may assume $D_N > w_{e_1}2^L$; otherwise the same analysis will hold up to v_L. Let e_{i+1} be the edge incident to v_i to the left of v_i. Let R_i be the path from v_{i-1} to v_i. Solve the LP of the instance with the newly added edges and clients and call the solution (x, y). We round the LP solution as follows:

1: Pick $\alpha \in (0, 1)$.
2: **for** accumulation thresholds $p \leftarrow \alpha, 2\alpha, \ldots, \lfloor \frac{1}{\alpha} \rfloor \alpha$ **do**
3: **for** $i \leftarrow 1, \ldots, L+1$ **do**
4: visit up to client v_i the first time t_i' such that $\sum_{t=1}^{t_i'} y_t^{e_i} \geq p$.
5: **end for**
6: **end for**
7: **for** each $(i, t) \in D(V \times [T])$ **do**
8: deliver at latest visit passing through i.
9: **end for**

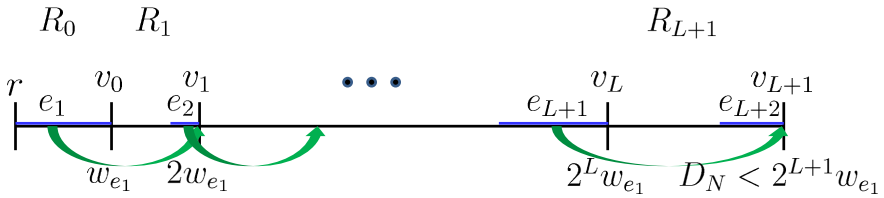


FIGURE 3.1: Each arrow from an edge to a location indicates that accumulation of the LP values of that edge determines the times assigned to visits up to that location.

We will choose α after finding the routing and holding costs. For each $i \in [1, L+1]$, let $t_i^1, \ldots, t_i^{l_i}$ be the set of earliest times such that $\sum_{t=1}^{t_i^j} y_t^{e_i} \geq j\alpha$. We set $t_i^0 = 0$. Observe that for each i, visits to v_{i+1} are less numerous over time than those to v_i. We illustrate the idea of our rounding algorithm below.

First, we bound the routing cost. Consider client v_i with $i \in [1, L+1]$. The routing cost of client v_i is at most $w_{e_1} 2^i$ (possibly lower for v_{L+1}). To visit v_i at time t_i^k, we charge the LP solution

$$\begin{aligned}
\sum_{e \in R_{i-1}} \sum_{t=t_i^{k-1}+1}^{t_i^k} w_e y_t^e &\geq \sum_{e \in R_{i-1}} w_e \sum_{t=t_i^{k-1}+1}^{t_i^k} y_t^{e_i} \\
&\geq \sum_{e \in R_{i-1}} w_e [k\alpha - (k-1)\alpha] \\
&= \left(\sum_{e \in R_{i-1}} w_e \right) \alpha \\
&= |R_{i-1}| \alpha \\
&= w_{e_1} (2^{i-1} - 2^{i-2}) \alpha \\
&= w_{e_1} 2^{i-2} \alpha.
\end{aligned}$$

So the routing cost ratio is $\frac{w_{e_1} 2^i}{w_{e_1} 2^{i-2} \alpha} = \frac{4}{\alpha}$.

Second, we bound the holding cost. Let $i \in [1, L+1]$. Let $v \in R_i$. Consider a demand time $\hat{t} \in [T]$. Let $t_i^{\hat{k}}$ be the latest t_i^j such that $t_i^j \leq \hat{t}$. Then $t_i^{\hat{k}+1} > \hat{t}$. Then the holding cost of demand point $(v, \hat{t})$ is $H^v_{t_i^{\hat{k}}, \hat{t}}$. To cover the holding cost of $(v, \hat{t})$, we charge the LP solution $\sum_{t=1}^{t_i^{\hat{k}}} H^v_{t,\hat{t}} x^v_{t,\hat{t}}$.

Lemma 3.3.1. $\sum_{t=1}^{t_i^{\hat{k}}} x^v_{t,\hat{t}} \geq 1 - \alpha$.

Proof. If $\sum_{t=1}^{t_i^{\hat{k}}} x^v_{t,\hat{t}} < 1 - \alpha$, then $\sum_{t=t_i^{\hat{k}}+1}^{\hat{t}} x^v_{t,\hat{t}} > \alpha$, i.e., $\sum_{t=t_i^{\hat{k}}+1}^{\hat{t}} y_t^{e_i} > \alpha$ since $e_i \in E_{rv}$, which contradicts the minimality of $t_i^{\hat{k}+1}$. □

So the LP pays

$$\begin{aligned}\sum_{t=1}^{t_i^{\hat{k}}} H^v_{t,\hat{t}} x^v_{t,\hat{t}} &\geq \sum_{t=1}^{t_i^{\hat{k}}} H^v_{t_i^{\hat{k}},\hat{t}} x^v_{t,\hat{t}} \\ &= H^v_{t_i^{\hat{k}},\hat{t}} \sum_{t=1}^{t_i^{\hat{k}}} x^v_{t,\hat{t}} \\ &\geq (1-\alpha) H^v_{t_i^{\hat{k}},\hat{t}}.\end{aligned}$$

So the holding cost ratio for all clients is $\frac{1}{1-\alpha}$.
If $\frac{4}{\alpha} = \frac{1}{1-\alpha}$, then $\alpha = \frac{4}{5}$, which gives an overall ratio of 5.

3.4 Primal Dual

Here, we give a 26-approximation using a primal dual approach similar to the approach from [67]. In subsection 3.4.1, we provide the primal and dual LPs that we use to construct the primal dual algorithm. The algorithm will involve two parts: a primal dual phase and a pruning phase. In subsection 3.4.2, we provide the primal dual phase of the algorithm and the analysis of the costs of this phase. In subsection 3.4.3, we give an example where the primal dual phase incurs a $\Omega(T/\log T)$ factor in the cost. In subsection 3.4.4, we give the pruning phase and the analysis of the constant factor that the pruning phase obtains.

3.4.1 LP Formulation

Here, we give our distance-based LP formulation for IRP on the line. This LP is a modification of Levi et al.'s LP for JRP. We label the vertices $0, \ldots, N$ on the line in increasing distance to r, identifying r with 0. Denote by D_i the distance from r to i. Without loss of generality, we assume that the depot is the leftmost vertex on the line. Otherwise, we may decompose the problem into the two sides of the depot since clients on one side do not affect those on the other side.

$$\begin{aligned}
\min \quad & \sum_{(i,t)\in D(V\times[T])} \sum_{s=1}^{T} H^i_{s,t} x^i_{s,t} + \sum_{s=1}^{T}\sum_{i=0}^{N} D_i y^i_s & & \\
\text{s.t.} \quad & \sum_{s=1}^{T} x^i_{s,t} = 1 & \forall (i,t) \in D(V\times[T]) & \quad (3.5)\\
& \sum_{j=i}^{N} y^j_s \geq x^i_{s,t} & \forall (i,t) \in D(V\times[T]), s \in [t] & \quad (3.6)\\
& x^i_{s,t} \geq 0 & \forall (i,t) \in D(V\times[T]), s \in [t] & \quad (3.7)\\
& y^i_s \geq 0 & \forall i \in [0,N], s \in [T]. & \quad (3.8)
\end{aligned}$$

The variable $x^i_{s,t}$ indicates if (i,t) is served at time s. The variable y^i_s indicates if the farthest client visited at time s is i. The first constraint ensures that each (i,t) is served by time t. The second constraint ensures that the path at time s passes through i if (i,t) is served at time s for some $t \geq s$.

We introduce dual variables b^i_t for the first primal constraint and $\beta^i_{s,t}$ for the second constraint. The dual LP is:

$$\max \sum_{(i,t)\in D(V\times[T])} b_t^i$$

$$\text{s.t} \quad b_t^i \le H_{s,t}^i + \beta_{s,t}^i \qquad \forall (i,t) \in D(V\times[T]), s \in [t] \quad (3.9)$$

$$\sum_{(j,t)\in D(V\times[T]):j\le i,t\ge s} \beta_{s,t}^j \le D_i \qquad \forall i \in [0,N], s \in [T] \quad (3.10)$$

$$\beta_{s,t}^i \ge 0 \qquad \forall (i,t) \in D(V\times[T]), s \in [t]. \quad (3.11)$$

The variable $b_{s,t}^i$ represents the budget (i,t) can pay for a visit. The variable $\beta_{s,t}^j$ represents the share of payment (j,t) contributes to a visit at time s that passes through (possibly beyond) j.

3.4.2 Basic Primal Dual Phase and Analysis

Before stating the algorithm formally, we give an overview of the ideas. We interpret of the dual variables b_t^i as the total budget that demand (i,t) has available to pay for potential visits who may serve it. We think of the dual variables $\beta_{s,t}^i$ as visit-specific payments that (i,t) may want to offer for a visit at time s to serve it. However, since $\beta_{s,t}^i$ variables are not part of objective function in the dual LP, we cannot directly use $\beta_{s,t}^i$ to pay for visits. Instead, $\beta_{s,t}^i$ represent copies of the total budget b_t^i, one copy for each s. In particular, the number of visits s for which $\beta_{s,t}^i > 0$ is an upper bound on the number of times that the whole budget b_t^i of (i,t) is overused. The general framework is to raise the budgets of demands as long as all constraints in the dual LP are able to hold. The final values of the budgets are determined by the tightening of dual constraints that they are involved in.

At the beginning of the algorithm, all budgets b_t^i and visit-specific payments $\beta_{s,t}^i$ are to start at 0. We introduce a continuous parameter τ that slides through time from T to 1 at a constant rate. The position of τ within the time horizon will determine what value to raise the budgets and visit-specific payments. Whenever τ passes through an integral time t (i.e. $\tau < t$), it "wakes up" the budgets b_t^i of demands (i,t) occurring at time t. Those b_t^i shall increase at the same rate that $H_{\tau,t}^i$ is increasing as τ is sliding towards 1, i.e., we keep b_t^i at exactly the same value as $H_{\tau,t}^i$. The definition of $H_{\tau,t}^i$ for non-integral τ is interpolated linearly, i.e., define $H_{\tau,t}^i = (1 - \tau + \lfloor\tau\rfloor)H_{\lfloor\tau\rfloor,t}^i + (\tau - \lfloor\tau\rfloor)H_{\lceil\tau\rceil,t}^i$.

Observe that keeping $b_t^i = H_{\tau,t}^i$ ensures that each demand (i,t) can at least pay for the holding cost from time τ to t. To maintain feasibility to the dual constraints, we also raise $\beta_{s,t}^i$ as needed to keep constraint 3.9 satisfied. That means for each demand (i,t) and each $s \in (\tau,t]$, we raise the value of $\beta_{s,t}^i$ to exactly $H_{\tau,t}^i - H_{s,t}^i$. As the visit-specific payments are increased, constraint 3.10 will start becoming tight (satisfied at equality) for some (i,s). Whenever a constraint is tight at (i,s), the algorithm shall create a visit up to D_i at time s. We may think of using the $\beta_{s,t}^i$ in the left hand sum of the constraint to pay for the visit up to D_i at time s. However, as mentioned earlier, the $\beta_{s,t}^i$ cannot be used directly to pay for visits. Rather, each budget b_t^i may need to be used many times to pay for all of the visits created while we want to use each budget only a constant number of times. We will repair this issue by a selecting only

a subset of the visits and extending the length of those visits, which we will describe in the pruning phase.

Now we formally define the primal dual algorithm.

As τ moves from T to 1, keep all unfrozen $b_t^i = H_{\tau t}^i$.

1. Initialize $b_t^i = 0$ and $\beta_{s,t}^i = 0$ for all $(i,t) \in D(V \times [T])$ and $s \leq t$.
2. Whenever τ reaches some s, raise β_{st}^i at the same rate as b_t^i for all $i \in [N], t \geq s$ so that constraint 3.9 continues to hold.
3. Whenever there is some i^*, s^* such that the second dual constraint is tight, for each such (i^*, s^*), freeze $\beta_{s^*t}^j$ for all $j \leq i^*, t \geq s^*$ (which freezes b_t^j for all $j \leq i^*$, $t \geq s^*$, which freezes β_{st}^j for all $j \leq i^*, t \geq s^*, s \leq \tau$). Visit up to i^* at time s^*.
4. If $\tau < 1$, set $\beta_{11}^i = D_i - D_{i-1}$ for all $i \in [N]$ such that $(i,1) \in D(V \times [T])$, and $b_1^i = \beta_{11}^i$. Freeze remaining b_t^i and β_{st}^i.
5. Let $\tilde{i}$ be the farthest client such that there exists $t \in [T]$ for which $(\tilde{i}, t)$ has not been served by the visits so far. Visit up to $\tilde{i}$ at time 1.

Next, we define some terms to be used in our analysis. For visit s, let $i(s)$ be the farthest client visited at time s. For demand point (i,t), let $s(i,t)$ be the latest visit through i by time t. Let $freeze(i,t)$ be the value of τ when b_t^i froze, which need not be integer. Let $\mu(i,t)$ be the number of visits $s \in [freeze(i,t), t]$ such that $i(s) \geq i$. Observe that demand point (i,t) has positive $\beta_{s,t}^i$ only if $s \in [freeze(i,t), t]$ and $i(s) \geq i$. We say that (i,t) *contributes* to a visit s if $\beta_{s,t}^i > 0$. We call $[freeze(i,t), t]$ the *active interval of* (i,t) since b_t^i is growing only when $\tau \in [freeze(i,t), t]$. The *freezing visit of* (i,t) is the visit s whose tightening of constraint 3.10 at $(i(s), s)$ caused b_t^i to freeze. For every demand point (i,t), denote the freezing visit of (i,t) by $s'(i,t)$.

We visualize our primal dual solution by a plot of the set of all active intervals with $[freeze(i,t), t]$ plotted at location D_i on the y-axis spanning its corresponding interval along the x-axis. We assume that distances are in increasing order downwards and times are in increasing order rightwards. We also plot each visit from the primal dual solution at time s down to $D_{i(s)}$. In the following figure, we show an example of a plot of active intervals with the visits grown by our primal dual algorithm.

We state some simple observations that will help with our analysis.

Proposition 3.4.1. *Given a demand point (i,t), we have $s'(i,t) \in [freeze(i,t), t]$.*

Proof. First, $s'(i,t) \leq t$ since $s'(i,t)$ freezes only the budges of demand points whose time is at least $s'(i,t)$. Let $(i', s'(i,t))$ be the dual constraint whose tightening froze b_t^i. Then $i' \geq i$. Suppose $s'(i,t) < freeze(i,t)$. Then when $\tau = freeze(i,t)$, $\beta_{s'(i,t),r}^j = 0$ have not grown for all $j \leq i'$ and $r \geq s'(i,t)$, contradicting that the constraint at $(i', s'(i,t))$ became tight at $\tau = freeze(i,t)$. □

Corollary 3.4.2. *For each demand point (i,t), the holding cost of (i,t) from our primal dual solution is at most b_t^i.*

Proof. For each (i,t) whose $freeze(i,t) > 1$, it is being served at or after $s'(i,t) \geq freeze(i,t)$, which means its holding cost is $H_{s(i,t),t}^i \leq H_{s'(i,t),t}^i <= H_{freeze(i,t),t}^i = b_t^i$.

For (i,t) whose $freeze(i,t) = 1$, step 5 of the primal dual algorithm ensures that $s(i,t) \in [1 = freeze(i,t), t]$, which means the holding cost of (i,t) is $H_{s(i,t),t}^i \leq H_{freeze(i,t),t}^i = b_t^i$. □

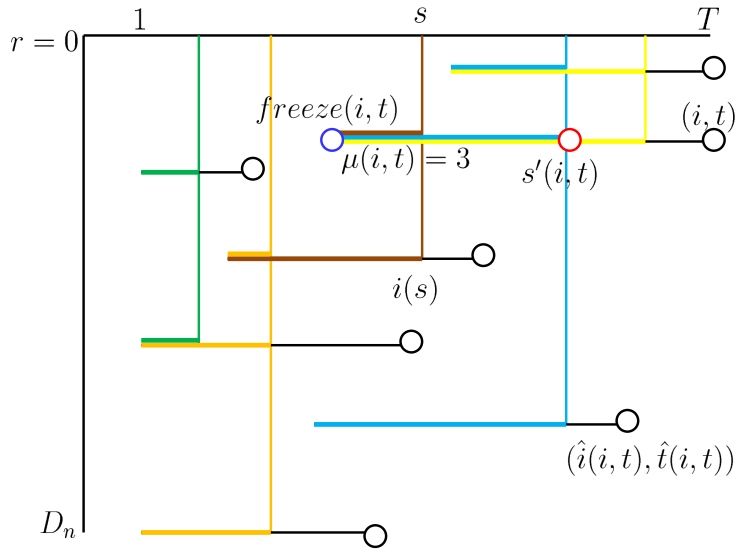


FIGURE 3.2: The red dots at location D_i time s indicate that visit s is the freezing visit of the demand point (i,t) whose $[freeze(i,t),t] \ni s$. The horizontal colored portion of each active interval $[freeze(i,t),t]$ represents the amount $\beta^i_{s,t} = H^i_{freeze(i,t),t} - H^i_{s,t}$ that (i,t) contributes to the routing cost of visit s of the same color.

Proposition 3.4.3. *Given a demand point (i_1,t_1), for all $i_2 > i_1$, if $t_2 \leq t_1$, then $freeze(i_2,t_2) \leq freeze(i_1,t_1)$.*

Proof. Assume $i_2 > i_1$ and $t_2 \leq t_1$. At $\tau = freeze(i_2,t_2)$, the freezing visit $s'(i_2,t_2)$ of (i_2,t_2) would also cause $b^{i_1}_{t_1}$ to freeze if it was not already frozen since $i_1 \leq i_2$ and $t_1 \geq t_2$, i.e., $freeze(i_1,t_1) \geq freeze(i_2,t_2)$. □

Proposition 3.4.4. *Given demand points (i_1,t_1) and (i_2,t_2) such that $i_2 \geq i_1$ and $freeze(i_2,t_2) > freeze(i_1,t_1)$, $s'(i_2,t_2) > t_1$.*

Proof. Assume $i_2 \geq i_1$ and $freeze(i_2,t_2) > freeze(i_1,t_1)$. If $s'(i_2,t_2) \leq t_1$, then $s'(i_2,t_2)$ would cause $b^{i_1}_{t_1}$ to freeze when $\tau = freeze(i_2,t_2)$ since $t_1 \geq s'(i_2,t_2)$ and $i_1 \leq i_2$, contradicting that $freeze(i_2,t_2) > freeze(i_1,t_1)$. □

Let (x,y) be the primal solution from our primal dual algorithm. Let OPT be the cost of an optimal solution. We will show that our holding cost is at most OPT and our routing cost is at most $\max_{(i,t)\in D(V\times[T])}(\mu(i,t)+1)OPT$.

Lemma 3.4.5. *Let $h(x,y)$ and $r(x,y)$ be the holding cost and routing cost of (x,y) respectively. Then $h(x,y) \leq OPT$ and $r(x,y) \leq \max_{(i,t)\in D(V\times[T])}(\mu(i,t)+1)OPT$.*

Proof. The holding cost is

$$\begin{aligned} h(x,y) &= \sum_{(i,t)\in D(V\times[T])} H^i_{s(i,t),t} \\ &\le \sum_{(i,t)\in D(V\times[T])} H^i_{freeze(i,t),t} \text{ by Proposition 3.4.1} \\ &\le \sum_{(i,t)\in D(V\times[T])} b^i_t \text{ by definition of } freeze(i,t) \\ &\le OPT. \end{aligned}$$

The routing cost is

$$\begin{aligned} r(x,y) &= \sum_{s=2}^{T} D_{i(s)} + D_{i(1)} \\ &\le \sum_{s=2}^{T} \sum_{(j,t)\in D(V\times[T]):j\le i(s),[freeze(j,t),t]\ni s} \beta^j_{st} + \max_{(i,t)\in D(V\times[T])} D_i \text{ by definition of freeze operation} \\ &= \sum_{(i,t)\in D(V\times[T])} \sum_{s\ge 2: s\in[freeze(i,t),t]: i\le i(s)} \beta^i_{st} + OPT \\ &\le \sum_{(i,t)\in D(V\times[T])} \sum_{s\ge 2: s\in[freeze(i,t),t]: i\le i(s)} b^i_t + OPT \text{ by dual feasibility} \\ &\le \sum_{(i,t)\in D(V\times[T])} \mu(i,t) b^i_t + OPT \\ &\le \max_{(i,t)\in D(V\times[T])} (\mu(i,t)+1) OPT. \end{aligned}$$

□

3.4.3 Example with High Routing Cost

From the analysis in the previous subsection, the routing cost depends on the number of times $\mu(i,t)$ that b^i_t is used. We show an example where the overuse of a budget b^i_t is as many as T times, although the actual primal cost is $\Theta(T/\log T)$ times the total dual budget available.

Example 1.
Consider an instance with T time units and only one client at distance D from r. Assume the demand is one each time i.e. $d_t = 1$ for all $t \in [T]$. Set holding costs by $h_{1,t} = D/(T-t+1)$ for all $t \in [T]$, and $h_{s,t} = D/(T-t+1) - D/(T-s+1)$ for all $s \ge 2$, $t \ge s$. Then as τ slides from T to 2, no constraint tightens. When τ reaches 1, all of the constraints tighten. So the client gets a visit at all times $1, \ldots, T$. In particular, $\beta_{s,T} > 0$ for all $s \in [T]$, i.e. b_T is used T times in our bound for the routing cost. The described behavior of primal dual on this instance can verified by checking that for all integer $t > \tau$, we have $H_{\tau,t} + \sum_{j=1}^{T} H_{\tau,t+j} - H_{t,t+j} \le D$, which holds with equality only at $\tau = 1$. Here, our dual budget is $\sum_{t=1}^{T} b_t = D + D/2 + \ldots + D/(T-1) + D = \Theta(D \log T)$. The primal cost of our T visits is DT. So the overhead cost relative to our available budget is $\Theta(T/\log T)$.

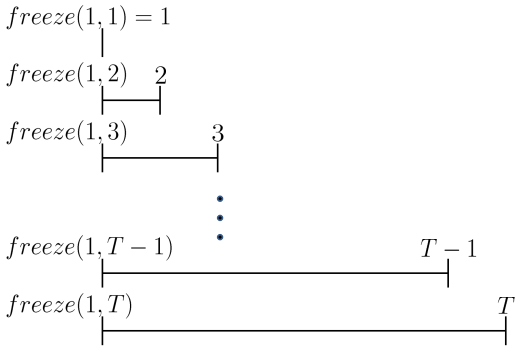


FIGURE 3.3: The above shows the active intervals of the instance in Example 1.

3.4.4 Pruning Phase and Analysis

As we saw in the example, the dual budget grown from our primal dual algorithm cannot always pay for the total cost of its corresponding primal solution. We now modify our primal dual solution to obtain a constant bound. Let M be the smallest integer such that $D_N \leq 2^M$. For all $i \in [M]$, let R_i be the rectangular region spanning $(2^{i-1}, 2^i]$ along the distance axis and spanning $[1, T]$ along the time axis. We write $R_i = [1, T] \times (2^{i-1}, 2^i]$ and let $R_0 = [1, T] \times [0, 1]$. The idea of our pruning procedure is to select a sparse set of visits for each region R_i, copy the original visits to a second visit time, and extend their lengths to 2^i so that every demand point within R_i is served by some visit within its active interval. This will keep the holding cost within OPT. We then refine our analysis so that each b_t^i is charged at most three times. In addition, our charging scheme will ensure that each visit's routing cost is paid for within one eighth of its total length. So the total dual budget will pay for one twenty-fourth of the total routing cost after accounting for using each b_t^i three times. Then the holding cost and routing cost together will incur a factor of twenty-six, after accounting for the extra route at time 1 in the last step of the primal dual phase.

Here, we define some notation we will need in our pruning algorithm. Given $(i, t) \in D(V \times [T])$, let $\hat{i}(i, t)$ be the first client at i or farther such that the dual constraint at $(\hat{i}(i, t), s'(i, t))$ became tight during the primal dual algorithm. Let $\hat{t}(i, t)$ be a demand time such that $(\hat{i}(i, t), \hat{t}(i, t))$ contributes to $s'(i, t)$. Observe that $freeze(i, t) = freeze(\hat{i}(i, t), \hat{t}(i, t))$ by definition of $\hat{i}(i, t)$.
The following modification of the visits grown from the primal dual algorithm will yield a 26-approximation.

