2002

Offline and online variants of the Traveling Salesman Problem

John Ebenezer Augustine
Louisiana State University and Agricultural and Mechanical College

OFFLINE AND ONLINE VARIANTS OF THE TRAVELING SALESMAN PROBLEM

A Thesis

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Master of Science in Electrical Engineering

in

The Department of Electrical and Computer Engineering

by
John Ebenezer Augustine
B.E. in Computer Science and Engineering., University of Madras, 1998
M.S. in Systems Science, Louisiana State University, 2001
December 2002

In memory of Dr. Steven Seiden

Acknowledgments

With the sense of accomplishment comes the need for gratitude and no one is more worthy of thanks than the God of the Bible. Spiritual experiences are often belittled in academia as merely figments of one's imagination. But, I would like to acknowledge God's sovereign direction in my life as the ultimate source of my strength. I am especially thankful to God for the many people who