1: Initialize $I_1, \ldots, I_M, S_1, \ldots, S_M$ each to $\emptyset$.
2: **for** $j \leftarrow M$ to 1 **do**
3: Greedily add to I_j the active interval $[freeze(i, t), t] \times \{D_i\} \in R_j$ with the largest $freeze(i, t) > 1$ having no time in common with the existing active intervals in I_j such that $[freeze(i, t), t] \not\ni s$ for every $s \in \cup_{k=j+1}^{M} S_k$ until no such active interval exists in R_j.
4: **for** each $[freeze(i, t), t] \times \{D_i\} \in I_j$ **do**
5: add to S_j two visits at times $\lceil freeze(i, t) \rceil$ and $s'(i, t)$, each up to distance $\max\{2^j, D_{\hat{i}(i,t)}\}$.
6: **end for**

7: **end for**
8: Return $S := \cup_{j=1}^{M} S_j \bigcup \{\{1\} \times [0, D_{\tilde{i}}]\}$.

Denote the holding cost of S by $h(S)$ and the routing cost of S by $r(S)$.

Proposition 3.4.6. $h(S) \leq OPT$.

Proof. Let $(i,t) \in D(V \times [T])$. Then there is some $j \in [M]$ such that $[freeze(i,t), t] \times \{D_i\} \in R_j$. First, assume that $freeze(i,t) > 1$. If $[freeze(i,t), t] \times \{D_i\} \in I_j$, then (i,t) is served at time $s'(i,t)$ or later, which incurs a holding cost of $H^i_{s'(i,t),t} \leq b^i_t$. Now assume that $[freeze(i,t), t] \times \{D_i\} \notin I_j$. Then there is some $(i',t') \in I_j$ such that $[freeze(i,t), t] \cap [freeze(i',t'), t'] \neq \emptyset$, i.e., $freeze(i,t) \leq \lceil freeze(i',t') \rceil \leq t'$ by our construction of I_j. Since $S_j \ni t'$ and t' visits up to at least 2^j, demand point (i,t) is served at time $\lceil freeze(i',t') \rceil$ or later.

Second, if $freeze(i,t) = 1$, then the visit at time 1 to $D_{\tilde{i}}$ serves (i,t) within $[freeze(i,t), t]$. So (i,t) incurs a holding cost of at most $H^i_{t',t} \leq H^i_{\lceil freeze(i',t') \rceil, t} = b^i_t$. Hence $OPT \geq \sum_{(i,t) \in D(V \times [T])} b^i_t$ is sufficient to pay for $h(S)$. □

Theorem 3.4.7. $r(S) \leq 25OPT$.

Proof. For the visit at time to distance $D_{\tilde{i}}$, we charge on copy of OPT. Now, we state which portion of the total budget we will charge for each visit after time 1 in S. Let $s \in S$. Let (i,t) be the demand point such that $s \in \{\lceil freeze(i,t) \rceil, s'(i,t)\}$, where $[freeze(i,t), t] \times \{D_i\} \in I_j$ for some j. Let $\hat{j}$ be the index such that $D_{\hat{i}(i,t)} \in (2^{\hat{j}-1}, 2^{\hat{j}}]$. Since the two visits $\lceil freeze(i,t) \rceil$ and $s'(i,t)$ reach the same distance, we will focus only on paying for the visit at time $s'(i,t)$ and account for the extra factor of 2 for having both visits in the final solution.

To pay for $s'(i,t)$, we will charge the budget of each demand point within $R_{\hat{j}} \cup R_{\hat{j}-1}$ whose active interval is crossed by $s'(i,t)$. More formally, let $P_{s'(i,t)} = \{(i',t') \in D(V \times [T]) \cap (R_{\hat{j}} \cup R_{\hat{j}-1}) : [freeze(i',t'), t'] \ni s'(i,t), i' \leq \hat{i}(i,t)\}$. Assume for now that $2^j \leq D_{\hat{i}(i,t)}$. If not, then we pay an extra factor of 2 for visiting up to 2^j since $D_{\hat{i}(i,t)} \geq D_i \geq 2^{j-1}$ (since $(i,t) \in R_j$). We will pay for $s'(i,t)$ up to distance $D_{\hat{i}(i,t)}$ within a factor of 2 by charging the dual solution $B_{s'(i,t)} := \sum_{(i',t') \in P_{s'(i,t)}} \beta^{i'}_{s'(i,t),t'}$. Now, we show that $B_{s'(i,t)} \geq D_{\hat{i}(i,t)}/2$. Observe that the only other demand points who could potentially contribute to visit $s'(i,t)$ are those in $P_{\tilde{s'(i,t)}} := \{(i',t') \in D(V \times [T]) \cap (R_0 \cup \cdots \cup R_{\hat{j}-2}) : [freeze(i',t'), t'] \ni s'(i,t), i' \leq \hat{i}(i,t)\}$. Let $B_{\tilde{s'(i,t)}} = \sum_{(i',t') \in P_{\tilde{s'(i,t)}}} \beta^{i'}_{s'(i,t),t'}$. We claim that $B_{\tilde{s'(i,t)}} \leq D_{\hat{i}(i,t)}/2$, which would imply $B_{s'(i,t)} \geq D_{\hat{i}(i,t)}/2$ as a consequence. To see why $B_{\tilde{s'(i,t)}} \leq D_{\hat{i}(i,t)}/2$, suppose for contradiction that $B_{\tilde{s'(i,t)}} \geq (D_{\hat{i}(i,t)}/2) + \delta$ for some $\delta > 0$. Since $D_{\hat{i}(i,t)} \geq 2^{\hat{j}-1}$, we have $B_{\tilde{s'(i,t)}} \geq 2^{\hat{j}-1} + \delta$. Let $(i_0, t_0) = \text{argmax}_{(i',t') \in P_{\tilde{s'(i,t)}}} D_{i'}$. By our primal dual algorithm, while τ is moving towards 1, there exists $(i',t') \in P_{\tilde{s'(i,t)}}$ such that $\beta^{i'}_{s'(i,t),t'}$ has not froze only if $B_{\tilde{s'(i,t)}} < D_{i_0}$ since as soon as $\sum_{(i',t') \in D(V \times [T]): i' \leq i_0, t' \geq s'(i,t)} \beta^{i'}_{s'(i,t),t'}$ reaches D_{i_0}, the budget $b^{i'}_{t'}$ freezes for all (i',t') such that $i' \leq i_0$ and $t' \geq s'(i,t)$. Since $B_{\tilde{s'(i,t)}} \leq \sum_{(i',t') \in D(V \times [T]): i' \leq i_0, t' \geq s'(i,t)} \beta^{i'}_{s'(i,t),t'}$, we have $B_{\tilde{s'(i,t)}}$ stays bounded above by $D_{i_0} \leq 2^{\hat{j}-2}$ during all values of τ when as τ lowers from T to 1. Hence $B_{s'(i,t)} \geq D_{\hat{i}(i,t)}/2$, which means that $B_{s'(i,t)}$ is sufficient to pay for $s'(i,t)$ within a factor of 4 (in case $2^j > D_{\hat{i}(i,t)}$). So we have a routing factor of 8 so far for paying for the

actual visits of S, which include both $\lceil freeze(i,t) \rceil$ and $s'(i,t)$ for each (i,t) whose active interval is in $\cup_{j=1}^{M} I_j$.

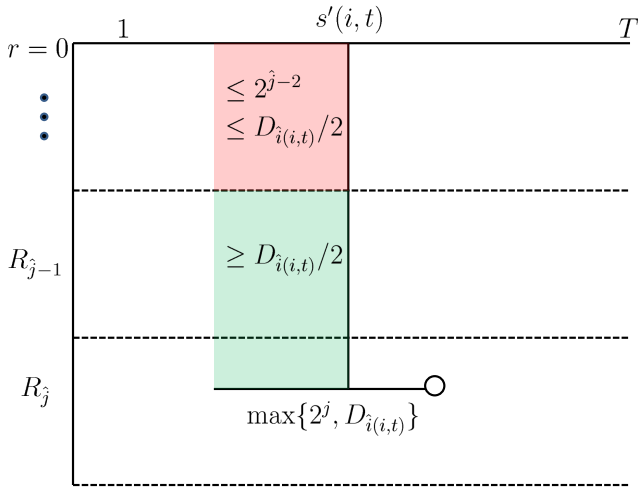


FIGURE 3.4: $\beta_{s'(i,t)} \geq D_{\hat{i}_{(i,t)}}/2$.

Now, we will show that each b_t^i is charged at most three times for the freezing visits $s'(i,t)$ by the above charging scheme that uses $B_{s'(i,t)}$ to pay for $s'(i,t)$. For each $j \in [M]$, let F_j be the set of freezing visits in S_j, i.e., $F_j := \{s'(i,t) : (i,t) \in R_j \text{ and } [freeze(i,t),t] \times \{D_i\} \in I_j\}$. Fix $(i,t) \in D(V \times [T])$ and let j be the index such that $(i,t) \in R_j$. More formally, we will show that there are at most three visits $s_1, s_2, s_3 \in \cup_{k=1}^{M} F_k$ such that $\beta^i_{s_l,t} > 0$ for all $l \in [3]$. Suppose for contradiction that (i,t) contributes to at least four visits $s_1, \ldots, s_4 \in \cup_{k=1}^{M} F_k$. For each $l \in [4]$, let (i_l, t_l) be the demand point whose active interval is in $\cup_{k=1}^{M} I_k$ such that $s'(i_l,t_l) = s_l$.

First, we show that no two s_ls are in the same F_k. Suppose that there is some $k \in [M]$ such that $s_l < s_m \in F_k$ for some $l, m \in [4]$. Then $s_l, s_m \in [freeze(i,t),t]$. Since $\hat{i}(i_l,t_l), \hat{i}(i_m,t_m) \geq i$, we have $freeze(\hat{i}(i_l,t_l), \hat{t}(i_l,t_l)), freeze(\hat{i}(i_m,t_m), \hat{t}(i_m,t_m)) \leq freeze(i,t)$ by Proposition 3.4.4. Then $freeze(i_m,t_m) = freeze(\hat{i}(i_m,t_m), \hat{t}(i_m,t_m)) \leq freeze(i,t) \leq s_l \leq t_l$. So $[freeze(i_m,t_m),t_m] \cap [freeze(i_l,t_l),t_l] \neq \emptyset$, which contradicts that the intervals in I_k are time-disjoint. Hence $s_1, \ldots, s_4$ are in disjoint F_ks.

Finally, we show that there cannot be four such visits. By our charging scheme, (i,t) pays for s_l only if $(\hat{i}(i_l,t_l), \hat{t}(i_l,t_l)) \in R_j \cup R_{j+1}$. Then no s_l is in $F_{j+2} \cup \cdots \cup F_M$ since $\hat{i}(i_l,t_l) \geq i_l$. So $s_1, \ldots, s_4 \in F_1 \cup \cdots \cup F_{j+1}$. Since each F_k contains at most one s_l among $s_1, \ldots, s_4$, there must be at least two of them in $F_1 \cup \cdots \cup F_{j-1}$. Without loss of generality, assume that $s_4 \in F_p$ and $s_3 \in F_q$ where $1 \leq p < q \leq j-1$. By the pruning algorithm, there are visits at times $\lceil freeze(i_3,t_3) \rceil$ and s_3 up to distance $D_{\hat{i}(i_3,t_3)} \in (2^{j-1}, 2^{j+1}]$. Since $[freeze(i_4,t_4),t_4] \in I_p$ and $p < j$, the interval $[freeze(i_4,t_4),t_4]$ has no overlap with times $\lceil freeze(i_3,t_3) \rceil$ and s_3. Since (i,t) contributes to s_4, we have $t_4 \geq s_4 \geq freeze(i,t)$. Observe that $freeze(i_4,t_4) = freeze(\hat{i}(i_4,t_4), \hat{t}(i_4,t_4)) \leq freeze(i,t)$ because $\hat{i}(i_4,t_4) \geq i$. Then $freeze(i_4,t_4) \in (freeze(i_3,t_3), freeze(i,t)]$. Since $[freeze(i_4,t_4),t_4] \not\ni s_3$, we have $t_4 < s_3 \leq t_3$. So $[freeze(\hat{i}(i_4,t_4), \hat{t}(i_4,t_4)] \in (freeze(i_3,t_3),t_3)$ while $\hat{i}(i_4,t_4) > i_3$ by $j > q$, which contradicts Proposition 3.4.3. Hence each (i,t) is charged for at most three visits in $\cup_{k=1}^{M} F_k$. This produces an extra factor of 3 on top of the 8 we had earlier for paying

for S. So the final routing cost of S is paid for by 24 copies of $\sum_{(i,t)\in D(V\times[T])} b_t^i \leq OPT$, i.e., $r(S) \leq 25OPT$ after accounting for the visit at time 1. □

3.5 Open Questions

The main open problem is improving the $O(\frac{\log T}{\log\log T})$ approximation for IRP on general metrics or showing hardness of approximation. It is unknown whether IRP on the grid admits a constant factor approximation. It is also interesting to find constant factor approximation on circularly decomposable metrics, which generalizes tree metrics.

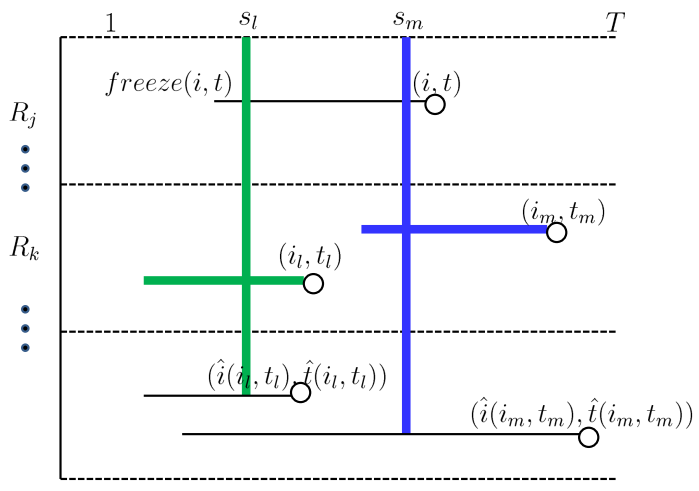


FIGURE 3.5: At most one freezing visit charges b_t^i per F_k.

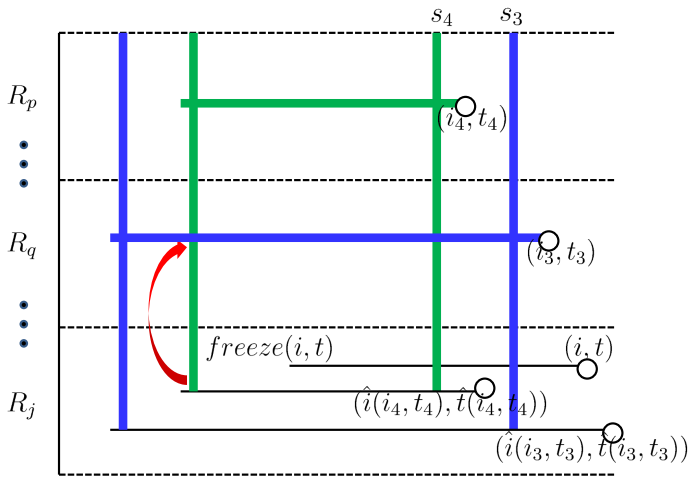


FIGURE 3.6: At most three freezing visits total charge b_t^i.

Chapter 4

Combinatorial Heuristics for Inventory Routing on General Metrics

4.1 Introduction

The Inventory Routing Problem (IRP) arises from Vendor Managed Inventory systems in which a product supplier and its retailers cooperate in the inventory planning. First, the retailers share with the supplier the demand patterns for its product and the storage costs for keeping early deliveries per retailer location. Then the supplier is responsible for planning a delivery schedule that serves all the demands on time. Naturally, the supplier wishes to minimize its routing cost and storage cost over the time horizon. This optimization problem is called IRP and has been studied extensively [23, 29, 25, 26].

In the classical single-depot IRP, a set of client demand locations in a metric containing the depot is given, and for a planning horizon of T days, a daily demand at each client location is specified. The goal is to come up with vehicle routing schedules in each of the T days to stock the client demands before they materialize. However, early stocking at a location incurs a location- and duration-specific *inventory* holding cost that are also specified. If we assume the daily replenishing vehicle has infinite capacity, the distance traveled by the vehicle in a daily route translates to a *routing* cost. The goal of IRP is to find daily vehicle schedules for the T days that deliver enough supply at each location to meet all daily demands and minimizes the sum of inventory holding costs for units supplied ahead of their demand and the daily routing costs of the vehicle, over the T days.

Problem Definition. We now give the formal definition of IRP. In *IRP*, we are given a complete graph $K_n = (V, E)$ whose vertices are potential locations of clients and whose edge weights are determined by a *metric* $w : E \to \mathbb{R}_{\geq 0}$ ($w_{xz} \leq w_{xy} + w_{yz}$ for all $x, y, z \in V$). In a *graph metric*, the distance between every pair of vertices is the length of the shortest path between them in a given weighted graph. There is a depot $r \in V$ from which a vehicle of infinite capacity loads supply to drop off to clients. The vehicle may carry any amount of supply from the depot at each time. We have a discrete time horizon $1, \ldots, T$ over which client $v \in V$ demands $d_t^v \geq 0$ units of supply to be delivered to it by time t. For each client $v \in V$, demand time $t \in [T]$, and potential serving time $s \leq t$, storing one unit of supply at v during time $[s, t]$ incurs a holding cost of $h_{s,t}^v$. We denote by $D(V \times [T])$ the set points (v, t) such that $d_t^v > 0$. When the context of V and $[T]$ are clear, we use D and $D(V \times [T])$ interchangeably. We call such points *demand points*. The objective is to select a tour from r through a subset of clients per time $t \in [T]$ to satisfy every demand point (no late delivery

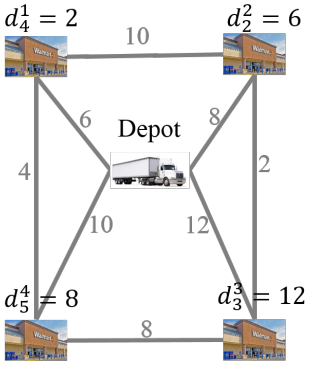


FIGURE 4.1: In this instance, we have four stores over five days. A demand at store v on day t is labeled d_t^v. The distances are labeled next to each edge. The holding cost is linear, i.e., $H_{s,t}^v = (t-s)d_t^v$.

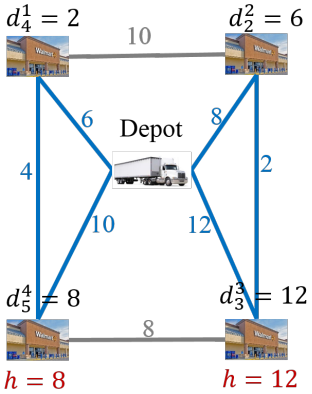


FIGURE 4.2: A possible feasible solution is to serve stores 2 and 3 on day 2 and stores 1 and 4 on day 4. Since stores 3 and 4 are served one day before their demands are due, they incur holding costs of 12 and 8, respectively. The route on day 2 has cost 22, and the route on day 4 has cost 20. The total IRP cost for this solution is 62.

allowed) so that the total routing cost and holding cost over $[T]$ is minimized. Denote by $H_{s,t}^v$ the holding cost incurred if d_t^v is served at time s, i.e., $H_{s,t}^v = h_{s,t}^v d_t^v$. We remark that there is always an optimal solution such that each d_t^v is served at a single time, for if d_t^v is delivered in separate portions at times $s_1 < \ldots < s_l$, then the total cost does not increase if we move all of d_t^v to be delivered at time s_l. This is due to the infinite capacity available at the delivery vehicle. In Figure 4.1 and Figure 4.2, we demonstrate an example of an IRP instance and a feasible solution for the instance.

A related problem, called Prize-Collecting Steiner Tree (PCST), will be crucially used in our heuristics for obtaining IRP solutions. The *Prize-Collecting Steiner Tree* problem has as input a graph $G = (V, E)$ with root r, edge weights $w : E \to \mathbb{R}_{\geq 0}$ and vertex penalties $\pi : V \to \mathbb{R}_{\geq 0}$. The goal is to find a tree visiting some subset of vertices minimizing the total edge cost of the tree and the penalties of vertices not spanned by the tree. The closely related Prize-Collecting Traveling Salesman Problem (PCTSP) is to find a tour instead of a tree with the same specifications.

Although existing computational research on IRP has been extensive, the instance sizes solved are still limited. So far, the largest instance sizes studied have only 200 clients over 6 days, detailed in Section 4.2. There have not been new conceptual ideas beyond refinements of traditional integer programming methods e.g. branch and cut [7], branch cut and price [37], and matheuristics [8][6].

The unifying theme of our heuristics is to reduce the search space of IRP by creating and solving PCTSP instances as intermediate steps to determine which clients to visit each day. PCST is a suitable intermediate problem because it is much faster to solve than IRP since it does not involve the inter-temporal constraints of IRP. In practice, PCST problems can be solved quickly to near-optimality over $200,000$ nodes [**LLLS18**]. Additionally, PCST is able to capture the challenge of IRP's trade off between holding cost and routing cost by using the trade off between routing cost and penalty cost in its own objective, even though it does not have the multi-period nature of IRP. Each of our heuristics will convert the holding cost of IRP over the whole planning horizon to penalties in PCST so that the PCST solutions eventually form a good IRP solution. Next, we explain in more detail our contributions to applying ideas from PCST solutions in deriving better computational results for IRP.
Contributions.

1. We exploit a recently discovered conceptual reduction of periodic IRP to PCST [49] and extend the ideas to general IRP.

2. We design a new suite of algorithms for general IRP using this reduction to look for local improvements; In particular, we define a very large neighborhood search step and reduce it to a PCST problem.

3. We design a new greedy construction heuristic using a reduction of the greedy step to a PCST problem.

4. We adapt the primal dual method to design a primal construction heuristic. While the algorithm proceeds using a reverse waveform method introduced in earlier work of Levi et al. [67] , we introduce a new additional step of choosing routes in each period by solving an appropriately defined PCST.

5. We implement all the above ideas and compare their performance using a data generation model of Archetti et al. [8] without the capacity constraints on the vehicles and clients, resulting in much faster solving time while still obtaining low gaps.

Techniques. Our heuristics are inspired by the theoretical work of Fukunaga et al.'s [49] approximation algorithms for the periodic case of IRP. In the *periodic* IRP, we are given additionally a set of frequencies $f_0, \ldots, f_k$ to assign to the clients. Assigning a client v to frequency f means that v must be visited exactly every f days. A feasible solution will choose a frequency from the available set for each client and produce a delivery schedule that obeys the assigned frequencies. Thus, the periodic IRP is more restrictive than the general IRP we consider here. Fukunaga et al. exploit the restrictiveness of periodic schedules by reducing the periodic IRP to the Prize-Collecting Steiner Tree (PCST) problem such that the holding costs are simulated by the penalties of PCST. The idea of the reduction is to create a copy of the input graph per frequency f_i. For the ith copy of the graph, scale the edge costs by roughly T/f_i because the clients assigned frequency f_i are to be visited $\lfloor T/f_i \rfloor$ many times in the schedule. To capture the holding costs of periodic IRP, they set the penalties $p(v_j)$ so that $\sum_{j=0}^{i-1} p(v_j)$ is the holding cost for v if v is visited every f_i days (Section 4.3.1 contains an example of how we adapt this to our setting). In this way, the connection cost and penalty cost of the PCST instance correspond to the routing cost and holding cost of the periodic IRP instance. See [49] for more details.

We adapt the ideas of [49] to propose three types of heuristics that each take advantage of solving the easier PCST problem. First, our local search heuristics use

PCST to quickly perform large neighborhood search among the potential improvements per round. Adaptive large neighborhood search has been used on IRP with Transshipments [**CCL12a**] and Multi-vehicle IRP [**CCL12b**]. Large neighborhood search has also been investigated on other variants of IRP [81, 2, 54].

Second, our greedy heuristic uses PCST to determine the best density demand set to cover each round. Finally, the primal-dual heuristic uses PCST to guide the growth of the dual values and identify the set of demands to serve. We describe the motivation for using PCSTs in Section 4.3. The best of our heuristics achieves optimality ratios between 1.04 and 1.08 (across various problem settings), and yet was able to solve instances of size 140 clients over 6 days and 50 clients over 14 days within 2.01 seconds and under 1 second, respectively. They are still able to solve even larger instances quickly, but we do not have lower bounds to evaluate their solution quality for instances larger than the aforementioned sizes. Both of these solution times are over three orders of magnitude faster than solving a single commodity flow MIP model of IRP, which requires over 2837 seconds and 1481 seconds, respectively, to solve within 10% MIPGap.

Chapter Outline. The remaining sections are organized as follows. In Section 4.2, we describe related work. Section 4.3 describes three local search heuristics. Section 4.4 provides a greedy heuristic. Section 4.5 presents a primal dual heuristic. In Section 4.6, we describe a single commodity flow MIP formulation of IRP to attain lower bounds for the costs of optimal solutions to compare with the heuristics. Finally, Section 4.7 shows the computational results of the various heuristics across a range of input parameters.

4.2 Related Work

Methods for Capacitated IRP. For the capacitated single-vehicle deterministic IRP, Archetti et al. [7] introduced and found exact solutions for *small* benchmark instances of up to 50 clients over 3 days and up to 30 clients over 6 days. Later, Archetti et al. [8] gave a heuristic combining tabu search with MIPs that found near-optimal solutions to the small instances. They also improved upper bounds on *large* instances of 200 clients over 6 days in the case of order-up-to-level policy. For the more general multivehicle case, Desaulniers et al. [37] provide a branch-price-and-cut algorithm, finding 54 new optima out of the 238 instances from Archetti et al. [8] with unknown optima. Archetti et al. [6] give a metaheuristic that solves MIPs both before and after tabu search. To initialize a solution, they formulate MIPs of different strengths and choose the MIP based on the instance size, stronger MIP for smaller instances. Then the tabu search adds, deletes, or moves visits. If the instance was small, the MIP after the tabu search fixes some variables to integer values based on how often the variable is 0 or 1 among the solutions from the search. For large instances, the routes from the tabu solutions are included as route variables in the MIP. They were able to improve the upper bound on 224 of the 240 large instances. We remark that our results are not directly comparable with these existing results since we consider the uncapacitated version of the problem. To the best of our knowledge, there are no computational studies specifically geared towards the uncapacitated IRP considered here. However, as a starting point, we test our heuristics on uncapacitated instances having the same parameter values (except for the capacities that we ignore) as the large instances of size 200 by 6 [8].

Prize-Collecting Steiner Trees and Tours. Since we will use the solvers for PCST in our implementation, we also review previous work on PCST. Specifically, we use

the dual-ascent-based branch-and-bound solver of Leitner et al. [**LLLS18**] for PCST, which reaches near-optimality and performs the fastest on many categories of instances from the DIMACS Challenge. Prior to this, Ljubić et al. [71] and Fischetti et al. [46] provided the most competitive algorithms to find exact solutions for PCST. Whenever we obtain a tree from the solver [**LLLS18**], we convert the tree to a tour by running the TSP solver Concorde [**Cook15**] on the subset of vertices spanned by the tree.

Theoretically, our greedy algorithm will solve a PCST problem to find low density set covers. The solution to each PCST instance will come from the primal dual algorithm for PCST [55], whose analysis is used in eventually proving a logarithmic factor guarantee of our greedy algorithm for IRP.

Approximation Algorithms for IRP. On the theoretical side, approximation algorithms for special cases of IRP have been studied but mostly for the uncapacitated versions that we study here. IRP on general metrics has a $O(\frac{\log T}{\log \log T})$-approximation by Nagarajan and Shi [73] and an $O(\log N)$-approximation by Fukunaga et al [49]. For variants of periodic IRP, Fukunaga et al. [49] provide constant approximations. Another special case of IRP is the joint replenishment problem (JRP). JRP is equivalent to IRP on a two-level tree metric, where the first level has one edge of cost K_0 and the second level consists of children of the first level, with an edge of cost K_i for each commodity i. JRP is known to be NP-hard [9] and APX-hard [75]. On the positive side, Levi et al. [67] give a 2-approximation via a primal dual approach. The approximation factor was reduced to 1.8 in [68] by LP rounding. Bienkowski et al. [17] improve the approximation factor further to 1.791 by randomized rounding.

4.3 Local Search

We examine three local search algorithms: DELETE, ADD, and prioritized. Among them, the latter two solve PCSTs to compute the improvement step. The reason we use PCST in our heuristics is that solving PCSTs allows us to attain locally optimal solutions to IRP. In the ADD local search, an ADD operation on day s will find the subset of clients whose addition to the existing visit set on day s maximizes the total cost savings from the current solution. This best subset is found by solving a PCST problem with appropriately constructed penalties that capture the savings in holding cost for extra visits. The extra connection cost is naturally modeled into the edge costs of the PCST problem. To keep track of the feasible solutions found during the local searches, we use $\mathbf{T}$ to denote candidate solutions to the IRP, overloading it with all the relevant information in the solution such as the trees in the daily visits, the visit times at each demand and the implied holding costs. Since global optimality implies local optimality, if we started with an optimal solution $\mathbf{T}$ and deleted all visits on a fixed day, then applying ADD on that day will yield back the original set visited by $\mathbf{T}$.

We prove this claim formally in Lemma 4.3.1 of Section 4.3.2.

All of the local searches rely on the fact that the test instances generated using Archetti et al.'s model have all positive demands for every $v \in V$, $1 \leq t \leq T$. So all local search heuristics consider modifying the visit set only on days $2, \ldots, T$, not day 1, since any feasible solution must satisfy all the demands on day 1.

To obtain an IRP solution at the end, we convert each day's tree into a tour visiting the clients spanned by the tree by calling Concorde on the graph induced by the spanned clients. We will apply the same conversion from trees to tours in all remaining heuristics as well.

In Section 4.3.1, we introduce a local search procedure that applies only DELETEs. Section 4.3.2 describes a local search procedure applying only ADDs. In Section 4.3.3 we introduce the DELETE-ADD operation and define a more refined local search that applies all three operations in order of complexity.

4.3.1 DELETE

In the delete-only local search, we start with a feasible solution that visits everyone everyday and delete an entire day's visit as long as it makes an improvement.

We define the following notations needed for the algorithm. Let $l(v,s)$ be the latest visit before day s that visits v. Denote by T_s the existing tree on day s in the current step of the algorithm. We use the vector $\mathbf{T}$ to represent the current set of existing trees on each day throughout the time horizon. Let $\hat{t}(v,s)$ be the latest day t such $\mathbf{T}$ does not visit v during the time interval $[s+1,t]$. Notice that if a visit to v on day s is removed, then each demand (v,t) with $t \in [s,\hat{t}(v,s)]$ would incur an extra holding cost of $H^v_{l(v,s),t} - H^v_{s,t}$. So we set the penalty of removing v on day s to be

$$\pi_s(v) := \begin{cases} \sum_{t=s}^{\hat{t}(v,s)} (H^v_{l(v,s),t} - H^v_{s,t}) & \text{if } v \in T_s \\ 0 & \text{else.} \end{cases} \tag{4.1}$$

If the tree T_s on day s is deleted, then the routing cost decreases by $c(E(T_s))$ and the change to the holding cost increases by $\pi_s(V(T_s))$. So the total change in cost for deleting the tree on day s is $\Delta_{DELETE}(s) := -c(E(T_s)) + \pi_s(V(T_s))$. Finally, the operation $DELETE(s)$ removes the existing tree T_s on day s.

Denote by $c(\mathbf{T})$ the total cost of a solution $\mathbf{T}$. The *improvement ratio* of an operation is the magnitude of the change in cost induced by the operation divided by the total cost of the current feasible solution. In all of the local search heuristics, whenever we scan through the time period to find an improving step, we will make the improving operation on the day that gives the best improvement. To avoid potentially long running time, we shall stop looking for improvements whenever the best possible improvement ratio is smaller than some small value, typically 0.01.

Now, we formally define the DELETE algorithm.

Algorithm 1 Local Search with DELETE

1: Initialize a feasible solution $\mathbf{T}$.
2: Let $t' = \arg\min_{s\in[2,T]} \Delta_{DELETE}(s)$.
3: **while** $\frac{|\Delta_{DELETE}(t')|}{|c(\mathbf{T})|} \geq 0.01$ **do**
4: $\quad DELETE(t')$
5: **end while**

The initial feasible solution here is a "full" solution, where each day consists of a minimum cost tree that visits everyone. We delete only trees within time $[2,T]$ to keep the solution feasible. Since all (v,t) have positive demands, any feasible solution must visit everyone on day 1. So leaving T_1 untouched does not make our solution costlier than a local optimum.

4.3.2 ADD

In the add-only local search, we start with a feasible solution and we find an optimal subset of clients to add to the current subset on some day as long as it improves the total cost. To find the best subset of clients to add on a given day s, we

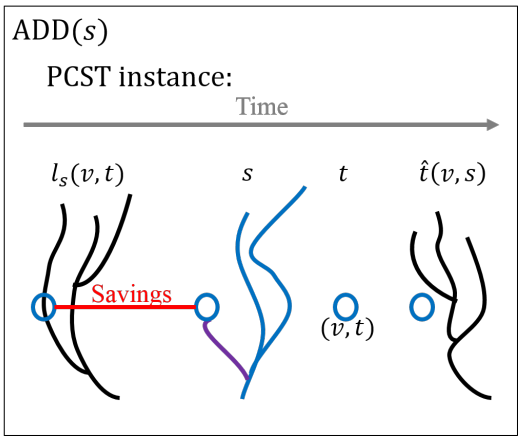


FIGURE 4.3: Here, we consider an ADD operation on day s. If a client v is added to the existing tree on day s (shown in blue), then any demand point at v with deadline day t within s and $\hat{t}(v,s)$ would benefit from the extra visit. The red segment represents the amount of holding cost that would be reduced for (v,t) if v was visited on day s instead of day $l(v,s)$. To reach v from the existing tree, the extra routing required is represented by the purple path connecting the blue tree to v.

solve an appropriately constructed PCST problem whose cost captures the rewards in terms of savings in holding cost when visits are added and the extra routing cost to connect the added visits.

We use the same definitions of $l(v,s)$, T_s, and $\hat{t}(v,s)$ from Section 4.3.1. However, the penalties now apply to the clients *not* visited on the day we are adding visits. In particular, if a visit to v on day s is added, then every demand point (v,t) with $t \in [s, \hat{t}(s,v)]$ *saves* a holding cost of $H^v_{l(v,s),t} - H^v_{s,t}$. So we set the penalties as follows.

$$\pi_s(v) := \begin{cases} \sum_{t=s}^{\hat{t}(v,s)} (H^v_{l(v,s),t} - H^v_{s,t}) & \text{if } v \notin T_s \\ 0 & \text{else.} \end{cases} \tag{4.2}$$

The total change in cost is $\Delta_{ADD}(s) := c(E(PCST_s)) - \pi_s(V(PCST_s))$, where $PCST_s$ denotes an optimal PCST solution on the instance G with penalty function π_s and edge weights defined as follows.

$$w_s(e) := \begin{cases} c(e) & \text{if } e \notin T_s \\ 0 & \text{else.} \end{cases} \tag{4.3}$$

The reason we set edge costs to 0 for the edges in T_s is that the existing tree should be free to use for connecting to the vertices that the PCST adds to the visit set. Notice that minimizing $c(E(PCST_s)) - \pi_s(V(PCST_s))$ is the same as minimizing this quantity after adding a fixed constant value $\pi_s(V)$. After this addition, the total minimization objective becomes $c(E(PCST_s)) + \pi_s(V \setminus V(PCST_s))$. Thus solving the PCST with its original objective function is consistent with minimizing $\Delta_{ADD}(s)$. The operation $ADD(s)$ adds the tree $E(PCST_s)$ to T_s covering the extra clients $V(PCST_s)$. Figure 4.3 shows how the penalty at each client captures the savings in holding cost if the client is added on a specified day.

Note that the neighborhood for the improvement step is of exponential size since each step decides which subset of vertices to add to the current tree for some fixed day. Creating and solving the appropriate PCST instance enables us to find the best

improvement step quickly in practice. Unlike the DELETE step where the PCST instance evaluates good possibilities for deletes (of which there are is at most one tree per day), the ADD step uses the auxiliary PCST instance to carry out the very large neighborhood search efficiently, and represents a key innovation in our algorithm. Algorithm 2 formally describes the ADD algorithm, where the initial feasible solution used is a near-empty solution, which visits everyone only on day 1.

Algorithm 2 Local Search with ADD

1: Initialize a feasible solution $\mathbf{T}$.
2: Let $t' = \arg\min_{s \in [2,T]} \Delta_{ADD}(s)$.
3: **while** $\frac{|\Delta_{ADD}(t')|}{c(\mathbf{T})} \geq 0.01$ **do**
4: $\quad ADD(t')$
5: **end while**

Next, we show that if the visit set of a fixed day was entirely removed from an optimal solution, then we can recover that solution using the local search with ADD.

Lemma 4.3.1. *Let* $\mathbf{T}$ *be an optimal solution to an instance* Π *of IRP. Let* $\mathbf{T}'$ *be the same as* $\mathbf{T}$*, except for removing the tree on day* s *from* $\mathbf{T}$*. Starting with* $\mathbf{T}'$ *as the feasible initial solution, Algorithm 2 will output* $\mathbf{T}$*.*

Proof. First, if $ADD(s)$ is applied to $\mathbf{T}'$, then it gives back $\mathbf{T}$ since there is no better subset to add on day s than T_s by the optimality of $\mathbf{T}$. Second, there cannot be another day s' such that $\Delta_{ADD}(s') < \Delta_{ADD}(s)$, again by the optimality of $\mathbf{T}$. So the first operation applied to $\mathbf{T}'$ in Algorithm 2 is exactly $ADD(s)$, which recovers the optimal solution $\mathbf{T}$. □

4.3.3 Prioritized Local Search

For the prioritized local search, we start with the final solution from the ADD local search of Section 4.3.2. Then we try three operations in order of complexity. First, we try to find a day that DELETE improves the cost on. If no such day exists, we try to find one that ADD improves the cost on. If still no such day exists, we try to find a pair (s_1, s_2) of days such that the net change in cost of $DELETE(s_1)$ followed by $ADD(s_2)$ is negative. As long as any of the three operations makes an improvement, we continue updating the solution. Now we formally define the final pairwise operation $DELETE - ADD(s_1, s_2)$.

In the previous sections, the cost change $\Delta_{DELETE}(s_1)$ and $\Delta_{ADD}(s_2)$ were each computed relative to the existing trees $\mathbf{T}$. Here, $\Delta_{DELETE}(s_1)$ will be defined relative to the existing trees $\mathbf{T}$, but $\Delta_{ADD}(s_2)$ will be defined relative to the leftover trees after deleting T_{s_1} from $\mathbf{T}$ since we want to find the cost of adding to day s_2 right after deleting everything from day s_1. To keep the context of which solution the cost changes are computed on, we denote the new cost changes by $\Delta_{DELETE}(\mathbf{T}, s_1)$ and $\Delta_{ADD}(\mathbf{T} - T_{s_1}\mathbf{e_{s_1}}, s_2)$. Then the change in cost for the pairwise $DELETE - ADD$ is $\Delta_{DA}(s_1, s_2) := \Delta_{DELETE}(\mathbf{T}, s_1) + \Delta_{ADD}(\mathbf{T} - T_{s_1}\mathbf{e_{s_1}}, s_2)$. We now put together all ideas to get a final local search algorithm. The algorithm for prioritized local search is as follows.

We define the greedy and primal dual heuristics in subsequent sections, and we also test prioritized local search starting with the solution from greedy and primal dual, respectively, as the initial feasible solution.

In the prioritized local search, the cost-minimizing pair of days for DELETE-ADD often has s_1 coinciding with s_2, although they are different occasionally.

Algorithm 3 Prioritized Local Search

Initialize the final solution **T** from Local Search with ADD.
while

1. $\exists s \in [2, T]$ such that $\frac{|\Delta_{DELETE}(s)|}{c(\mathbf{T})} \geq 0.01$,
2. or $\exists s \in [2, T]$ such that $\frac{|\Delta_{ADD}(s)|}{c(\mathbf{T})} \geq 0.01$,
3. or $\exists s_1, s_2 \in [2, T]$ such that $\frac{|\Delta_{DA}(s_1, s_2)|}{c(\mathbf{T})} \geq 0.01$

do
  if Condition 1 holds **then**
    Apply $DELETE(\arg\min_{s \in [2,T]} \Delta_{DELETE}(s))$
  end if
  if Condition 2 holds **then**
    Apply $ADD(\arg\min_{s \in [2,T]} \Delta_{ADD}(s))$
  end if
  if Condition 3 holds **then**
    Apply $DELETE - ADD(\arg\min_{s_1, s_2 \in [2,T]} \Delta_{DA}(s_1, s_2))$
  end if
end while

As a heuristic to reduce the run time of prioritized local search, we also test the restricted version of it that only applies DELETE-ADDs to the same day, i.e., $DELETE - ADD(s, s)$.

We will next examine another use of the PCST ideas to design a greedy construction heuristic in the next section.

4.4 Greedy Heuristic

In this section, we introduce a greedy heuristic for IRP. Section 4.4.1 adapts the greedy framework of set cover to IRP, where a minimum density set is repeatedly chosen to cover some subset of demands. The search space for a minimum density set involves an exponential number of subsets of vertices. To simplify the choices needed to pick the set, we instead will show how to find a set whose density at most 3 times the minimum density value in Section 4.4.2. We prove that picking the approximately minimum density as the greedy step achieves a logarithmic approximation factor for IRP. However, this greedy step is still computationally expensive (even though it is in polynomial time). So the implementation will modify the algorithm to repeatedly pick any set whose density is within a certain specified threshold and raise that threshold whenever no more such sets exists. The details of the implementation are described in 4.4.3.

4.4.1 Greedy Framework

The greedy algorithm will attempt to cover the demands with routes choosing a route that minimizes the ratio of the coverage cost to the number of newly covered demands.

As before, T_t denotes the existing tree on day t. Let D be the set of uncovered demands. Let r be the routing cost function, h the holding cost function. For $D' \subset D$,

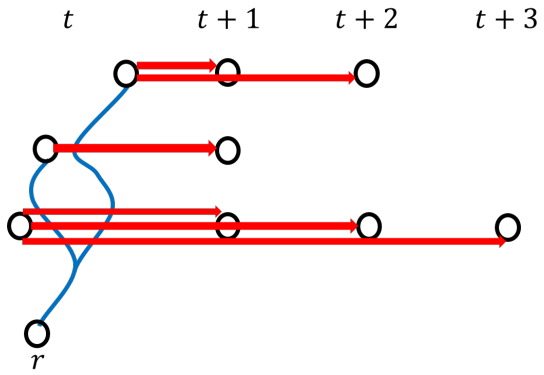


FIGURE 4.4: To apply the greedy algorithm for set cover to IRP, we define a set in IRP to be a subset of demands. The way that a set is served is determined by three choices: a day t of service, a subset of stores to visit on day t which induces a minimum cost tree T spanning the subset, and a subset $D(T)$ of demands with deadlines no earlier than t. The routing cost of this set is cost of the blue tree. The holding cost of this set is the holding cost to serve $D(T)$ from day t, represented by the red segments.

define $d(D') = \sum_{(v,t)\in D'} d_t^v$. The *density* of a tree T and coverage set $D(T)$ of demands is $\rho(T, D(T)) := \frac{r(T)+h(D(T))}{d(D(T))}$.

The greedy algorithm is as follows.

Algorithm 4 Greedy Framework

1: Initialize $T_t \leftarrow \emptyset \forall t \in [1, T]$ and $D \leftarrow D(V \times [T])$.
2: **while** $|D| > 0$ **do**
3: Find a day t, tree T on day t, and coverage set $D(T) \subset D$ minimizing $\rho(T, D(T))$
4: $D \leftarrow D \setminus D(T)$
5: $T_t \leftarrow T_t \cup T$
6: **end while**

Figure 4.4 illustrates how sets of demands are chosen to be covered by visits. Instead of finding the exact minimum density tree and coverage set, we will find those of density at most 3 times the minimum density by solving a PCST whose penalties will represent the best total value of new demands to cover.

4.4.2 Approximate Minimum Density Set

Formally, given a time t that we attempt to add client v to and target density value ρ, define the *coverage number* $\eta(v, t, \rho)$ to be the maximum number of consecutive (with respect to the timeframe) uncovered demand points D' at client v day within days $[t, T]$ such that the weighted average holding cost $\frac{h(D')}{d(D')}$ to serve all such demands is at most ρ. Let $A(v, t, \rho)$ and $h(v, t, \rho)$ be the total demand and total holding cost, respectively, corresponding to the $\eta(v, t, \rho)$ many uncovered demand points whose weighted average holding cost stays within ρ. We use the Algorithm 0 to approximate the minimum density tree and coverage set.

Algorithm 5 Approximately Minimum Density Set

1: Guess the best day t^* to add a minimum density tree and density value ρ^* of the best coverage set.
2: Find the classical primal dual solution [55] to the PCST with edge costs and penalties

$$w(e) := \begin{cases} c(e) & \text{if } e \notin E(T_{t^*}) \\ 0 & \text{else} \end{cases} \tag{4.4}$$

$$\pi(v) := \begin{cases} A(v, t^*, \rho^*)\rho^* & \text{if } v \notin V(T_{t^*}) \\ 0 & \text{else.} \end{cases} \tag{4.5}$$

3: Return the PCST tree $PCST_{t^*}$ and coverage set $\cup_{v \in V(T_{PCST}) \setminus V(T_{t^*})} D_v$ where D_v is the set of $\eta(v, t^*, \rho^*)$ uncovered demands with demand time closest to t^* (starting from t^*).

Next, we show that the above procedure approximates the minimum density set within a factor of 3. For the analysis, we provide the dual LP used to construct the primal dual solution for PCST [55].

$$\min \sum_{e \in E} c_e x_e + \sum_{X \subset V \setminus \{r\}} \pi(X) z_X \tag{4.6}$$

$$\text{s.t.} \sum_{e \in \delta(S)} x_e + \sum_{X : X \supset S} z_X \geq 1 \qquad \forall S \subset V \setminus \{r\} \tag{4.7}$$

$$x_e \geq 0 \qquad \forall e \in E \tag{4.8}$$

$$z_X \geq 0 \qquad \forall X \subset V \setminus \{r\} \tag{4.9}$$

$$\max \sum_{S \subset V \setminus \{r\}} y_S$$

$$\text{s.t.} \sum_{S : e \in \delta(S), S \not\ni r} y_S \leq w_e \qquad \forall e \in E \tag{4.10}$$

$$\sum_{S : S \subset X} y_S \leq \pi(X) \qquad \forall X \subset V \setminus \{r\} \tag{4.11}$$

$$y_S \geq 0 \qquad \forall S \subset V \setminus \{r\} \tag{4.12}$$

We can bound the routing cost with respect to the dual values using the same analysis as [55], except that we bound the cost with respect to $\sum_{S \subset V(T)} y_S$ instead of their $\sum_{S \subset V \setminus \{r\}} y_S$. Let y denote the dual solution defined by the algorithm in selecting the tree T. The following lemma is implicit in [55].

Lemma 4.4.1. $r(T) \leq 2 \sum_{S \subset V(T)} y_S$.

Lemma 4.4.2. *Let T and $D(T)$ be the tree and coverage set returned by the above PCST algorithm. Then $\rho(T, D(T)) \leq 3\rho^*$.*

Proof. First, we compute the routing cost of T. We have

$$\begin{aligned} r(T) \leq & 2 \sum_{S \subset V(T)} y_S \text{ by Lemma 4.4.1} \\ \leq & 2\pi(V(T)) \text{ by the dual constraints} \\ \leq & 2 \sum_{v \in V(T)} A(v, t^*, \rho^*)\rho^*. \end{aligned}$$

Second, the holding cost of $D(T)$ is

$$\begin{aligned} h(D(T)) &\leq \sum_{v \in V(T)} \sum_{(v,t) \in D_v} H^v_{t^*,t} \\ &\leq \sum_{v \in V(T)} A(v, t^*, \rho^*)\rho^*. \end{aligned}$$

We know $d(D(T)) = \sum_{v \in V(T)} A(v, t^*, \rho^*)$. So $\rho(T, D(T)) = \frac{r(T)+h(D(T))}{d(D(T))} \leq 3\rho^*$. □

Finally, we show that if all demand values are at least 1, then the greedy algorithm which uses a 3-approximate minimum density tree and coverage set each round still attains a logarithmic approximation for IRP. The derivation is a simple modification of the set cover analysis.

Theorem 4.4.3. *If $d^v_t \geq 1 \forall (v,t) \in D$, then iteratively picking a 3-approximate minimum density tree and coverage set yields an $O(\log(d(D)))$-approximation for IRP.*

Proof. In each iteration, a set of density at most 3 times the minimum density was found. For each demand point, define its price to be the density of the set that covered it. Label the demand points in order of coverage from (v_1, t_1) to $(v_{|D|}, t_{|D|})$. Then the kth demand point covered has price at most $\frac{3OPT}{d(D \setminus \{(v_1,t_1),\ldots,(v_{k-1},t_{k-1})\})}$. So the final cost is at most $\sum_{k=1}^{|D|} \frac{3OPT}{d(D \setminus \{(v_1,t_1),\ldots,(v_{k-1},t_{k-1})\})} \leq 3H_{d(D)}OPT = O(\log(d(D)))OPT$, where the first inequality follows from $d^v_t \geq 1 \forall (v,t) \in D$. □

4.4.3 Implementation Detail

While the aforementioned greedy algorithm attains a provable bound on the cost, it is impractical to run on large instances. Just finding one coverage set per round involves a binary search for the best density value $\rho*$ that will induce an approximately minimum density set. On instances of 30 clients over 60 days, the version of greedy that searches for the approximately lowest density set took over an hour per instance. Another technique we tried to reduce the running time was to search for a single value of the lowest density such that there is a feasible set cover, instead of a different low density per round. However, the single density version of greedy still took over 45 minutes per 30 by 60 sized instance. Finally, we implemented a modification of this version that further limits the search space. First, given the existing trees T_t per day t and the uncovered set D, define the procedure $COVER(\rho)$ as in the following algorithm.

If ρ is too low, it is possible that $COVER(\rho)$ does not satisfy all demands. So whenever $COVER(\rho)$ stops serving new demands, we will relax the target density ρ by multiplying it a factor $\alpha > 1$ to continue serving demands until all are satisfied. Formally, given a relaxation factor $\alpha > 1$, we implement a heuristic called $GREEDY(\alpha)$ defined in Algorithm 7.

Algorithm 6 $COVER(\rho)$

while there is a set of density $\leq \rho$ **do**
    Find a day t, tree T on day t, and coverage set $D(T) \subset D$ with lowest density (minimized over $t \in T$), with PCST penalties induced by ρ and t.
    $D \leftarrow D \setminus D(T)$.
    $T_t \leftarrow T_t \cup T$.
end while

Algorithm 7 $GREEDY(\alpha)$

Initialize $T_t \leftarrow \emptyset \forall t \in [1, T]$.
Find lowest value $\rho_{\min}$ such that $COVER(\rho_{\min})$ serves a nonempty set of demands and apply $COVER(\rho_{\min})$.
while $|D| > 0$ **do**
    $\rho_{\min} \leftarrow \alpha * \rho_{\min}$.
    Apply $COVER(\rho_{\min})$.
end while

To estimate the correct value for $\rho_{\min}$, we start with a small initial value for $\rho_{\min}$ and double it until $COVER(\rho_{\min})$ returns a nonempty set.

Since PCST already has fast near-optimal solvers, our implementation also differs from the stated algorithm by finding using the solver of Leitner et al. [**LLLS18**] to solve PCSTs rather than the primal dual algorithm of Goemans and Williamson [55].

Besides the pure Greedy heuristic, we also test how well Prioritized local search does if it is initialized with the solution from Greedy instead of from local search with ADDs. We refer to the combination of Greedy with Prioritized local search as *Pruned Greedy*.

In the next section, we show a third application of the PCST ideas to design a primal-dual based heuristic for IRP.

4.5 Primal Dual

In this section, we investigate a primal-dual approach similar to [67] for solving it. Inspired by the *waveform mechanism* introduced in [67] which was used for solving JRP, we generalize this idea and try to make it applicable for solving IRP. We will solve PCST instances where each vertex of the input represents a demand point of the IRP instance.

Section 4.5.1 states the primal and dual LP relaxations for IRP. Using the LPs, the primal dual algorithm is presented in Section 4.5.2. For simplicity of the algorithm, not all of the dual values are defined explicitly in the algorithm. In Section 4.5.3, we prove that there is always a feasible setting of dual values corresponding to the growth of moats in the algorithm. Finally, Section 4.5.4 discusses a more efficient way that the primal dual algorithm can be implemented.

4.5.1 LP Formulation

To simplify the notation, we assume that each client v has a unique day t such that $d_t^v > 0$, otherwise we may add cost 0 edges to relabel multiple demand points at the same vertex as different vertices. Given a client v, the day t for which $d_t^v > 0$ is denoted by $t(v)$. For convenience, we use v to represent the demand point

$(v, t(v))$. The variable y_s^e indicates whether edge e is used on day s. The variable $x_{s,t(v)}^v$ indicates whether demand $(v, t(v))$ is served on day s.

First, we state primal linear program and its dual:

$$\min \sum_{e\in E}\sum_{s=1}^{T} w_e y_s^e + \sum_{v\in D}\sum_{s=1}^{t(v)} H_{s,t(v)}^v x_{s,t(v)}^v$$

$$\text{s.t.} \sum_{s=1}^{t(v)} x_{s,t(v)}^v \geq 1 \qquad \forall (v,t(v)) \in D \tag{4.13}$$

$$\sum_{e\in\delta(X)} y_s^e \geq x_{s,t(v)}^v \qquad \forall (v,t(v)) \in D, 1 \leq s \leq t(v), X \subset V : X \ni v, X \not\ni r \tag{4.14}$$

$$x_{s,t(v)}^v \geq 0 \qquad \forall (v,t(v)) \in D, 1 \leq s \leq v(t) \tag{4.15}$$

$$y_s^e \geq 0 \qquad \forall e \in E, 1 \leq s \leq T \tag{4.16}$$

$$\max \sum_{(v,t(v))\in D} b_{t(v)}^v$$

$$\text{s.t. } b_{t(v)}^v \leq H_{s,t(v)}^v + \sum_{X\ni v, X\not\ni r} \beta_{s,t(v)}^{v,X} \qquad \forall (v,t(v)) \in D, 1 \leq s \leq t(v) \tag{4.17}$$

$$\sum_{(v,t(v))\in D, X\ni v, X\not\ni r, \delta(X)\ni e} \beta_{s,t(v)}^{v,X} \leq w_e \quad \forall e \in E, 1 \leq s \leq T \tag{4.18}$$

$$b_{t(v)}^v \geq 0 \qquad \forall (v,t(v)) \in D \tag{4.19}$$

$$\beta_{s,t(v)}^{v,X} \geq 0 \qquad \forall (v,t(v)) \in D, X \subset V, 1 \leq s \leq T \tag{4.20}$$

In the primal LP, constraint 4.13 ensures that every demand point is served on time. Constraint 4.14 ensures that whenever a client v is served on day s, there is a path from the depot to v on day s.

4.5.2 A Primal-dual Approach

For the dual LP, variable $b_{t(v)}^v$ represents the budget amount that demand point $(v, t(v))$ has available to pay for visits to serve it. Variable $\beta_{s,t(v)}^{v,X}$ represents the amount that $(v, t(v))$ contributes towards building a tree crossing X to get served on day s. However, since the $\beta_{s,t(v)}^{v,X}$ variables are not part of objective function in the dual LP, we cannot directly use $\beta_{s,t(v)}^{v,X}$ to pay for visits. Instead, $\beta_{s,t(v)}^{v,X}$ represent copies of the total budget $b_{t(v)}^v$, one copy for each s. The general framework is to raise the budgets of demands as long as all constraints in the dual LP are able to hold. The final values of the budgets are determined by the tightening of dual constraints that they are involved in.

First, we describe the intuition of the algorithm. At the beginning of the algorithm, all budgets $b_{t(v)}^v$ and visit-specific payments $\beta_{s,t(v)}^{v,X}$ are to start at 0. We introduce a continuous parameter τ that slides through time from T to 1 at a constant rate. The position of τ within the time horizon will determine what value to raise the budgets and visit-specific payments. Whenever τ passes through an integral time t (i.e. $\tau < t$), it "wakes up" the budgets $b_{t(v)}^v$ of demands $(v, t(v))$ occurring on day $t(v) = t$. Those $b_{t(v)}^v$ shall increase at the same rate that $H_{\tau,t(v)}^v$ is

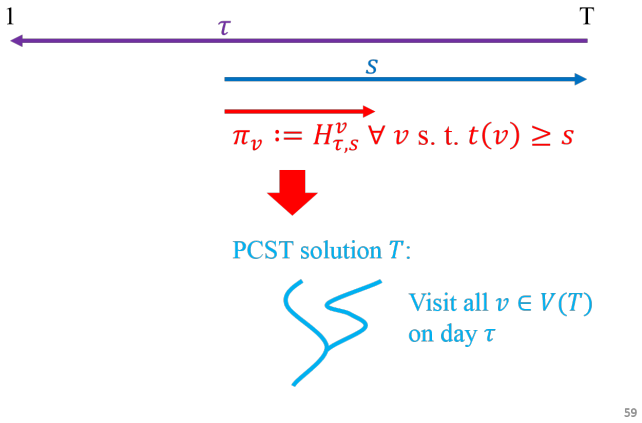


FIGURE 4.5: At each value of τ and s, we define a PCST instance whose penalty at each client is the holding cost to store the product there from day τ to day s. A solution to the PCST instance determines the subset of clients to visit on day τ. After this procedure is repeated for every value of τ and s, we know exactly which clients are visited that day.

increasing as τ is sliding towards 1, i.e., we keep $b^v_{t(v)}$ at exactly the same value as $H^v_{\tau,t(v)}$. The definition of $H^v_{\tau,t}$ for non-integral τ is interpolated linearly, i.e., define $H^v_{\tau,t} = (1 - \tau + \lfloor\tau\rfloor)H^v_{\lfloor\tau\rfloor,t} + (\tau - \lfloor\tau\rfloor)H^v_{\lceil\tau\rceil,t}$.

Observe that keeping $b^v_{t(v)} = H^v_{\tau,t(v)}$ ensures that each demand $(v, t(v))$ can at least pay for the holding cost from time τ to $t(v)$. To maintain feasibility to the dual constraints, we also raise $\beta^{v,X}_{s,t(v)}$ as needed to keep constraint 4.17 satisfied. That means for each demand $(v, t(v))$ and each $s \in (\tau, t(v)]$, we raise the value of $\sum_{X \ni v, X \not\ni r} \beta^{v,X}_{s,t(v)}$ to at least $H^v_{\tau,t(v)} - H^v_{s,t(v)}$. For each value of τ and s, we create a PCST instance whose penalty at v is assigned to $H^v_{\tau,t(v)} - H^v_{s,t(v)}$ and solve it using the primal dual algorithm of Goemans and Williamson [55]. The value to raise each $\beta^{v,X}_{s,t(v)}$ will be determined by the dual values of the PCST instance set by primal dual algorithm [55]. We defer the details of the exact values to set them to the proof of feasibility for Theorem 4.5.1.

Next, we give the necessary definitions to state the algorithm formally. Initially, all the dual variables are *unfrozen*. During the running of the algorithm, we set the value of the dual variables as τ goes to 1. By *freezing* a dual variable we mean that the value of that particular variable will not change from then on. A vertex $v \in D$ is a *frozen vertex* if and only if $b^v_{t(v)}$ is frozen. In the algorithm, we shall serve a vertex whenever it becomes frozen. Let $\mathcal{F}$ denote the set of the all frozen vertices since the beginning of the algorithm until the current moment, i.e. since when $\tau = T$ till when $\tau = t$ where t is the current location of the sweep line.

The algorithm assigns a service time $l(v)$ to each frozen vertex v; the details of the assignment to v will be explained later. This assignment would be in such a way that: $1 \leq l(v) \leq t(v) \leq T$, and for any $v \in \mathcal{F}$, we have $b^v_{t(v)} = H^v_{l(v),t(v)}$.

Finally, define the set of *active vertices at time* s to be $\mathcal{A}(s) = \{v : v \in D, s \leq t(v)\}$ for all $s \in [1, T]$.

Now, we are ready to give the algorithm formally in Algorithm 8. For the sake of intuition, we give a continuous description of the algorithm which can be easily modified to be a discrete and polynomial time algorithm. Figure 4.5 provides a visualization of the algorithm.

Algorithm 8 Primal Dual

Initialize $\mathcal{F} \leftarrow \emptyset$ and $\forall 1 \leq s \leq T$, $\mathcal{A}(s) \leftarrow \{v : v \in D, s \leq t(v)\}$.
for $\tau \leftarrow T$ towards 1 **do**
  for $s \leftarrow \lceil \tau \rceil$ to T **do**
    Make an instance of the prize-collecting Steiner tree problem by assigning a penalty π_v to each vertex $v \in \mathcal{A}(s)$ as follows
    for all $v \in \mathcal{A}(s)$ **do**
      if $v \notin \mathcal{F}$ **then**
        $\pi_v = H^v_{\tau,t(v)} - H^v_{s,t(v)}$
      else
        $\pi_v = 0$
      end if
    end for
    Solve the prize-collecting Steiner tree instance using the classical primal dual algorithm [55] and let X be the subset of $\mathcal{A}(s)$ getting connected to the root r in the solution
    if $X \not\subset \mathcal{F}$ **then**
      For all $v \in X \backslash \mathcal{F}$ let $l(v) = \tau$ and $b^v_{t(v)} = H^v_{l(v),t(v)}$, and visit v at time $\lceil l(v) \rceil$ (the values to set $\beta^{v,X}_{s,t(v)}$ will be provided in the proof of Theorem 4.5.1)
      $\mathcal{F} \leftarrow \mathcal{F} \cup X$
      Freeze the unfrozen vertices in X
    end if
  end for
end for
For all $v \notin \mathcal{F}$ let $b^v_{t(v)} = H^v_{1,t(v)}$ and visit $V \setminus \mathcal{F}$ on day 1.
Output the IRP schedule specified by the service times for each demand point.
Output the dual variables $b^v_{t(v)}$.

Observe that at the end, all clients will have been frozen and served at some point.

4.5.3 Defining a Feasible Dual

Next, we show that there is a feasible dual solution $\boldsymbol{b}, \boldsymbol{\beta}$ satisfying the assignment of values for b_t^v from the algorithm. For the analysis, we shall refer to an particular iteration in the algorithm by the value of τ and s at that point.

Theorem 4.5.1. *During any moment (τ, s) of the algorithm, for the setting $b^v_{t(v)} = H^v_{\tau,t(v)}$, there is an assignment of $\boldsymbol{\beta}$ so that $\boldsymbol{b}, \boldsymbol{\beta}$ is feasible to the dual.*

Proof. Assume that we are in iteration (τ, s) of the algorithm. Let y_S be the values of the dual variables corresponding to the primal dual solution for PCST in this iteration. Note that y_S depends on (τ, s), but we omit further subscripting by (τ, s) for simplicity of notation. We will distribute the dual value y_S among the client-specific dual variables $\beta^{v,S}_{s,t(v)}$ with the goal of satisfying constraint 4.17.

Define the *potential* of client v to be $p(v) := \pi_v - \sum_{S \ni v, S \not\ni r} \beta^{v,S}_{s,t(v)}$. Initialize $\boldsymbol{\beta} = 0$. As y_S grows, assign $\beta^{v,S}_{s,t(v)} = \frac{y_S}{|\{v \in S : p(v) > 0\}|}$. Next, we show that this setting of $\boldsymbol{\beta}$ along with the setting $b^v_{t(v)} = H^v_{\tau,t(v)}$ of the algorithm constitutes a feasible dual solution to IRP.

First, we can easily verify constraint 4.18. For a given $e \in E, s \leq T$, we have

$$\begin{aligned}\sum_{v \in V, X \ni v, X \not\ni r, \delta(X) \ni e} \beta^{v,X}_{s,t(v)} \leq & \sum_{X : \delta(X) \ni e, X \not\ni r} \sum_{v \in X} \beta^{v,X}_{s,t(v)} \\ \leq & \sum_{X : \delta(X) \ni e, X \not\ni r} y_X \text{ by definition of } \boldsymbol{\beta} \\ \leq & w_e \text{ by the dual constraints for PCST.}\end{aligned}$$

Second, we show that $\sum_{X \ni v, X \not\ni r} \beta^{v,X}_{s,t(v)} \geq H^v_{\tau,t(v)} - H^v_{s,t(v)}$ for all $v \in V$ and $s \leq t(v)$, which would imply constraint 4.17. Fix v and s. Consider the moment just before $b^v_{t(v)}$ froze, which means the previous PCST solution did not span v. By the primal dual algorithm of Goemans and Williamson, v was in some set X such that $\pi(X) = \sum_{S : S \subset X} y_S$. Then

$$\begin{aligned}\pi(X) = & \sum_{S : S \subset X} y_S \\ = & \sum_{S \in X} \sum_{v \in S} \beta^{v,S}_{s,t(v)} \\ \leq & \sum_{S \not\ni r} \sum_{v \in S \cap X} \beta^{v,S}_{s,t(v)} \\ = & \sum_{v \in X} \sum_{S \ni v, S \not\ni r} \beta^{v,S}_{s,t(v)} \\ \leq & \sum_{v \in X} \pi_v \text{ since only those } v \text{ whose potential are positive grow their } \beta^{v,S}_{s,t(v)} \\ = & \pi(X).\end{aligned}$$

So all inequalities must be equalities, which means that $\sum_{X \ni v, X \not\ni r} \beta^{v,X}_{s,t(v)} = \pi_v = H^v_{\tau,t(v)} - H^v_{s,t(v)}$. Hence constraint 4.18 holds.

□

4.5.4 Implementation

Here, we provide a simpler implementation of Algorithm 8, which does not require setting dual values and eliminates the loop over s from $\leftarrow \lceil \tau \rceil$ to T. In Algorithm 8, the purpose of the loop over s is to help determine feasible dual values to set for the variables $\beta^{v,X}_{s,t(v)}$ to prove Theorem 4.5.1. However, for purposes of obtaining the same primal solution, we do not need to create and solve the PCST instance per s value. For a fixed τ, no matter which value s takes, the vertices spanned by the PCST solution all become assigned to the service day τ. Also, for each demand day $t(v)$, there is some round when s takes value $t(v)$, so that the penalty assigned to v is at its highest possible value $H^v_{\tau,t(v)}$. If v gets assigned to be served on day τ by any s, it would certainly be part of the PCST solution to the instance having the highest penalty $H^v_{\tau,t(v)} - H^v_{t(v),t(v)}$. So instead of collecting the visits on day τ separately through different values of s, we could solve one PCST instance to determine the visit set for day τ by setting penalty $H^v_{\tau,t(v)}$ for v to collect all visits that could possibly have been induced by the largest s. Similarly, the raising of dual values $b^v_{t(v)}$ in Algorithm 8 was included to help prove Theorem 4.5.1 and is not needed to determine the primal solution. One last detail we modify is the solution method for PCSTs. Algorithm 8 solved PCSTs using [55] so that the dual values of PCST from [55] could be used to determine the dual values to set $\beta^{v,X}_{s,t(v)}$, again to prove feasibility. However, for faster solving time, we solve PCSTs using **[LLLS18]** instead since we only need to recover the primal solution at the end regardless of the dual values. Further, our implementation is a simplification of the original algorithm that discretizes τ to take only integer values from T to 1. This allows us to use the fast PCST solver of Leitner et al. **[LLLS18]** in a self-contained manner rather than having the breakpoints of τ depend on the dual solution for PCST. However, as noted above, the simplification only finds a primal solution for IRP. The dual values are no longer valid after restricting the breakpoints of τ to only integers. Algorithm 9 describes the aforementioned heuristic exactly as implemented.

In addition to the pure Primal Dual heuristic, we test Prioritized local search initialized with the solution from the Primal Dual heuristic, which we call *Pruned Primal Dual*.

4.6 Benchmark MIP Formulation

In this section, we describe a compact MIP formulation of the IRP, that we use with modern solvers to establish the benchmark for comparing our solutions. Our exact MIP formulation for IRP is of size $O(N^2T) + O(NT^2)$. When the problem instances get larger, we are however only able to generate lower bounds for the value of a solution even using state-of-the-art solvers such as Gurobi Version 7.

As before, $x^v_{s,t}$ will be the variable indicating whether to serve (v,t) on day s. Define a related variable X^v_s indicating whether v is visited on day s. Let z^{uw}_s be the variable indicating whether to use an arc uw on day s. Let h^{uw}_s be the continuous variable representing the amount of total flow through arc uw on day s coming from the depot.

Intuitively, the purpose of h^{uw}_s is to enable expressing connectivity in a polynomial number of constraints, in contrast with using a non-compact set of exponentially many cut-covering constraints. In Figure 4.6, we provide an example of how the values of h^{uw}_s are set in a feasible solution. Then IRP is modeled by the following MIP.

Algorithm 9 Primal Only Implementation

Initialize $\mathcal{F} \leftarrow \emptyset$ and $\forall 1 \leq s \leq T$, $\mathcal{A}(s) \leftarrow \{v : v \in D, s \leq t(v)\}$.
for $\tau \leftarrow T$ to 1 **do**
    Make an instance of the prize-collecting Steiner tree problem by assigning a penalty π_v to each vertex $v \in \mathcal{A}(\tau)$ as follows
    for all $v \in \mathcal{A}(\tau)$ **do**
        if $v \notin \mathcal{F}$ **then**
            $\pi_v = H^v_{\tau,t(v)}$
        else
            $\pi_v = 0$
        end if
    end for
    Solve the prize-collecting Steiner tree instance using the solver [**LLLS18**] and let X be the subset of $\mathcal{A}(\tau)$ getting connected to the root r in the solution
    if $X \not\subset \mathcal{F}$ **then**
        For all $v \in X \backslash \mathcal{F}$, visit v at time τ
        $\mathcal{F} \leftarrow \mathcal{F} \cup X$
        Freeze the unfrozen vertices in X
    end if
end for
For all $v \notin \mathcal{F}$, visit $V \setminus \mathcal{F}$ on day 1.
Output the IRP schedule specified by the service times for each demand point.

$$\min \sum_{u \in V} \sum_{w \neq u \in V} \sum_{s=1}^{T} c_{uw} z_s^{uw} + \sum_{(v,t) \in D} \sum_{s=1}^{t} H^v_{s,t} x^v_{s,t}$$

$$\text{s.t. } z_s^{uw} + z_s^{wu} \leq 1 \qquad \forall u \in V, w > u, s \leq T \tag{4.21}$$

$$\sum_{s=1}^{t} x^v_{s,t} = 1 \qquad \forall v \in V, t \leq T \tag{4.22}$$

$$X^v_s \geq x^v_{s,t} \qquad \forall v \in V, t \leq T, s \leq t \tag{4.23}$$

$$\sum_{w \neq v} z_s^{vw} = X^v_s \qquad \forall v \in V \setminus \{r\}, s \leq T \tag{4.24}$$

$$\sum_{w \neq v} z_s^{wv} = X^v_s \qquad \forall v \in V \setminus \{r\}, s \leq T \tag{4.25}$$

$$\sum_{w \neq r} z_s^{rw} \leq 1 \qquad \forall s \leq T \tag{4.26}$$

$$\sum_{w \neq r} z_s^{wr} \leq 1 \qquad \forall s \leq T \tag{4.27}$$

$$\sum_{w \neq u} h_s^{wu} - \sum_{w \neq u} h_s^{uw} = \begin{cases} X^u_s, u \neq r \\ \sum_{a \neq r} -X^a_s, u = r \end{cases} \qquad \forall u \in V, s \leq T \tag{4.28}$$

$$h_s^{uw} \leq (N-1) z_s^{uw} \qquad \forall u \in V, w \neq u, s \leq T \tag{4.29}$$

$$X^v_s, x^v_{s,t}, z_s^{uw} \in \{0,1\} \qquad \forall v \in V, u \in V, w \neq u \in V, t \leq T, s \leq T \tag{4.30}$$

$$h_s^{uw} \geq 0 \qquad \forall u \in V, w \neq u, s \leq T \tag{4.31}$$

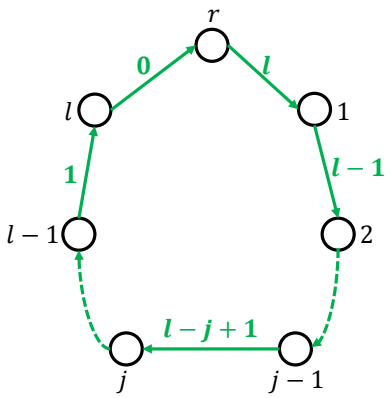


FIGURE 4.6: For a fixed day s, suppose that nodes $1, \ldots, l$ are visited by a cycle in a feasible solution to IRP. To determine the appropriate values to set h_s^{uw} variables, note that each visited node contributes one unit of flow along the path from from r to itself. Then the flow through an arc uw would be the total number of all the paths between r and visited nodes that have uw in the path. The labels along the arcs indicate the values that h_s^{uw} would take per arc uw. Values of the remaining variables would be set in the obvious ways: $z_s^{uw} = 1$ if and only if arc uw is in the cycle, $X_s^v = 1$ if and only if $v \in \{1, \ldots, l\}$, $x_{st}^v = 1$ if and only if day s is the latest day before or on day t having a visit to v.

Constraint 4.21 ensures that each edge is used at most once. Constraint 4.22 guarantees that all demands are satisfied on time. Constraint 4.23 ensures that whenever a demand is served on a specified day, there must be a visit to the client on that day. Constraints 4.24 and 4.25 guarantee that if a vertex is visited, then some in-arc and some out-arc incident to it must be traversed. Constraints 4.26 and 4.27 limit the number of cycles to 1.

We needed a separate case for the fractional degree at r because the depot could be served by itself on day s while not building any arcs on day s, which means that $\sum_{w \neq r} z_s^{rw}$ and $\sum_{w \neq r} z_s^{wr}$ could potentially be 0 even when $X_s^r = 1$. Constraint 4.28 ensures that the net in-flow into any $u \neq r$ corresponds to whether u is visited on that day, and the net in-flow into r corresponds to the negative of the number of vertices visited (i.e., out-flow of one per node). Constraint 4.29 requires that on each day, an arc must be built if there is flow through it from the depot, and the flow allowed is bounded by the maximum possible number of visited nodes.

Solving this MIP directly within MIPGap of 10% was not practical past instances of size 140 (nodes) by 6 (days) and 50 by 14. We use the lower bound found at 10% MIPGap to compare with the costs from our heuristics.

4.7 Computational Results

Before summarizing the results, we define the necessary parameters involved in the tests instances. To remain consistent with the problem notation, we still use N for the number of clients and T for the number of days. A new parameter H is the amount that the holding cost part of the objective function is multiplied to generate instances with different trade-offs between routing cost and holding cost. Thus a larger H is oriented towards a solution that pays more attention in optimizing holding versus the routing costs. To measure the quality of our solutions, we examine

the ratio between the cost of a heuristic solution and the lower bound on the optimal cost found by Gurobi solving the MIP. We call this ratio the *gap* between the respective costs. We implemented all the heuristics we described earlier, to compare them against the lower bounds from the benchmark MIP model described above.

1. Local search with DELETE (Algorithm 1).
2. Local search with ADD (Algorithm 2).
3. Prioritized Local Search (Section 4.3.3).
4. Greedy (Section 4.4.3).
5. Pruned Greedy - this is the prioritized local search algorithm applied to an initial solution obtained by Greedy.
6. Primal Dual (Section 4.5.4).
7. Pruned Primal Dual - this is the prioritized local search algorithm applied to an initial solution obtained by Primal Dual.

Here is a summary of the performance of the various methods we implemented (details of the data generation model are in the next section).

1. The heuristics achieving the best gap and solving time are the prioritized local search and the local search with ADD steps, shown in Figure 4.7.
2. Greedy has the largest gaps among all heuristics according to Figures 4.7, 4.8, and 4.9 because picking low density sets to cover demands does not directly help lower the final cost of the IRP solution. Greedy also incurs the longest solving time due the computationally expensive nature of picking low density sets.
3. The prioritized local search starting with greedy or primal dual solutions had worse gaps than simply starting with the solution after local search with ADDs according to Figures 4.7, 4.8, and 4.9. The worse performance could be caused by greedy and primal dual being closer to locally optimal but still somewhat far from the lower bound.
4. From Figure 4.7, as H increases, greedy's gap decreases dramatically because low H values incentivize greedy to pick large sets to start with, which leaves no room to cover demands optimally. The larger H becomes, the more opportunities arise for the sets to be more fine tuned. Primal dual and the three local search heuristics have gaps that initially increase up to some H value within $[2, 3]$, then decrease thereafter. This pattern occurs because mid-range H values create instances with more balanced trade-offs between routing cost and holding cost, which correspond to higher difficulty to solve.
5. As T increases, primal dual's gap worsens according to Figure 4.9 because primal dual tends to build denser visits than needed because the backwards sweep construction of visit sets cannot account for potentially early visits serving late deadlines.

The rest of this section explains these points in more detail.

The heuristics were implemented in C++ on an Intel Xeon processor X5680, 3.33 GHz machine with 8 GB RAM. A copy of the code and data used to conduct the

experiments will soon be available on github. Since solving the MIP was the major bottleneck in completing the experiments, we allowed multiple threads for the MIP to speed up the solving time. The MIP was solved by Gurobi Version 7 on default settings using 8 threads. Each heuristic uses 1 thread. We report the run times directly without accounting for number of threads. Next, we describe how the test instances were generated. Recall that we defined the actual versions of the heuristics implemented in their corresponding sections. We then show their performance in cost and runtime relative to each of the three varying parameters.

4.7.1 Data Generation Model

We follow the same generation model of the largest instances tested in Archetti et al. [8], except that our model has no capacity constraints at vehicles and store locations. To find how the heuristics perform on different types of instances, we choose three parameters N, T, and H to vary (one at at time). For each setting of parameter values, we generate 100 test instances.

- Time horizon $T = 6, 8, \ldots, 18$
- Number of clients $N = 110, 120, \ldots, 160$
- Demands d_t^v randomly generated as an integer in the interval $[10, 100]$
- Unit holding cost h_v randomly generated real number in the interval $[0.01, 0.05]$
- Transportation cost $c_{uv} = \sqrt{(X_u - X_v)^2 + (Y_u - Y_v)^2}$, with each coordinate of (X_u, Y_u) randomly generated as an integer in $[0, 500]$

We have an additional parameter $H = .01, 0.51, \ldots, 6.01$ which we use as the scaling factor for the holding cost to generate a wide variety of instances. Our objective function relative to H is $r(S) + H \times h(S)$, where $r(S)$ is the routing cost and $h(S)$ is the holding cost of a solution S.

4.7.2 Performance Evaluation

For each heuristic, we plot two values as H, N, or T varies: the gap and running time. Recall that the gap is the ratio between the heuristic's solution cost and the lower bound found for the optimal cost of the MIP model. Henceforth, all gap or running time values mentioned refer to the **average values** over the 100 instances per choice of parameter values.

Varying H

Here, N and T are fixed to 100 and 6, respectively. The holding cost scale H varies from 0.01 to 6.01 at increments of 0.5. Results that require lower bound from the MIP go up to only $H = 4.51$ due to high running times of the MIP and the large number of instances per parameter value. We compare the performance of the three local search heuristics, pure greedy, greedy with pruning, pure primal dual, and primal dual with pruning.

First, we state the results for the gap.

- Greedy has decreasing gaps as H increases, starting at 2.33 when $H = 0.01$ to 1.53 when $H = 4.51$.

- Pruned greedy drastically improves upon greedy due to the additional pruning by prioritized local search and has the largest gap of 1.11 at $H = 3.01$. As mentioned before, the mid-range values of H correspond to harder instances because the trade-off between the routing cost and holding cost is in the middle.
- Primal dual has its largest gap of 1.19 at $H = 2.51$.
- Pruned primal dual's gaps increase as H increases, starting from 1.01 at $H = 0.01$ to 1.11 at $H = 4.51$. It might be the case that the gap would eventually taper off at a larger value of H, but we do not have the lower bounds to compare with due to the large solving times of the MIP.
- For DELETE local search, the gap is largest at 1.1 for H at 2.51 and 3.01.
- For ADD and prioritized local searches, the gap is at most 1.05 for all values of H. The values of H for which they have their highest gaps occur at 2.01 and 2.51. Since ADD and DELETE-ADD are richer operations in their use of the PCST solutions, we expect them to reach close to optimality. The prioritized local search and local search with ADDs have similar gaps because the solution from ADD is already nearly optimal and there is little room to improve the cost.

In summary, the heuristics in order of lowest to highest gaps are local search with ADDs, prioritized local search, local search with DELETEs, pruned greedy, pruned primal dual, primal dual, and greedy.

Next, we examine the running time of each heuristic.

- The MIP requires over 47 seconds to solve for $H = 0.01$ and over 768 seconds at $H = 4.51$.
- Greedy and pruned greedy take the longest to solve among the heuristics. Their solving times start at more than 19 seconds when $H = 0.01$ and increase to over 24 seconds at $H = 6.01$.
- Pruned primal dual requires no more than 3 seconds for all values of H.
- Each of the three local searches (prioritized, ADD, DELETE) take less than 1 second on average for instances with any value of H.
- Primal dual also takes under 1 second throughout all instances.

To summarize, the heuristics in order running times from the fastest to slowest are local search with DELETE-ADDs, primal dual, local search with ADDs, Prioritized local search, pruned primal dual, greedy and pruned greedy.

Local search with ADDs prevails as the strongest of the heuristics with a gap of at most 1.05 throughout and runtime of less than 1 second at $H = 4.51$, which is nearly three orders of magnitude faster than the MIP at $H = 4.51$.

Varying N

Here, T and H are fixed to 6 and 2.6, respectively. The number of clients N varies from 110 to 160 at increments of 10. Due to limited computation time, results which require MIP values are restricted to $N \leq 140$.

First, we provide the gaps for each heuristic. The gap values of the heuristics did not form any particular patterns with respect to N.

- Among all methods, greedy has the highest gap of 1.57 to 1.58.

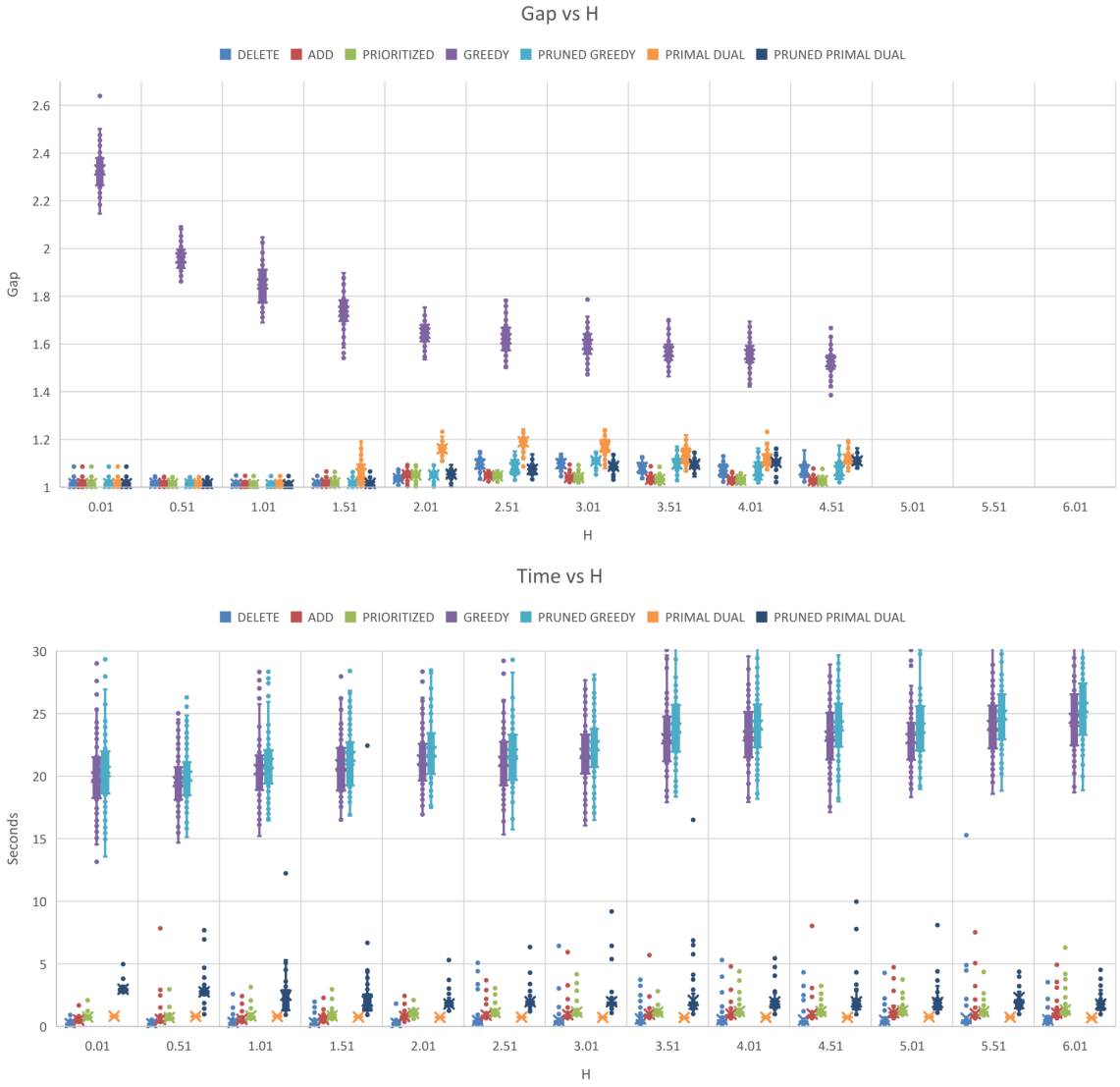


FIGURE 4.7: Delete, Add, and Prioritized each correspond to the local search that DELETEs, ADDs, and all operations, respectively. The gaps and running times for Delete, Add, Prioritized local search, Greedy, Pruned Greedy, Primal Dual, and Pruned Primal Dual are shown in blue, red, green, purple, light blue, orange, and dark blue, respectively.

- Pruned greedy improves the gap to 1.1 up to 1.11.
- Primal dual's gap is within 1.16 to 1.18.
- Pruned primal dual improves to 1.08 up to 1.09.
- Local search with DELETEs has slightly higher gap than pruned primal dual, from 1.09 to 1.1.
- Local search with ADDs and prioritized local search both have gaps of 1.04.

The heuristics in order from lowest to highest gap are local search with ADDs and prioritized local search, pruned primal dual, local search with DELETEs, pruned greedy, primal dual, and greedy.

Second, we provide the running times of the MIP and all heuristics. Generally, the solving times increase as N increases.

- The MIP incurs the highest running time, starting at over 658 seconds for $N = 110$ to over 2837 seconds at $N = 140$.
- Greedy and pruned greedy running time starts around 27 seconds for $N = 110$ and increases up to 62 seconds for $N = 160$.
- Pruned primal dual takes no more than 6 seconds for all values of N.
- Prioritized local search stays under 4 seconds throughout.
- Local search with ADDs takes under 3 seconds on all instances.
- Primal dual and local search with DELETEs both take at most 2 seconds throughout.

The methods in order from lowest to highest running time are local search with DELETEs, primal dual, local search with ADDs, prioritized local search, pruned primal dual, greedy, and pruned greedy.

Again, local search with ADDs is the best heuristic, having a gap of at most 1.04 throughout and running time of 2.01 seconds at $N = 140$, which is also three orders of magnitude faster than the MIP at the same value of N.

Varying T

Here, N and H are fixed to 50 and 2.6, respectively. The number of days T varies from 6 to 18 in increments of 2. Results that involve MIP values are available only up to $T = 14$ due to the MIP's high computation time.

First, we provide the results for the gap.

- As before, greedy has the highest gap among all heuristics, from 1.7 to 1.8 across the tested values of T. The gap decreases slightly as T increases because longer time horizon allows greedy more room to pick good sets that serve farther into the future.
- Pruned greedy has the opposite pattern as greedy. Its gap starts at 1.04 when $T = 6$ and increases to 1.11 when $T = 14$ because larger time horizon makes the instance harder for the prioritized local search used to prune greedy.
- Primal dual's gap ranges from 1.14 to 1.2, also increasing as T increases.
- Pruned primal gap improves the gap to 1.05 at $T = 6$ up to 1.15 at $T = 14$.

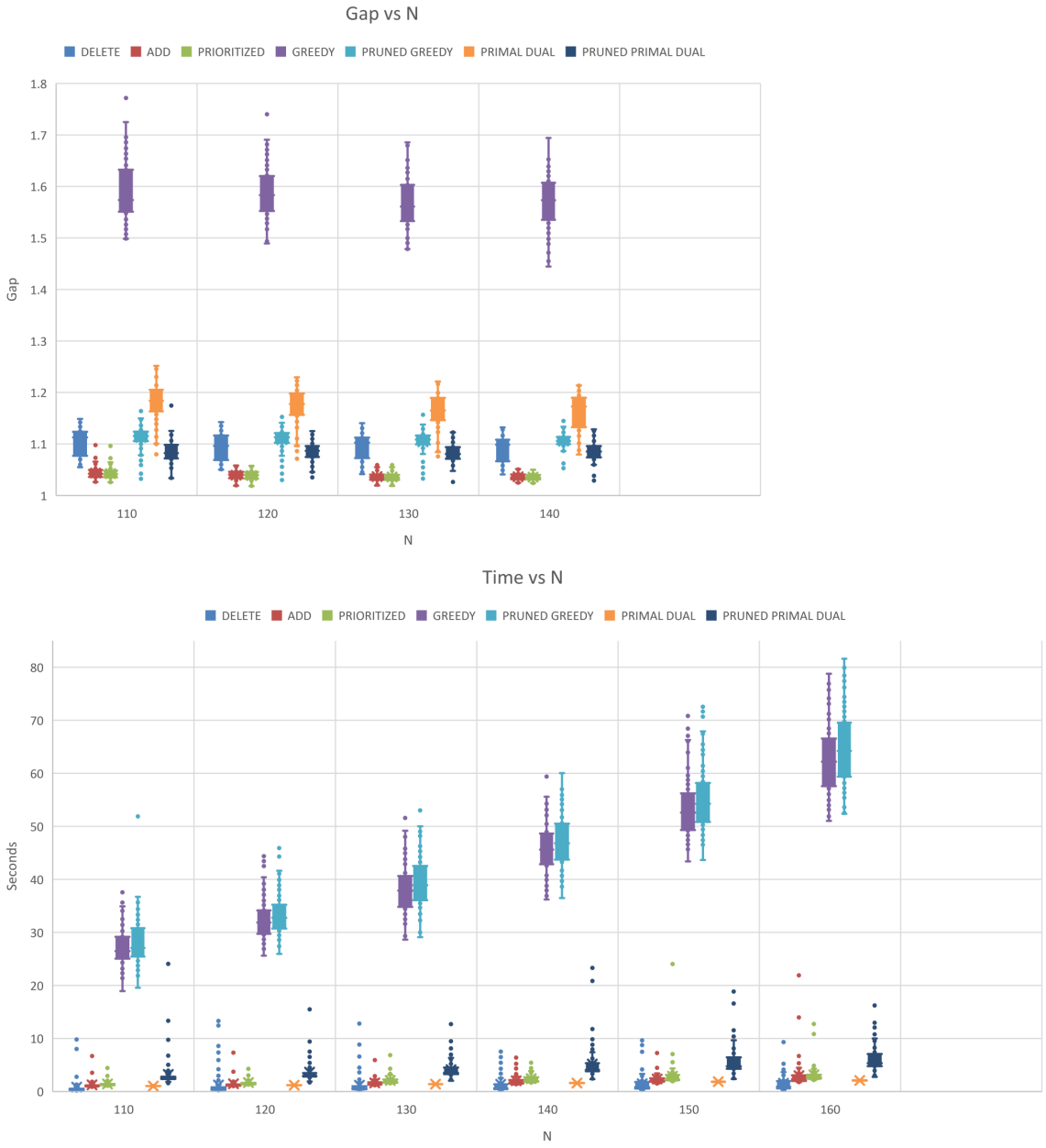


FIGURE 4.8: The gaps and running times for Delete, Add, Prioritized local search, Greedy, Pruned Greedy, Primal Dual, and Pruned Primal Dual are shown in blue, red, green, purple, light blue, orange, and dark blue, respectively.

- Local search with DELETEs' gap does not exhibit any trend with respect to T. The gap ranges from 1.03 to 1.09.
- Local search with ADDs and prioritized local search both have gaps starting from 1.03 at $T = 6$ to 1.08 at $T = 14$.

In order from lowest to highest gap, the heuristics are local search with ADDs and prioritized local search, local search with DELETEs, pruned primal dual, pruned greedy, primal dual, and greedy.

Second, we show the solving times. As expected, the solving times all increase as T increases.

- MIP solves in over 10 seconds for $T = 6$ and increases to over 1481 seconds at $T = 14$.
- Greedy and pruned greedy take around 6 seconds at $T = 6$, increasing to 30 seconds at $T = 18$.
- Pruned primal dual requires less than 3 seconds on all instances.
- Primal dual, prioritized local search, and local search with ADDs take under 2 seconds throughout.
- Local search with DELETEs take under 1 second throughout.

The methods in increasing order of solving time are local search with DELETEs, local search with ADDs, prioritized local search, primal dual, pruned primal dual, greedy, and pruned greedy.

As expected, local search with ADDs has the best gap of at most 1.08 through all test instances and a solving time of less than 1 second for $T = 14$, which is three orders of magnitude faster than the MIP at $T = 14$.

4.7.3 Conclusions

In this chapter, we introduced new local search, greedy, and primal dual heuristics that exploit the PCST as intermediate steps towards obtaining near-optimal solutions for IRP. We used the PCST subroutine to design very large neighborhood search methods, as well as to find low density sets to serve for the solution in Greedy and good primal visits for the solution for a primal-dual method. We proved a performance guarantee for Greedy and showed how the PCST method provides the needed properties of the solutions in these heuristics. Thus a main contribution of our work has been to extend the theoretical insights from the work of [49] in using the PCST problem as an effective tool to solve IRP. Among the different heuristics we experimented with, the local search with ADDs performed the best. For 140 by 6 instances, it was able to reach within 1.04 factor of optimal cost with a speedup of three orders of magnitude compared to solving the MIP using Gurobi. Similarly, for 50 by 14 instances, it reaches 1.08 factor of optimal cost with similar speedups. Generally, primal dual and greedy were less effective than the local search based methods because their better starting solution structure affords less flexibility in the later prioritized local search pruning phase. In all instance sets, greedy had the highest gap and the longest running time. An interesting research direction for future work is to extend the PCST-based approaches to the more general case of IRP with vehicle and inventory capacities.

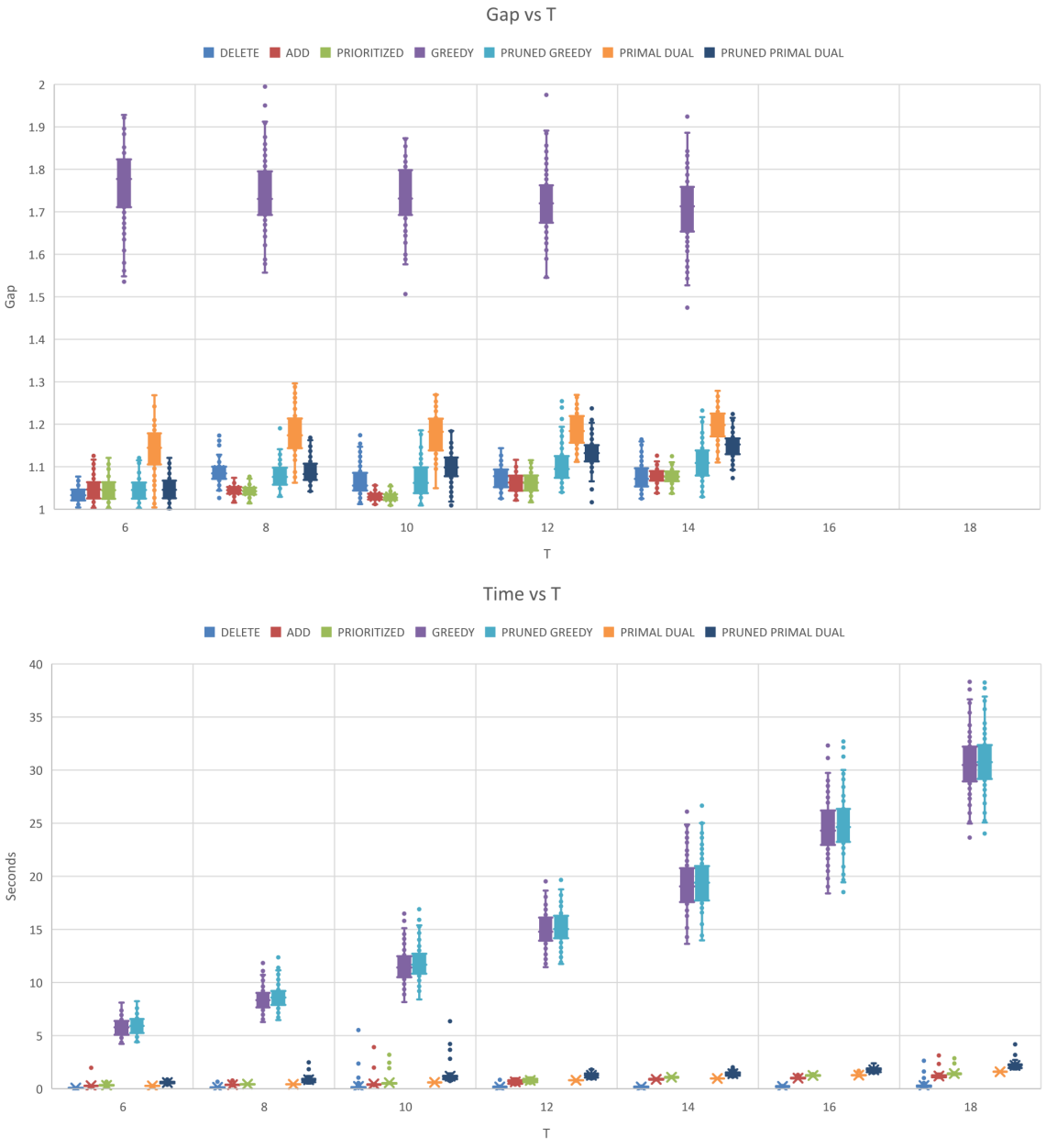


FIGURE 4.9: The gaps and running times for Delete, Add, Prioritized local search, Greedy, Pruned Greedy, Primal Dual, and Pruned Primal Dual are shown in blue, red, green, purple, light blue, orange, and dark blue, respectively.

4.8 Acknowledgments

We would like thank Afshin Nikzad for his contribution to the idea of solving PCSTs within the primal dual heuristic.

Chapter 5

Deadline Inventory Routing Problem

5.1 Introduction

The Deadline Inventory Routing Problem (Deadline IRP) is a new variant of the classical IRP [25]. In the classical single-depot IRP [23, 29, 25, 26], a set of client demand locations in a metric containing the depot is given, and for a planning horizon of T days, a daily demand at each client location is specified. The goal is to come up with vehicle routing schedules in each of the T days to stock the client demands before they materialize. However, early stocking at a location incurs a location- and duration-specific *inventory* holding cost that are also specified. If we assume the daily replenishing vehicle has infinite capacity, the distance traveled by the vehicle in a daily route translates to a *routing* cost. The goal of IRP is to find daily vehicle schedules for the T days that deliver enough supply at each location to meet all daily demands and minimizes the sum of inventory holding costs for units supplied ahead of their demand and the daily routing costs of the vehicle, over the T days.

The *Deadline IRP (DIRP)* replaces the daily client demands with a deadline for revisit. In other words, if the vehicle visits a location v at time t, this results in a requirement that the next visit of the vehicle to this location must occur by the time the resources delivered at v are depleted - this sets up a deadline for the next visit which is $t + D_v$ where D_v denotes the depletion time. The routing costs are unchanged from the classical IRP formulation. One example of where this alternate formulation appears is in the replenishment of automatic teller machines (ATMs) with cash [4]: based on the currency demand at each location and the service requirements of the bank, each visit automatically triggers a deadline for the next visit.

Problem Definition. Formally, the *Deadline Inventory Routing Problem* is given as input the following.

- a set V of vertices/clients on a metric c that defines the routing costs,
- a root node $r \in V$ from which an uncapacitated supply vehicle starts,
- a discrete time horizon $\{1, 2, \ldots, T\}$ over which the clients require service, and
- a length D_v of time called the *depletion time* for each client v that indicates how long the clients will last from the previous visit before its supplies run out.

The objective of deadline IRP is to minimize the vehicle routing cost while supplying the clients within their deadlines over the time horizon. We assume that the daily vehicle route is a tour that starts and ends in the root. We relax this to be a Steiner

tree connecting the root to the clients visited that day with a loss of a factor of 2 in the approximation guarantee in the sequel[1]. Note again that the deadlines are visit dependent.

5.2 Related Work

As mentioned before, our problem can be seen as a special case of the INVENTORY ROUTING PROBLEM (IRP) [34]. Here, clients (vertices) have their own storage with a certain capacity and for each day a demand is specified. The clients pay holding cost over their inventory. However, omitting inventory cost, we can interpret our problem as such an inventory routing problem in which the demand at any given location is the same every day.

Another closely related problem is the PERIODIC LATENCY PROBLEM [35], which features the recurring visits requirement of DIRP. We are given recurrence length q_i for each client i and travel distances between clients. Client i is considered *served* if it is visited every q_i time units. The server does not return to the depot at the end of each time unit (e.g. day), but keeps moving continuously between clients at uniform speed. Another difference between PERIODIC LATENCY PROBLEM and DIRP is the objective function. Coene et al. [35] study two versions of the problem: one that maximizes the number of served clients by one server, and one that minimizes the number of servers needed to serve all clients. They resolve the complexity of these problems on lines, circles, stars, trees, and general metrics.

A more restrictive problem than DIRP, but with a compact input size is known under the guise of PINWHEEL SCHEDULING. It has been introduced to model the scheduling of a ground station to receive information from a set of satellites without data loss. In terms of our problem no more than one vertex can be replenished per day and all distances to the depot are the same; the interesting question here is if there exists a feasible schedule for replenishing the vertices. Formally, a set of jobs $\{1, ..., n\}$ with periods $p_1, ..., p_n$ is given, and the question is whether there exists a schedule $\sigma : \mathbb{N} \to \{1, .., n\}$ such that $j \in \bigcup_{k=t+1}^{t+p_j} \sigma_k$, $\forall t \geq 0$ and $\forall j$. Note that the compact representation of time in Pinwheel Scheduling makes its complexity incomparable to that of our problem.

PINWHEEL SCHEDULING was introduced by Holte et al. [59], who showed that it is contained in PSPACE. The problem is in NP if the schedule σ is restricted to one in which for each job the time between two consecutive executions remains constant throughout the schedule. In particular this holds for instances with density $\rho = \sum_j 1/p_j = 1$ [59]. They also observed that the problem is easily solvable when $\rho \leq 1$ and the periods are harmonic, i.e. p_i is a divisor of p_j or vice versa for all i and j. As a corollary, every instance with $\rho \leq \frac{1}{2}$ is feasible.

Chan and Chin [31] improved the latter result by giving an algorithm that produces a feasible schedule for PINWHEEL SCHEDULING whenever $\rho \leq \frac{2}{3}$. In [30], they improved this factor to $\frac{7}{10}$. Later, Fishburn and Lagarias [47] showed that every instance with $\rho \leq \frac{3}{4}$ has a feasible schedule. All these papers work towards the conjecture that there is a feasible schedule if $\rho \leq \frac{5}{6}$. That this bound is tight can be seen by the instance with $p_1 = 2$, $p_2 = 3$ and $p_3 = M$, with M large. This instance cannot be scheduled, but has a density of $\frac{5}{6} + \frac{1}{M}$.

[1] This relaxation of the tour to be a tree is purely for the sake of simplification of the approximation algorithms for finding such a route - the algorithms are better understood for trees rather than tours. Moreover, we can convert trees to tours in the end with a further loss of factor of 2 in metrics using an Euler walk on the tree.

The complexity of PINWHEEL SCHEDULING has been open since it was introduced. It was only recently shown by Jacobs and Longo [60] that there is no pseudopolynomial time algorithm solving the problem unless SAT has an exact algorithm running in expected time $n^{O(\log n \log\log n)}$, implying for example that the randomized exponential time hypothesis fails to hold [24, 36]. Since the latter is unlikely, one could conclude that PINWHEEL SCHEDULING is not solvable in pseudopolynomial time. It remains open whether the problem is PSPACE-complete.

Similar to PINWHEEL SCHEDULING, the k-SERVER PERIODIC MAINTENANCE PROBLEM [72, 15, 42] has n jobs, each with a specified periodicity and a processing time. Each server may serve at most one job per time unit. However, job i is required to be served exactly every m_i days apart rather than within every m_i days. The case $k = 1, c_j = 1$ for all j is analogous to PINWHEEL SCHEDULING, except for the exact periodicity constraint. For any $k \geq 1$, Mok et al. [72] have shown it is NP-complete in the strong sense. For the special case when m_i are multiples of each other or when there are at most 2 different periodicities, they have shown it is in P.

Other related problems with a compact input representation include real-time scheduling of sporadic tasks [14, 20], where we are given a set of recurrent tasks. On a single machine, EDF (Earliest Deadline First) is optimal. However, we remark that the complexity of deciding whether a given set of tasks is feasible has been open for a long time and only recently proved showing that it is coNP-hard to decide whether a task system is feasible on a single processor even if the utilization is bounded [43].

Another related problem is the BAMBOO GARDEN TRIMMING PROBLEM introduced by Gasieniec et al. [51]. There are n bamboos, each having a given growth rate, which may be viewed as inducing a periodicity. On each day, a robot may trim at most one bamboo back to height 0. The goal is to minimize the maximum height of the bamboos. Gasieniec et al. provide a 4-approximation for the general case and a 2-approximation for balanced growth rates.

The full paper including results for DIRP on special metrics appeared in the proceedings of the 13th Latin American Symposium [21].

5.3 Approximation Algorithms

In this section, we study the design of near-optimal periodic solutions for DIRP. First we show that the periods can be restricted to be of the same order of magnitude as the maximum client deadline, and all deadlines can be assumed to be powers of two with a constant factor loss in optimality. Next, we prove a logarithmic approximation algorithm by using the ideas of increasing and synchronized solutions that we define and approximate.

We reduce arbitrary Deadline Inventory Routing Problems to those whose depletion times are powers of 2 losing only a constant factor.

Lemma 5.3.1. *Let Π be a Deadline Inventory Routing Problem instance. For a client v, denote its depletion time by T_v. For any time t, let $r(t)$ be the largest power of 2 not exceeding t. Let Π' be the Deadline Inventory Routing Problem instance having the same vertex set as Π, but depletion time $r(T_v)$ for each $v \in V$. Let OPT and OPT' be the minimum cost of Π and Π'. Then any solution feasible for Π' is feasible for Π, and $OPT' \leq 4OPT$.*

Proof. Feasibility for Π' implies feasibility for Π by $r(T_v) \leq T_v$ for all v. To show that $OPT' \leq 4OPT$, we show that any solution S of Π can be converted to a solution S' of Π' such that $c(S') \leq 4c(S)$. Given a solution S to Π, let $\hat{S}$ be the solution such that for each t, copy the tree of S at time t to time $\lceil \frac{t}{2} \rceil$ in addition to the original tree at time t.

We claim that $\hat{S}$ is feasible to the first half subinstance of Π'. To show this, observe no $\hat{S}$ visit between t_k and t_{k+1} implies no S visit between $2t_k$ and $2t_{k+1}$. If $t_{k+1} - t_k \geq r(T_v)$, then $2t_{k+1} - 2t_k \geq 2\lceil \frac{T_v}{2} \rceil \geq T_v$, which would imply that S was not feasible to Π. So $\hat{S}$ must be feasible to at least the first half subinstance of Π'. By construction, $c(\hat{S}) \leq 2c(S)$.

Let S' be the solution that copies the trees of $\hat{S}$ in the first half of the instance to the second half and retains the same first half as $\hat{S}$. Then S' is feasible to the entire instance Π', and $c(S') \leq 2c(\hat{S}) \leq 4c(S)$. □

Consider a powers of 2 instance of the Deadline Inventory Routing Problem in which depletion times span from 2^0 to 2^L. Denote by V_{2^j} the set of vertices who have depletion time 2^j; so $V = V_1 \cup \cdots \cup V_{2^L}$. Define a *synchronized* solution to be one which, for each $l \leq L$, visits exactly $V_1 \cup \cdots \cup V_{2^l}$ on day 2^l. Given a tree T covering some subset of V, we say that T is *non-decreasing* with respect to the Deadline Inventory Routing Problem instance if for each $v \in V(T)$, $v \in V_{2^j}$ implies that the r-v path in T contains only vertices from V_{2^l} for $l \leq j$.

Proposition 5.3.2. *Given a solution S to a powers of 2 instance of Deadline Inventory Routing Problem, if the tree in S for each day is non-decreasing, then there is a synchronized solution S' of cost at most $c(S)$.*

Proof. We shall use each edge of S at most once to build the trees for S'. First, we know that for each $j \leq L$, every vertex of V_{2^j} occurs at least once per time window of length $2^j - 1$ in S. So S has paths that reach all of V_{2^j} for each window of the form $[k2^j + 1, (k+1)2^j]$. We will use this to build paths for each j that reach V_{2^j} on all days that are multiples of 2^j.

In the base case, we build paths between r and v for all $v \in V_{2^0}$ once for each day by using exactly the same V_{2^0}-reaching paths in S. For the remaining vertices, we build the synchronized solution inductively. Starting from $j = 1$ to L, for each k within 0 to $2^{L-j} - 1$, take the union of all paths of S occurring between $V_{2^{j-1}}$ and V_{2^j} within days $[k2^j + 1, (k+1)2^j]$ and place the edge union at time $(k+1)2^j$. Note that we already have paths from r to $V_{2^{j-1}}$ per day of the form $l2^{j-1}$ from the earlier rounds. So the current round has guaranteed to reach V_{2^j} per day of the form $l2^j$. At the end, we have a solution that visits each V_{2^j} on all days that are multiples of 2^j and uses each edge of S only once. □

Now, we show that arbitrary trees can be approximated by nondecreasing trees within logarithmic factor. To do so, we use a tree pairing lemma of Klein and Ravi's [64] that will help us construct the appropriate nondecreasing trees. We include the proof for completeness.

Lemma 5.3.3. *[64] Given any tree T and an even subset S of its vertices, there is a pairing of vertices covering S such that the tree-path induced by the pairs are edge-disjoint.*

Proof. Let T be a tree on vertex set V and $S \subset V$. For a pairing P, let $\phi(P)$ be the total length of the paths induced by the pairs in P. We will show that a pairing P^* minimizing $\phi(P^*)$ has all edge-disjoint pair-induced paths. Suppose there are pairs $\{u_1, v_1\}$ and $\{u_2, v_2\}$ that share some edge $\{a, b\}$. WLOG, assume that u_1 and u_2 are closer to a than b and v_1 and v_2 are closer to b than a. Then repairing the four vertices to $\{u_1, u_2\}$ and $\{v_1, v_2\}$ reduces the total length of the paths. So a pairing minimizing the total length cannot have any pairs that overlap. □

Using the tree pairing lemma 5.3.3, we will construct nondecreasing trees to approximate arbitrary trees. First we define the notations needed for the algorithm.

For $v \in V$, let $l(v)$ be the label of the class v belongs to, i.e. $V_{l(v)} \ni v$. A *nondecreasing arc* $a(\{u,v\})$ of $\{u,v\}$ is the arc between u and v that points from the vertex of lower label to the vertex of higher label (ties are broken arbitrarily). The vertex of lower (higher) label between u and v is denoted by $L(\{u,v\})$ ($H(\{u,v\})$). We denote by U the unpaired vertices and A the arcs of the nondecreasing tree being constructed.

Initialize $U \leftarrow V$ and $A \leftarrow \emptyset$.
while $|U| > 0$ **do**:
    Find an edge-disjoint pairing P of a largest even subset of U.
    for $\{u,v\} \in P$ **do**:
        $A \leftarrow A \cup a(\{u,v\})$.
        $U \leftarrow U \setminus H(\{u,v\})$.
    end for
end while

Proposition 5.3.4. *Given an arbitrary tree T of cost $c(T)$ with respect to vertex sets $V_0, \ldots, V_L$, there is a nondecreasing tree of cost at most $\lceil \log_2 n \rceil c(T)$.*

Proof. Consider an instance of DIRP. Let T be a tree. We will construct a nondecreasing tree T' by iteratively pairing off the vertices and directing each pair in a nondecreasing manner.

In the algorithm, we apply the pairing procedure $\lceil \log_2 n \rceil$ times to get a nondecreasing tree of the desired cost. In each round, we pair a largest subset of V such that the pairs induce edge disjoint paths in T. Then we direct each pair $\{u,v\}$ from the vertex of lower label to the vertex of higher label according to the labeling l and delete the vertex of higher label from consideration. These arcs are added to the arc set of T'. We can think of each pair as a connected component represented by the vertex of the lower label. In the end, T' is finalized when no unpaired vertices remain. Note that picking the vertex of minimum label as the representative per connected component ensures that the final tree is indeed an out-arborescence from the vertex of smallest label.

In each round, we used each edge of T at most once since all pair-induced paths were edge-disjoint. Let $\kappa(t)$ be the number of vertices at the beginning of round t. Since each round paired off either all vertices or all but vertex, we have $\kappa(t) = \lceil \kappa(t-1)/2 \rceil$. So the total number of rounds is $\lceil \log_2 n \rceil$. Hence $c(T') \leq \lceil \log_2 n \rceil c(T)$. □

Theorem 5.3.5. *There is a $O(\log n)$-approximation for the Deadline Inventory Routing Problem.*

Proof. Given an DIRP instance, OPT denote the cost of an optimal solution. Let ρ_{ST} be the best known approximation ratio for Steiner tree. We will construct a synchronized solution to DIRP that has cost at most $O(\log n)OPT$. Note that the subset of clients to visit per day is completely determined in any synchronized solution. So optimal synchronized solutions may be approximated within ρ_{ST} factor. As ρ_{ST} is constant, it suffices to show that optimal synchronized solutions to DIRP are within $O(\log n)$ factor from OPT.

By proposition 5.3.4, there is a nondecreasing solution S' to DIRP of cost at most $\lceil \log_2 n \rceil OPT$. By Proposition 5.3.2 on each tree of S', there is a synchronized solution $\hat{S}$ to DIRP of cost $c(S')$. So an optimal synchronized solution to DIRP has cost at most $c(\hat{S}) \leq c(S') \leq \lceil \log_2 n \rceil OPT$. □

However, the above method cannot be improved. There is a class of instances for which the cost of optimal solutions is logarithmic factor away from the cost of

optimal monotone solutions. The class of examples is due to a co-author, Thomas Bosman.

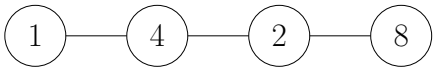
Proposition 5.3.6. *There exists a class of instances in which there is a logarithmic optimality gap between the optimal and the optimal non-decreasing solution.*

Consider the following sequence of sequences $(a^0, a^1, \ldots)$ where $a^0 = (1)$ and a^{i+1} is generated by alternatingly taking an element from a^i and then from the sequence $b^i = (2^i + 1, 2^i + 2, \ldots, 2^{i+1})$. For example:

- $a^0 = 1$
- $a^1 = 1, 2$
- $a^2 = 1, 3, 2, 4$
- $a^3 = 1, 5, 3, 6, 2, 7, 4, 8$
- $\ldots$

Then define the (unweighted) graph G_i as the path graph with 2^i vertices, where the jth vertex has depletion time $2^{a^i_j - 1}$. See Figure 5.1 for an example.

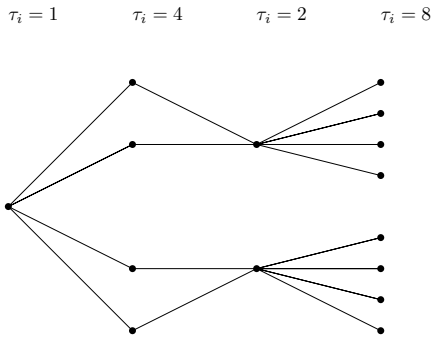
FIGURE 5.1: Illustration of G_2, (depletion times in circles)



The minimum spanning tree in G_i costs $2^i - 1$. It is easy to check that the decreasing spanning tree produced by Lemma 5.3.3 costs $i2^{i-1}$. Moreover, since the solution produced attaches every vertex to a nearest vertex with lower depletion time, it must be optimal.

To show tightness of our main theorem, we will define another class of graphs H_i for $i \geq 1$ that are constructed from $\{G_i\}$. The idea is to make τ_j copies of each terminal j, and then connect them in a regular way, for example like in Figure 5.2.

FIGURE 5.2: Illustration of H_2



Formally H_i is constructed as follows. For simplicity of description, we assume that G_i is planarly embedded from left to right, and we assume that we keep a planar embedding of H_i during construction.

We first copy node 1 to H_i. Now we work from left to right, starting from the second node. When we are at node j, we put τ_j copies of j vertically above each other and to the right of the copies of $j - 1$ in H_i. Then we connect them to the copies of $j - 1$ in H_i such that the graph remains planar and all copies of $j - 1$ have the same degree, and all copies of j have the same degree. This can be done in only one way. Furthermore we identify vertex 1 with the depot.

Proposition 5.3.7. *The instance induced by H_i has a logarithmic optimality gap between the optimal and the optimal non-decreasing solution.*

Proof. There exists an obvious solution that visits exactly one client of each turnover time per day, that costs $2(2^i - 1)$.

Now suppose we impose non-decreasing constraints. In this case we need to use (on average) at least one edge pointing from a client with a lower depletion time to one with a higher depletion time per day. But from our reasoning on the decreasing minimum spanning tree in G_i, we find that the cheapest set of edges that contains at least one edge pointing from a client with depletion time 2^i to one with lower depletion time for all i, costs at least $i2^{i-1}$. Therefore the optimal solution under non-decreasing constraints is at least a logarithmic factor more expensive than the optimal solution. □

Next, we show an $O(\log T)$-approximation rounding an LP formulation of DIRP, where we may assume $T = \max_{v \in V} D_v$. Note that T is incomparable with n, thus making this result potentially interesting in relation to Theorem 5.3.5. We first give the LP for DIRP.

$$\begin{aligned}
\min \quad & \sum_{uv \in A} \sum_{s=1}^{T} c_{uv} y_s^{uv} & & \\
\text{s.t.} \quad & \sum_{s \in [k, k+D_v-1]} x_s^v \geq 1 & \forall v \in V, k \leq T - D_v & \quad (5.1)\\
& \sum_{uw \in \delta^+(X)} y_s^{uw} \geq x_s^v & \forall v \in V, s \leq T, X \subset V \text{ s.t. } X \not\ni v, X \ni r & \quad (5.2)\\
& x_s^v \geq 0 & \forall v \in V, s \leq T & \quad (5.3)\\
& y_s^{uw} \geq 0 & \forall uw \in A, s \leq T. & \quad (5.4)
\end{aligned}$$

Variable x_s^v indicates whether client v is visited on day s. Variable y_s^{uw} indicates whether arc uw is used on day s. The first constraint ensures that each client is visited within its depletion time over the time horizon. The second constraint ensures that if client v is visited on day s, then the tree built on day s must reach v.

Theorem 5.3.8. *There is an $O(\log T)$-approximation for the Deadline Inventory Routing Problem.*

Proof. Let (x, y) be an optimal solution to the LP for DIRP. For each set V_i, we will obtain trees that cover V_i every 2^i days with cost at most $c \cdot y$. Doing so separately per V_i will eventually incur a factor of $O(\log T)$.

To obtain the desired trees per $i \leq L$, we round Steiner trees on the LP values per time interval of length 2^i. Specifically, for each $0 \leq i \leq L, 0 \leq l \leq 2^{L-i} - 1, v \in V, a \in A$, define the aggregate LP values $x^v_{li,(l+1)i} := \sum_{s=l2^i+1}^{(l+1)2^i} x_s^v$ and $y^a_{li,(l+1)i} := \sum_{s=l2^i+1}^{(l+1)2^i} y_s^a$. For each i and l, we know that $x^v_{li,(l+1)i} \geq 1$ for all $v \in V_i$ by constraint (5.1). By aggregating constraint (5.2), we have $\sum_{a \in \delta^+(X)} y^a_{li,(l+1)i} \geq x^v_{li,(l+1)i} \geq 1$ for all $v \in V_i, X \subset V$ such that $X \not\ni v, X \ni r$. In particular, the variables $y^a_{li,(l+1)i}$ satisfy the Steiner tree LP with terminal set V_i. Since the Steiner tree LP has integrality gap at most 2, there is an integral Steiner tree covering V_i of cost at most $2\sum_{a \in A} c_a y^a_{li,(l+1)i}$. Add the approximate Steiner tree on day $l2^i$ to our solution. Doing so for each $l \in [0, 2^{L-i} - 1]$ will ensure that each interval of length 2^i has a tree that covers V_i with at most twice the cost of the LP solution. Furthermore, the arcs in all of the trees

covering V_i throughout the whole time horizon can be paid for by just one copy of y since $y^a_{li,(l+1)i}$ charges distinct portions of y for each l.

To cover V_i for all $0 \leq i \leq L$, we will use at most $L+1$ copies of y. Since $L = \lceil \log T \rceil$, the final solution has cost at most $(\lceil \log T \rceil + 1)OPT$. □

5.4 Conclusion

We provided an $O(\log n)$-approximation and an $O(\log T)$-approximation for Deadline IRP on general metrics. For the $O(\log n)$-approximation, we showed a class of instances such that the method cannot be improved. The full paper [21] from our collaborations with coauthors also resolve special cases of Deadline IRP, i.e., on paths and trees. In addition, we study Deadline IRP with a Min Max objective, where the goal is to minimize the cost of the largest route among the routes per day. For Min Max Deadline IRP on general metrics, we also gave logarithmic approximations with respect to n and T, respectively. It remains open to design approximation algorithms with better guarantees for either versions of the Deadline IRP, especially constant factors.

Chapter 6

Inventory Routing Problem with Facility Location

We study Inventory Routing with Facility Location assuming that facility to client connections directly use the edge between them and call this variant Star Inventory Routing Problem with Facility Location. The *Star Inventory Routing Problem with Facility Location (SIRPFL)* is inventory routing with the extra choice to build depots at a subset of the locations for additional costs before the first day, which then can be used to route deliveries throughout the entire time horizon. Formally, we are given a graph G with edge weights w_e, a time horizon $1, \ldots, T$, a set D of demand points (v, t) with d_t^v units of demand due by day t, facility costs f_v for vertex v, holding costs $h_{s,t}^v$ per unit of demand delivered on day s serving (v, t). The objective is to open a set $F \in V$ of facilities that can be used throughout the entire time horizon, determine the set of demands to serve per day, and connect any visited clients to opened facilities per day so that the total cost from facility openings, client-facility connections, and storage costs for early deliveries is minimized.

6.1 Related Work

A related problem is the *Tree IRPFL*, which has the same requirements except that the connected components are trees instead of stars. Tree IRPFL allows saving connection costs by connecting clients through various other clients who are connected to an opened facility. Single-day variants of Tree IRPFL have been studied extensively. Some of the approximation ratios depend on the best known ratios of other related problems. We use ρ_Π to denote the best existing approximation ratio for problem Π. For uncapacitated single-day Tree IRPFL, Goemans and Williamson [55] provide a $(2 - \frac{1}{|V|-1})$-approximation. If clients are given in groups such that only one client per group needs to be served, Glicksman and Penn [53] generalize the previous result to $(2 - \frac{1}{|V|-1})L$-approximation, where L is the largest size of a group.

For the capacitated single-day case of Tree IRPFL, Harks et al. [58] provide a 4.38-approximation. They also give constant approximations for the prize collecting variant and the cross-docking variant. For the group version of the problem, Harks and König show a $4.38L$-approximation. Ravi and Sinha [79] give a $(\rho_{\text{ST}} + \rho_{\text{UFL}})$-approximation for a generalization of the capacitated single-day Tree IRPFL called Capacitated-Cable Facility Location (CCFL). ST stands for Steiner Tree and UFL stands for Uncapacitated Facility Location. In CCFL, the amount of demand delivered through each edge must be supported by building enough copies of cables on the edge. They give a bicriteria $(\rho_{k-\text{MEDIAN}} + 2)$-approximation opening $2k$ depots for the k-median version of the CCFL, which allows k depots to be located at no cost.

6.2 Inventory Access Problem

The *Inventory Access Problem (IAP)* is the single client case of the Inventory Routing Problem. The only decision needed is to determine on each day whether to visit the client and how much supply to drop off. In SIRPFL, if we know where to build the facilities, then the best way to connect clients would be to the closest opened facility. So once facility openings are determined, the remaining problem decomposes into solving IAP for every client.

6.2.1 Uncapacitated IAP

Uncapacitated IAP is the simplest special case of SIRPFL. This can be solved by an efficient dynamic program as follows. Let $C(t, l)$ be the minimum cost to satisfy all demands within the instance containing only demands up from day 1 to day t such that l is the last day of service within day t in the optimal solution for this restricted instance. To obtain the recurrence for $C(t+1, l)$, observe that we must pay routing cost w_{rv} for the trip on day l and storage cost $\sum_{s=l+1}^{t+1} h_{l,s} d_s$ for all demands occurring after day l since l is the latest visit before their due date (i.e. serving them on day l minimizes the holding cost required to satisfy those demands). Then the remaining cost is determined by the cost to serve an instance up to day $l-1$ with last service day $k \leq l-1$, minimized over the choices k. Hence the following recurrence solves Uncapacitated IAP:

$$C(t+1, l) = \begin{cases} \min_{k \leq l-1} C(l-1, k) + \sum_{s=l+1}^{t+1} h_{l,s} d_s + w_{rv}, & l \neq t+1 \\ \min_{k \leq t} C(t, k) + w_{rv}, & l = t+1 \end{cases}$$

6.2.2 Capacitated Unsplittable IAP

To model more realistic scenarios, we now impose a capacity U on the supply vehicle. The vehicle may make multiple trips in one day to meet the required demands. Also, we assume that demands are *unsplittable*, i.e., each demand is within capacity and must be completely delivered in one trip.

We show that Capacitated Unsplittable IAP is weakly NP-hard by reducing from Number Partitioning.

Definition In *Number Partitioning*, we are given a set S of positive integers and wish to determine whether there is a subset $X \subset S$ such that $\sum_{a \in X} a = \sum_{b \in S \setminus X} b$.

Theorem 6.2.1. *Number Partitioning $\leq_P$ Capacitated Unsplittable IAP.*

Proof. Let S be the set in a given instance of the Number Partitioning problem. We will create an instance I of Capacitated Unsplittable IAP as follows. For each $a \in S$, create a demand point on day t_a with a units of demand. Set all holding costs to 0. This means that serving all demands on the single day $\arg\min_a t_a$ forms a valid optimal solution. Set the capacity of the vehicle to $U := \frac{\sum_{a \in S} a}{2}$. Let the distance between the depot and the client be any positive number w_{rv}. We will show that there is a solution of cost at most $2w_{rv}$ to I if and only if S has a valid number partitioning.

First, we prove the forward direction. Assume that there is a solution to I of cost at most $2w_{rv}$. Since the total demand is $2U$, any solution must cost at least $2w_{rv}$. Furthermore, the only way to obtain cost exactly $2w_{rv}$ is to drop off exactly $U = \frac{\sum_{a \in S} a}{2}$ units of demand per trip in two trips total. Let X be the set of numbers

corresponding to the demands in the first trip. By definition of the trips, we have $\sum_{a\in X} a = U = \sum_{b\in S\setminus X} b$.

Second, we prove the backward direction. Assume that there is a number partitioning $X \subset S$. Then serving X and $S \setminus X$ each in a trip on day 1 forms a feasible solution since $\sum_{a\in X} a = U = \sum_{b\in S\setminus X} b$. The cost of each trip is w_{rv}, which yields a total cost of $2w_{rv}$. □

6.2.3 Capacitated Splittable IAP

Here, we consider *Capacitated Splittable IAP*, in which a single demand is allowed to be served in parts over multiple days. Let W be the distance between the depot and the client. Denote by $h_{s,t}$ the holding cost to store one unit of demand from s to deadline t. The demand with deadline t is denoted by d_t. As before, U denotes the capacity of the vehicle. We model Capacitated Splittable IAP by the following LP relaxation.

$$\begin{aligned}
\min \quad & \sum_{s\le T} W y_s + \sum_{t\in D}\sum_{s\le t} h_{s,t} d_t x_{s,t} \\
\text{s.t.} \quad & \sum_{s\le t} x_{s,t} \ge 1 && \forall t \in D && (6.1)\\
& y_s \ge \sum_{t=s}^{T} \frac{x_{s,t} d_t}{U} && \forall s \le T && (6.2)\\
& y_s \ge x_{s,t} && \forall t \le T, s \le t && (6.3)\\
& x_{s,t} \ge 0 \forall t \in D, && && (6.4)
\end{aligned}$$

The variable y_s indicates the number of trips on day s. Variable $x_{s,t}$ indicates the fraction of d_t to deliver on day s. Constraint 6.1 requires that each demand becomes entirely delivered by the due date (possibly split over multiple days). Constraint 6.2 ensures that the total demand that day s serves do not exceed the total capacity among all trips on day s. Constraint 6.3 ensures that there is a trip whenever some delivery is made on day s.

Let (x, y) be an optimal LP solution. For convenience of the analysis, let $r(x, y) = \sum_{s\le T} W y_s$ and $h(x, y) = \sum_{t\in D}\sum_{s\le t} h_{s,t} d_t x_{s,t}$. We will use the LP values $x_{s,t}$ to determine when to visit the client and which demands to drop per visit. For each $t \in D$, let s_t be the latest day for which $\sum_{s=s_t}^{t} x_{s,t} \ge \frac{1}{2}$. Denote by t_L the latest day among all demand days. We will keep track of the visit set S along with an anchor set A consisting of demand days that caused the creation of new visits.

For the analysis, denote by T_s the set of all demand days t such that t was satisfied by s in Algorithm 10.

Proposition 6.2.2. *The holding cost of the solution from Algorithm 10 is at most* $2h(x, y)$.

Proof. 1. Assume that $t \in A$. Then t was served on day s_t, i.e., incurs holding cost $h_{s_t,t} d_t$. To pay for the holding cost, we use the following part of the LP

$$\begin{aligned}
\sum_{s=1}^{s_t} h_{s,t} d_t x_{s,t} &\ge h_{s_t,t} d_t \sum_{s=1}^{s_t} x_{s,t} \\
&\ge \frac{h_{s_t,t} d_t}{2}.
\end{aligned}$$

2. Assume that $t \notin A$. Let $\tilde{s}$ be the latest day in S such that $\tilde{s} \le t$. Then the holding cost incurred by the demand on day t is $h_{\tilde{s},t} d_t$. By definition of the

Algorithm 10 Visit Rule for Capacitated Splittable IAP

Initialize $A \leftarrow \{t_L\}$.
Initialize $S \leftarrow \{s_{t_L}\}$.
for unsatisfied demand day $t \geq s_{t_l}$ **do**
    satisfy t by dropping off d_t on day s_{t_L}.
end for
while there is any unsatisfied demand **do**
    Denote by t the unsatisfied demand day with the latest s_t
    $A \leftarrow A \cup \{t\}$.
    $S \leftarrow S \cup \{t\}$.
    Satisfy t by dropping off d_t on day s_t.
    for unsatisfied demand day $\hat{t} \geq s_{\hat{t}}$ **do**
        satisfy $\hat{t}$ by dropping off $d_{\hat{t}}$ on day s_t.
    end for
end while
Output the visit set S.

chosen visit days S, t was not chosen as anchor because s_t was earlier than $\tilde{s}$. So we pay for the holding cost using

$$\begin{aligned}\sum_{s=1}^{\tilde{s}} h_{s,t} d_t x_{s,t} &\geq \sum_{s=1}^{s_t} h_{s,t} d_t x_{s,t} \text{ by } s_t \leq \tilde{s} \\ &\geq \frac{h_{s_t,t} d_t}{2} \\ &\geq \frac{h_{\tilde{s},t} d_t}{2} \text{ by monotonicity of holding costs.}\end{aligned}$$

□

Proposition 6.2.3. *The routing cost of the solution from Algorithm 10 is at most* $3r(x,y)$.

Proof. For each visit day $s_{\tilde{t}} \in S$, the number of trips made is $\left\lceil \frac{\sum_{t \in T_{s_{\tilde{t}}}} d_t}{U} \right\rceil \leq \frac{\sum_{t \in T_{s_{\tilde{t}}}} d_t}{U} + 1$. So the total number of trips made is at most $\sum_{\tilde{t} \in A} \left(\frac{\sum_{t \in T_{s_{\tilde{t}}}} d_t}{U} + 1 \right) \leq \left(\sum_{\tilde{t} \in A} \frac{\sum_{t \in T_{s_{\tilde{t}}}} d_t}{U} \right) + |A|$. We will use 3 copies of $\sum_{s=1}^{T} y_s$ to pay for the routing cost–1 copy to pay for the first term and 2 copies to pay for the second term. The total LP budget for the number of trips is

$$\begin{aligned}\sum_{s=1}^{T} y_s &\geq \frac{\sum_{s=1}^{T} \sum_{t=s}^{T} x_{s,t} d_t}{U} \text{ by constraint 6.2} \\ &\geq \frac{\sum_{t=1}^{T} \sum_{s=1}^{t} x_{s,t} d_t}{U} \\ &\geq \sum_{t=1}^{T} \frac{d_t}{U} \\ &\geq \sum_{\tilde{t} \in A} \frac{\sum_{t \in T_{s_{\tilde{t}}}} d_t}{U}.\end{aligned}$$

So we can pay for the first term using one copy of the LP budget from all the y variables.

To pay for the second term, we will use constraint 6.3 instead so that we can use disjoint intervals of y for different anchors. In particular, for anchor $\tilde{t}$, we will charge

$$\begin{aligned} 2\sum_{s=s_{\tilde{t}}}^{\tilde{t}} y_s &\geq \sum_{s=s_{\tilde{t}}}^{\tilde{t}} x_{s,t} \\ &\geq 2 \cdot \frac{1}{2} \text{ by definition of } s_{\tilde{t}} \\ &= 1. \end{aligned}$$

By the construction of A, for any $t_1, t_2 \in A$, we have $[s_{t_1}, t_1] \cap [s_{t_2}, t_2] = \emptyset$. So the payment for different anchors use disjoint portions of y. Hence the second term can be payed for within 2 copies of the budget provided by y. □

Since all costs are bounded within 3 times the optimal cost, we have the following Corollary.

Corollary 6.2.4. *Algorithm 10 is a 3-approximation.*

6.3 Uncapacitated SIRPFL

In this section, we give a constant approximation for Uncapacitated SIRPFL. First, we state the LP formulation for Uncapacitated SIRPFL. Let z_v indicate whether a facility at v is opened, y_s^{uv} indicate whether edge uv is built on day s, y_{st}^{uv} indicate whether to deliver the demand of (v,t) on day s from facility u, and $x_{s,t}^v$ indicate whether demand point (v,t) is served on day s. Then Uncapacitated SIRPFL has the following LP relaxation.

$$\begin{aligned}
\min \quad & \sum_{v\in V} f_v z_v + \sum_{s\leq T}\sum_{e\in E} w_e y_s^e + \sum_{(v,t)\in D}\sum_{s\leq t} H_{s,t}^v x_{s,t}^v \\
\text{s.t.} \quad & \sum_{s\leq t} x_{s,t}^v \geq 1 && \forall (v,t)\in D && (6.5)\\
& \sum_{u\in V} y_{st}^{uv} \geq x_{s,t}^v && \forall (v,t)\in D, s\leq t && (6.6)\\
& z_u \geq \sum_{s=1}^{T} y_{st}^{uv} && \forall (v,t)\in D, u\in V && (6.7)\\
& y_s^{uv} \geq y_{st}^{uv} && \forall (v,t)\in D, u\in V, s\leq t && (6.8)\\
& z_u \geq y_s^{uv} && \forall u,v\in V, s\leq T && (6.9)\\
& \sum_{u\in V}\sum_{s=s'}^{t_2} y_{st_2}^{uv} \geq \sum_{u\in V}\sum_{s=s'}^{t_2} y_{st_1}^{uv} && \forall v\in V, t_2 > t_1 \geq s' && (6.10)\\
& z_u, y_r^e, y_{l,m}^a x_{s,t}^v \geq 0 && \forall u,v\in V, e,a\in E, r,m,t\leq T, l\leq m, s\leq t. && (6.11)
\end{aligned}$$

Constraint 6.5 requires that every demand point is served by its deadline. Constraint 6.6 enforces that v gets connected to some facility on day s if (v,t) is served on day s. Constraint 6.7 ensures that facility u is opened to the extent that u is assigned to any demand point over the time horizon. Observe that this lower bound for z_u is valid for any optimal solution since any demand point (v,t) only needs to connect

to some facility on at most one day – the day that the demand at (v,t) is served. Any extra connectivity would cause the solution to incur higher cost than necessary. So optimal solutions satisfy constraint 6.7. Constraint 6.8 ensures that whenever (v,t) is served on day s from u, an edge between u and v must be built on day s. Constraint 6.9 ensures that whenever some client v is connected to u on some day s, a facility must e built at u. Constraint 6.10 is valid for optimal solutions since for any v, if there is a service to (v,t_1) within $[s',t_1]$ and $t_1 < t_2$, then the service to t_2 is either on the same day or later, i.e., there must be a service to (v,t_2) within $[s',t_2]$.

Using the above LP formulation, we provide an LP rounding algorithm. Before stating the algorithm, we define the necessary notation. First, let (x,y,z) be an optimal LP solution. Let $f(x,y,z)$, $r(x,y,z)$, and $h(x,y,z)$ denote the facility cost, routing cost, and holding cost of (x,y,z) respectively. Define $s_{v,t}$ to be the latest day s^* such that $\sum_{u\in V}\sum_{s=s^*}^{t} y_{st}^{uv} \geq \frac{1}{2}$. Ideally, we would like to use $s_{v,t}$ to bound the holding cost incurred when serving (v,t) on day $s_{v,t}$. However, to avoid high routing costs, not all demands will get to be served by the desired $s_{v,t}$. Instead, for each client v, an appropriately chosen subset of $\{s_{v,t} : t \leq T\}$ will be selected to be the days that have service to v. To determine facility openings and client-facility connections, the idea is to pick balls that gather enough density of z_u values so that the cheapest facility within it can be paid for by the facility cost part of the LP objective. To be able to bound the routing cost, we would like to pick the radii of the balls based on the amount of y_s^{uv} values available from the LP solution. However, y_s^{uv} by itself does not give a good enough lower bound for z_u. So we will assign disjoint portions of y_s^{uv} to y_{st}^{uv} for different ts and ultimately use y_{st}^{uv} to bound the facility cost. Then the disjoint portions of y_s^{uv} will be used to pay for the routing cost. With these goals in mind, we now formally define the visit days and the radius for each client.

Fix a client v. The set A_v of demand days t that v gets visited on their $s_{v,t}$ will be assigned based on collecting enough y_{st}^{uv} over u and s. We call the days in A_v *anchors* of v. Denote by t_{L_v} the latest day that has positive demand at v. We use S_v to keep track of the service days for the anchors.

Algorithm 11 Visits for v

```
Initialize A_v ← {t_{L_v}}.
Initialize S_v ← {s_{v,t_{L_v}}}.
Denote by t̃ the earliest anchor in A_v.
while there is a positive demand at v on some before t̃ do
    Denote by t the latest day before t̃ with positive demand at (v,t).
    if t ≥ s_{v,t̃} then
        Serve (v,t) on day s_{v,t̃}.
    else
        Update A_v ← A_v ∪ {t}.
        Update S_v ← S_v ∪ {s_{v,t}}.
        Update t̃ ← t.
    end if
end while
Output the visit set S_v for v.
```

Define $W_{v,t} = \sum_{u\in V}\sum_{s=s_{v,t}}^{t} w_{uv} y_{st}^{uv}$. Let $W_v = \min_{t\in A_v} w_{v,t}$. Finally, define $B_v = \{u \in V : w_{uv} \leq 4W_v\}$, which is a ball of radius $4W_v$ centered at v. For ball B_v, let $F_v = \arg\min_{q\in B_v} f_q$. Simply, F_v is a location in B_v with the lowest facility cost. Now we are ready to state the algorithm for opening facilities in Algorithm 12.
Denote by $B_{v_1},\ldots,B_{v_l}$ the balls picked into $\mathcal{B}$ by Algorithm 12.

Algorithm 12 12-approximation for Uncapacitated SIRPFL

1: $\mathcal{B} \leftarrow \emptyset$
2: **while** there is any ball B_v disjoint from all balls in $\mathcal{B}$ **do**
3: Add to $\mathcal{B}$ the ball B_{v_i} of smallest radius
4: **end while**
5: Within each ball B_{v_i}, open a facility at F_{v_i}.
6: Assign each client v to the closest opened facility $u(v)$.
7: For each v, serve it on all days in A_v by building an edge from facility $u(v)$ to v per day $s \in A_v$.

Proposition 6.3.1. *The holding cost of the solution from the algorithm is at most* $2h(x, y, z)$.

Proof. For each demand point (v, t), we will charge a disjoint part of twice the x values in the LP solution to pay for the holding cost. In particular, to pay for the holding cost incurred by (v, t), we charge $\sum_{s=1}^{s_{v,t}} H^v_{s,t} x^v_{s,t}$ part of the LP solution. We consider two cases: $t \in A_v$ and $t \notin A_v$.

1. In this case, assume that $t \in A_v$. Then (v, t) is served on day $s_{v,t}$, and incurs a holding cost of $H^v_{s_{v,t},t}$. By definition of $s_{v,t}$, we have $\sum_{u \in V} \sum_{s=s_{v,t}+1}^{t} y^{uv}_{st} < \frac{1}{2}$. Then

$$\begin{aligned}
\sum_{s=1}^{s_{v,t}} x^v_{s,t} &\geq 1 - \sum_{s=s_{v,t}+1}^{t} x^v_{s,t} \\
&\geq 1 - \sum_{s=s_{v,t}+1}^{t} \sum_{u \in V} y^{uv}_{st} \\
&> 1 - \frac{1}{2} \\
&= \frac{1}{2}
\end{aligned}$$

   So our budget of $\sum_{s=1}^{s_{v,t}} H^v_{s,t} x^v_{s,t}$ is at least $H^v_{s_{v,t},t} \sum_{s=1}^{s_{v,t}} x^v_{s,t} \geq \frac{H^v_{s_{v,t},t}}{2}$.

2. In this case, assume that $t \notin A_v$. Let $\tilde{t}$ be the earliest anchor after t. Since t is not an anchor, $[s_{v,t}, t]$ must have overlapped $[s_{v,\tilde{t}}, \tilde{t}]$. So $s_{v,\tilde{t}} \leq t$. So (v, t) is served on $s_{v,\tilde{t}}$. By constraint 6.10, we have $s_{v,t} \leq s_{v,\tilde{t}}$. By monotonicity of holding cost, the holding cost incurred by serving (v, t) on $s_{v,\tilde{t}}$ is at most $H^v_{s_{v,t},t} \leq 2 \sum_{s=1}^{s_{v,t}} H^v_{s,t} x^v_{s,t}$.

□

Proposition 6.3.2. *The routing cost of the solution from the algorithm is at most* $12r(x, y, z)$.

Proof. In a similar manner, we will charge a disjoint portion of 12 times the y values in the LP solution to pay for the routing cost. Note that only anchors cause new visit days to be created in the algorithm. So consider a demand point (v, t) such that t is an anchor for v.

1. First, consider the case that $v \in \{v_1, \ldots, v_l\}$. Then the routing cost to connect (v, t) to the nearest opened facility is

$$\begin{aligned} W_{F_v,v} &\leq 4W_v \\ &\leq 4W_{v,t} \text{ by definition of } W_v \\ &\leq 4 \sum_{u \in V} \sum_{s=s_{v,t}}^{t} w_{uv} y_s^{uv} \text{ by constraint 6.8.} \end{aligned}$$

   Since it is within 4 times the LP budget, the desired claim holds.

2. Now, assume that $v \notin \{v_1, \ldots, v_l\}$. Then B_v overlaps $B_{v'}$ for some v' of smaller radius than B_v (otherwise B_v would have been chosen into $\mathcal{B}$ instead of the larger balls that overlap B_v). Then the edge built to serve (v, t) connects $F_{v'}$ to v. So the routing cost to serve (v, t) is

$$\begin{aligned} W_{F_{v'},v} &\leq W_{v,v'} + W_{v',F_{v'}} \\ &\leq 2 \cdot 4W_v + 4W_v \text{ since radius of } B_v \text{ is at least radius of } B_{v'} \\ &\leq 12W_v \\ &\leq 12W_{v,t} \\ &\leq 12 \sum_{u \in V} \sum_{s=s_{v,t}}^{t} w_{uv} y_s^{uv} \text{ by constraint 6.8.} \end{aligned}$$

Observe that for every v and any two anchors t_1, t_2 for v, we have $[s_{v,t_1}, t_1] \cap [s_{v,t_2}, t_2] = \emptyset$ by the construction of anchors. So each y_s^{uv} is charged at most once among all demands whose deadline correspond to anchors. □

Before bounding the facility costs, we show a Lemma that will help prove the desired bound.

Lemma 6.3.3. *For all $i \in \{1, \ldots, l\}$, we have $\sum_{v \in B_{v_i}} z_v \geq \frac{1}{4}$.*

Proof. Suppose there is some $i \in \{1, \ldots, l\}$ such that $\sum_{u \in B_{v_i}} z_u < \frac{1}{4}$. Let $\hat{t} = \arg\min_t W_{v_i,t}$. Then

$$
\begin{aligned}
W_{v_i} &= W_{v,\hat{t}} \\
&= \sum_{u \in V} \sum_{s=s_{v,\hat{t}}}^{\hat{t}} w_{uv} y_{s\hat{t}}^{uv} \\
&\geq \sum_{u \notin B_{v_i}} \sum_{s=s_{v,\hat{t}}}^{\hat{t}} w_{uv} y_{s\hat{t}}^{uv} \\
&\geq 4W_v \sum_{u \notin B_{v_i}} \sum_{s=s_{v,\hat{t}}}^{\hat{t}} y_{s\hat{t}}^{uv} \text{ by } u \notin B_{v_i} \\
&\geq 4W_v \Big(\sum_{u \in V} \sum_{s=s_{v,\hat{t}}}^{\hat{t}} y_{s\hat{t}}^{uv} - \sum_{u \in B_{v_i}} \sum_{s=s_{v,\hat{t}}}^{\hat{t}} y_{s\hat{t}}^{uv} \Big) \\
&\geq 4W_v \Big(\frac{1}{2} - \sum_{u \in B_{v_i}} \sum_{s=s_{v,\hat{t}}}^{\hat{t}} y_{s\hat{t}}^{uv} \Big) \text{ by definition of } s_{v,t} \\
&\geq 4W_v \Big(\frac{1}{2} - \sum_{u \in B_{v_i}} z_u \Big) \text{ by constraint 6.7} \\
&> W_v \text{ by the supposition that } \sum_{u \in B_{v_i}} z_u < \frac{1}{4}, \text{ which leads to a contradiction.}
\end{aligned}
$$

□

Proposition 6.3.4. *The facility cost of the algorithm's solution is at most* $4f(x, y, z)$.

Proof. We will charge four times the z values of the LP solution to pay for the facilities opened by the algorithm. Since the balls picked by Algorithm 12 are disjoint, we can pay for each facility opened using the LP value in its ball. Consider ball B_{v_i} picked by the algorithm and its cheapest facility F_{v_i}. Then the cost of opening F_{v_i} is at most f_v for all $v \in B_{v_i}$. So the facility cost for F_{v_i} is

$$
\begin{aligned}
f_{F_{v_i}} &\leq 4 \sum_{v \in B_{v_i}} z_v f_{F_{v_i}} \\
&\leq 4 \sum_{v \in B_{v_i}} z_v f_v.
\end{aligned}
$$

The first inequality follows from Lemma 6.3.3. The second inequality is due to F_{v_i} being the cheapest facility in the ball. □

Corollary 6.3.5. *Algorithm 12 is a 12-approximation for Uncapacitated SIRPFL.*

6.4 Conclusion

We studied the Uncapacitated, Capacitated Unsplittable, and Capacitated Splittable variants of IAP and provided a polynomial time algorithm, an NP-hardness proof, and a 3-approximation, respectively. For the Uncapacitated SIRPFL, we gave a 12-approximation by integrating rounding ideas for Facility Location and disjoint intervals charging scheme from our 3-approximation for Capacitated Splittable IAP. It remains open whether Capacitated Splittable IAP is NP-hard. It would also be

interesting to obtain a constant approximation for Capacitated Unsplittable IAP. Finally, are there constant approximations for Capacitated Unsplittable SIRPFL and Capacitated Splittable SIRPFL?

Chapter 7

Conclusion

We studied ranking problems and three types of routing problems involving scheduling deliveries. In the following, we summarize the results and open problems from each chapter.

Ranking In Chapter 2, we defined a new set of ranking problems motivated by ordering large sets of participants and tasks. We provided polynomial time algorithms for all except for the Unconstrained k-near Chain Editing problem, for which we showed NP-hardness and gave an $O(kn)$ additive approximation. It still remains open to obtain any approximation for the (both sides unconstrained) Chain Editing. Also, is there a constant approximation for the (both sides unconstrained) Chain Addition problem? Finally, is there an $o(kn)$-approximation for the Unconstrained k-near Chain Editing problem?

Inventory Routing Problem Chapter 3 provided constant approximations for IRP on line metrics. First, we proved a 5-approximation by rounding the LP. Second, we showed a 26-approximation by appropriately pruning a primal dual method that extends the wavelength idea of Levi et al. [67]. The primal dual phase by itself produced solutions that could be super-constant factor far from the optimal solution. In order to obtain a solution within constant factor of optimal cost, we sparsified the visits and extended carefully chosen visits based on a partitioning of the line at powers of 2 distances from the depot. The main open question is to improve the $O(\frac{\log T}{\log \log T})$-approximation for IRP on arbitrary metrics. As an intermediate step, it would also be interesting to obtain constant approximations for IRP on grids and planar graphs.

Chapter 4 studied IRP on general metrics from a computational perspective. Inspired by the Prize-Collecting Steiner Tree (PCST) reduction for periodic IRP [49], we designed three heuristics that all use PCST (in different ways) as an intermediate problem to ultimately find good IRP solutions. The local search heuristics reduced large neighborhood searches to PCST problems. The greedy heuristic found sets to cover demands by creating PCSTs with penalties that simulate holding costs of the demands to be covered. The primal dual heuristic solved PCSTs to determine the set of clients to visit at breakpoints of the dual growth. We tested the heuristics on instances generated at the same parameter settings as the benchmark instances of Archetti et al. [8]. The best heuristic among them was the local search with ADD operations, which solved within 1.08 factor of the optimal cost at three orders of magnitude the speed of Gurobi. Since previous computational literature have focused on the capacitated case of IRP, rather than the uncapacitated version studied here, a natural question is to design heuristics that perform as competitively on the Capacitated IRP.

Deadline Inventory Routing Problem In Chapter 5, we introduced the Deadline Inventory Routing Problem, which only has routing costs, but whose optimal solutions require carefully aggregating visits satisfying depletion times to minimize the

total routing cost over time. We gave a $\log(n)$-approximation by first finding solutions whose trees per day are nondecreasing and then showing that arbitrary trees can be converted to nondecreasing trees losing at most a $\log(n)$ factor. On the other hand, we provide a class of instances on which optimal solutions are $\log(n)$ times cheaper than nondecreasing solutions. Since the number of days T is incomparable to the number of clients n, we also proved a $\log(T)$-approximation by LP rounding. In the published version [21] of this Chapter with co-authors, the Min Max objective was considered, i.e. minimize the cost of the largest route among all the days. The results for the Min Max version are also logarithmic approximations for general metrics. Natural open questions are to obtain better than logarithmic factors for both minimizing the total cost and minimizing the highest cost, respectively.

Inventory Routing Problem with Facility Location Finally, Chapter 6 integrates the Inventory Routing Problem with Facility Location, which introduces the extra choice of where to open facilities in the beginning and which client-facility connections to make per day. First, we studied the special case that only one depot and one client are present, which we called the Inventory Access Problem (IAP). For the Uncapacitated IAP, we provided a simple dynamic program. For the Capacitated Unsplittable IAP, we gave an NP-hardness reduction from Number Partition. For the Capacitated Splittable IAP, we proved a 3-approximation by rounding the LP. For the more general Star Inventory Routing Problem with Facility Location, we gave a 12-approximation by combining rounding ideas from Facility Location and the visitation ideas from our 3-approximation for Capacitated Splittable IAP. It is open to determine whether Capacitated Splittable IAP is NP-hard. Despite the capacity constraints' similarity to Knapsack, the allowance of splitting up demands fractionally potentially differentiates it from Knapsack. Another interesting direction would be to approximate the Capacitate Unsplittable IAP, which requires more delicate coordination of packing demands into trips to make the best use of the capacity. Finally, constant approximations for the Capacitated Splittable SIRPFL and Capacitated Unsplittable SIRPFL remain open.

Bibliography

[1] A. Agrawal, P. Klein, and R. Ravi. "Cutting down on fill using nested dissection: provably good elimination orderings". In: *Graph Theory and Sparse Matrix Computation*. Springer, 1993, pp. 31–55.

[2] D. Aksen et al. "An adaptive large neighborhood search algorithm for a selective and periodic inventory routing problem". In: *European Journal of Operational Research* 239.2 (2014), pp. 413–426.

[3] Reid Andersen et al. "Trust-based recommendation systems: an axiomatic approach". In: *WWW*. ACM. 2008, pp. 199–208.

[4] R. Anholt et al. "An Inventory-Routing Problem with Pickups and Deliveries Arising in the Replenishment of Automated Teller Machines". In: *Transportation Science* 50.3 (2016), pp. 1077–1091.

[5] S. Anily and A. Federgruen. "One Warehouse Multiple Retailer Systems with Vehicle Routing Cost". In: *Mangement Science* 36.1 (1990), pp. 92–114.

[6] C. Archetti, N. Boland, and M. G. Speranza. "A Matheuristic for the Multi-vehicle Inventory Routing Problem". In: *INFORMS Journal on Computing* 29.3 (2017), pp. 377–387.

[7] C. Archetti et al. "A branch-and-cut algorithm for a vendor-managed inventory routing problem". In: *Transportation Science* 41.3 (2007), pp. 382–391.

[8] C. Archetti et al. "A Hybrid Heuristic for an Inventory Routing Problem". In: *INFORMS Journal on Computing* 24.1 (2012), pp. 101–116.

[9] E. Arkin, D. Joneja, and R. Roundy. "Computational complexity of uncapacitated multi-echelon production planning problems". In: *Operations Research Letters* 8 (1989), pp. 61–66.

[10] B. Aydin et al. "Crowdsourcing for multiple-choice question answering". In: *IAAI*. 2014, pp. 2946–2953.

[11] E. Balas and N. Simonetti. "Linear Time Dynamic-Programming Algorithms for New Classes of Restricted TSPs: A Computational Study". In: *INFORMS Journal on Computing* 13.1 (2000), pp. 56–75.

[12] J. Bard et al. "A branch and cut algorithm for the vrp with satellite facilities". In: *IIE Transactions* 30.9 (1998), pp. 821–834.

[13] J. Bard et al. "A decomposition approach to the inventory routing problem with satellite facilities". In: *Transportation Science* 32.2 (1998), pp. 189–203.

[14] Sanjoy Baruah and Joël Goossens. "Scheduling real-time tasks: Algorithms and complexity". In: *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. CRC Press, Boca Raton, 2003. Chap. 28.

[15] Sanjoy Baruah et al. "The complexity of periodic maintenance". In: *Proceedings of the International Computer Symposium*. 1990, pp. 315–320.

[16] D. P. Bertsekas. *Non-linear Programming*. Athena Scientific, 1999.

[17] M. Bienkowski et al. “Better Approximation Bounds for the Joint Replenishment Problem”. In: *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms*. 2014, pp. 42–54.

[18] M. Bienkowski et al. “Online control message aggregation in chain networks”. In: *Proceedings of the 13th Algorithms and Data Structures Symposium*. 2013, pp. 133–145.

[19] Ivan Bliznets et al. “Lower Bounds for the Parameterized Complexity of Minimum Fill-in and Other Completion Problems”. In: *SODA*. 2016, pp. 1132–1151.

[20] Vincenzo Bonifaci and Alberto Marchetti-Spaccamela. “Feasibility Analysis of Sporadic Real-Time Multiprocessor Task Systems”. In: *Algorithmica* 63.4 (2012), pp. 763–780.

[21] T. Bosman et al. “Approximation Algorithms for Replenishment Problems with Fixed Turnover Times”. In: *LATIN*. 2018, pp. 217–230.

[22] N. Buchbinder et al. “Online Make-to-order Joint Replenishment Model: Primal Dual Competitive Algorithms”. In: *Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms*. 2008, pp. 952–961.

[23] L. Burns et al. “Distribution Strategies that Minimize Transportation and Inventory Costs”. In: *Operations Research* 33.3 (1985), pp. 469–490.

[24] Chris Calabro et al. “The complexity of Unique k-SAT: An Isolation Lemma for k-CNFs”. In: *Journal of Computer and System Sciences* 74.3 (2008), pp. 386–393.

[25] A. Campbell and M. Savelsbergh. “Inventory Routing in Practice”. In: *The Vehicle Routing Problem*. SIAM Monographs on Discrete Mathematics and Applications, 2002.

[26] A. Campbell et al. “The Inventory Routing Problem”. In: *Fleet Management and Logistics*. Springer US, 1998.

[27] Yixin Cao and Dániel Marx. “Chordal Editing is Fixed-Parameter Tractable”. In: *Algorithmica* 75.1 (2016), pp. 118–137.

[28] Yixin Cao and R. B. Sandeep. “Minimum Fill-In: Inapproximability and Almost Tight Lower Bounds”. In: *CoRR* abs/1606.08141 (2016). URL: http://arxiv.org/abs/1606.08141.

[29] L. Chan, A. Fedgergruen, and D. Simchi-Levi. “Probabilistic Analyses and Practical Algorithms for Inventory-Routing Models”. In: *Operations Research* 46.1 (1998), pp. 96–106.

[30] Mee Yee Chan and Francis Y. L. Chin. “General schedulers for the pinwheel problem based on double-integer reduction”. In: *IEEE Transactions on Computers* 41.6 (1992), pp. 755–768.

[31] Mee Yee Chan and Francis Y. L. Chin. “Schedulers for larger classes of pinwheel instances”. In: *Algorithmica* 9.5 (1993), pp. 425–462.

[32] M. Cheung et al. “The submodular joint replenishment problem”. In: *Mathematical Programming* 158.1 (2016), pp. 207–233.

[33] T. Chien, A. Balakrishnan, and R. Wong. “An integrated inventory allocation and vehicle routing problem”. In: *Transportation Science* 23.2 (1989), pp. 67–76.

[34] Leandro C. Coelho, Jean-François Cordeau, and Gilbert Laporte. “Thirty years of inventory routing”. In: *Transportation Science* 48.1 (2013), pp. 1–19.

[35] Sofie Coene, Frits C. R. Spieksma, and Gerhard J. Woeginger. "Charlemagne's Challenge: The Periodic Latency Problem". In: *Operations Research* 59.3 (2011), pp. 674–683.

[36] Holger Dell et al. "Exponential Time Complexity of the Permanent and the Tutte Polynomial". In: *ACM Transactions on Algorithms* 10.4 (2014), 21:1–21:32.

[37] G. Desaulniers, J. Rakke, and L. Coelho. "A Branch-Price-and-Cut Algorithm for the Inventory-Routing Problem". In: *Transportation Science* 50.3 (2016), pp. 1060–1076.

[38] X. L. Dong, L. Berti-Equille, and D. Srivastava. "Integrating Conflicting Data: The Role of Source Dependence". In: *PVLDB* 2.1 (2009), pp. 550–561.

[39] Pål Grønås Drange et al. "On the Threshold of Intractability". In: *ESA*. 2015, pp. 411–423.

[40] M. Dror and M. Ball. "Inventory/routing: Reduction from an annual to short period problem". In: *Naval Research Logistics Quarterly* 34.6 (1987), pp. 891–905.

[41] M. Dror, M. Ball, and B. Golden. "Computational comparisons of algorithms for the inventory routing problem". In: *Annals of Operations Research* 4.1 (1985), pp. 3–23.

[42] Friedrich Eisenbrand et al. "Scheduling periodic tasks in a hard real-time environment". In: *Proceedings of the 37th International Colloquium on Automata, Languages, and Programming*. Springer. 2010, pp. 299–311.

[43] Pontus Ekberg and Wang Yi. "Schedulability analysis of a graph-based task model for mixed-criticality systems". In: *Real-Time Systems* 52.1 (2016), pp. 1–37. DOI: 10.1007/s11241-015-9225-0. URL: http://dx.doi.org/10.1007/s11241-015-9225-0.

[44] T. Feder, H. Mannila, and E. Terzi. "Approximating the Minimum Chain Completion Problem". In: *Inf. Process. Lett.* 109.17 (2009), pp. 980–985.

[45] A. Federgruen and P. Zipkin. "A combined vehicle routing and inventory allocation problem". In: *Operations Research* 32.5 (1984), pp. 1019–1037.

[46] M. Fischetti et al. "Thinning out Steiner trees: A node-based model for uniform edge costs". In: *Mathematical Programming Computation* 9.2 (2017), pp. 203–229.

[47] Peter C. Fishburn and Jeffrey C. Lagarias. "Pinwheel scheduling: Achievable densities". In: *Algorithmica* 34.1 (2002), pp. 14–38.

[48] Fedor V. Fomin and Yngve Villanger. "Subexponential Parameterized Algorithm for Minimum Fill-in". In: *SODA*. 2012, pp. 1737–1746.

[49] T. Fukunaga, A. Nikzad, and R. Ravi. "Deliver or hold: Approxmation algorithms for the periodic inventory routing problem". In: *Proceedings of the 17th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems*. 2014, pp. 209–225.

[50] Alban Galland et al. "Corroborating information from disagreeing views". In: *WSDM*. ACM. 2010, pp. 131–140.

[51] Leszek Gasieniec et al. "Bamboo Garden Trimming Problem". In: *SOFSEM*. Springer, 2017, pp. 229–240.

[52] Wolfgang Gatterbauer and Dan Suciu. "Data conflict resolution using trust mappings". In: *SIGMOD*. 2010, pp. 219–230.

[53] H. Glicksman and M. Penn. "Approximation algorithms for group prize-collecting and location-routing problems". In: *Discrete Applied Mathematics* 156.17 (2008), pp. 3238–3247.

[54] V. Goel et al. "Large neighborhood search for LNG inventory routing". In: *Journal of Heuristics* 18.6 (2012), 821–848.

[55] M. Goemans and D.P. Williamson. "A general approximation technique for constrained forest problems". In: *SIAM Journal on Computing* 24.2 (1995), pp. 296–317.

[56] B. Golden, A. Assad, and R. Dahl. "Analysis of large scale vehicle routing with an inventory component". In: *Large Scale Systems* 7.2-3 (1984), pp. 181–190.

[57] Manish Gupta and Jiawei Han. "Heterogeneous network-based trust analysis: a survey". In: *SIGKDD Explorations Newsletter* 13.1 (2011), pp. 54–71.

[58] T. Harks, F. König, and J. Matuschke. "Approximation Algorithms for Capacitated Location Routing". In: *Transportation Science* 47.1 (2013), pp. 3–21.

[59] Robert Holte et al. "The pinwheel: A real-time scheduling problem". In: *Proceedings of the 22th Annual Hawaii International Conference on System Sciences*. Vol. 2. 1989, pp. 693–702.

[60] Tobias Jacobs and Salvatore Longo. "A new perspective on the windows scheduling problem". In: *arXiv preprint arXiv:1410.7237* (2014).

[61] P. Jaillet et al. "A rolling horizon framework for the inventory routing problem". 1997.

[62] Y. Jiao, R. Ravi, and W. Gatterbauer. "Algorithms for Automatic Ranking of Participants and Tasks in an Anonymized Contest". In: *WALCOM*. 2017, pp. 335–346.

[63] Haim Kaplan, Ron Shamir, and Robert E. Tarjan. "Tractability of Parameterized Completion Problems on Chordal, Strongly Chordal, and Proper Interval Graphs". In: *SIAM J. Comput.* 28.5 (1999), pp. 1906–1922.

[64] P. N. Klein and R. Ravi. "A nearly best-possible approximation algorithm for node-weighted Steiner trees". In: *J. Algorithms* 19.1 (1995), pp. 104–114.

[65] Jon M Kleinberg. "Authoritative sources in a hyperlinked environment". In: *JACM* 46.5 (1999), pp. 604–632.

[66] A. Kleywegt, V. Nori, and M. Savelsbergh. "The Stochastic Inventory Routing Problem with Direct Deliveries". In: *Transportation Science* 36.1 (2003), pp. 94–118.

[67] R. Levi, R. Roundy, and D. Shmoys. "Primal-Dual Algorithms for Deterministic Inventory Problems". In: *Mathematics of Operations Research* 31.2 (2006), pp. 267–284.

[68] R. Levi et al. "First Constant Approximation Algorithm for the One-Warehouse Multi-Retailer Problem". In: *Management Science* 54.4 (2008), pp. 763–776.

[69] Q. Li et al. "Resolving conflicts in heterogeneous data by truth discovery and source reliability estimation". In: *SIGMOD*. 2014, pp. 1187–1198.

[70] Y. Li et al. "A Survey on Truth Discovery". In: *SIGKDD Explorations Newsletter* 17.2 (2015), pp. 1–16.

[71] I. Ljubić et al. "An algorithmic framework for the exact solution of the prize-collecting Steiner tree problem". In: *Mathematical Programming* 105.2-3 (2006), pp. 427–449.

[72] Al Mok et al. "Algorithms and complexity of the periodic maintenance problem". In: *Microprocessing and Microprogramming* 27.1-5 (1989), pp. 657–664.

[73] V. Nagarajan and C. Shi. "Approximation algorithms for inventory problems with submodular or routing costs". In: *Mathematical Programming* (2016), pp. 1–20.

[74] A. Natanzon, R. Shamir, and R. Sharan. "A Polynomial Approximation Algorithm for the Minimum Fill-In Problem". In: *SIAM J. Comput.* 30.4 (2000), pp. 1067–1079.

[75] T. Nonner and A. Souza. "Approximating the joint replenishment problem with deadlines". In: *Discrete Math., Alg. and Appl.* 1.2 (2009), pp. 153–174.

[76] J. Pasternack and D. Roth. "Knowing What to Believe (when you already know something)". In: *COLING*. 2010, pp. 877–885.

[77] J. Pasternack and D. Roth. "Latent Credibility Analysis". In: *WWW*. 2013, pp. 1009–1021.

[78] Jeffrey Pasternack, Dan Roth, and V.G. Vinod Vydiswaran. *Information Trustworthiness*. AAAI Tutorial. 2013.

[79] R. Ravi and A. Sinha. "Approximation algorithms for problems combining facility location and network design". In: *Operations Research* 54.1 (2006), pp. 73–81.

[80] A. M. Sarmiento and R. Nagi. "A review of integrated analysis of production–distribution systems". In: *IIE Transactions* 31 (1999), pp. 1061–1074.

[81] V. A. Shirokikh and V. V. Zakharov. "Dynamic Adaptive Large Neighborhood Search for Inventory Routing Problem". In: *Modelling, Computation and Optimization in Information Systems and Management Sciences* 359 (2015), pp. 231–241.

[82] Yu (Ledell) Wu et al. "Inapproximability of Treewidth, One-shot Pebbling, and Related Layout Problems". In: *J. Artif. Int. Res.* 49.1 (2014), pp. 569–600.

[83] M. Yannakakis. "Computing the Minimum Fill-In is NP-Complete". In: *SIAM Journal on the Algebraic and Discrete Methods* 2.1 (1981), pp. 77–79.

[84] Xiaoxin Yin, Jiawei Han, and Philip S Yu. "Truth discovery with multiple conflicting information providers on the web". In: *TKDE* 20.6 (2008), pp. 796–808.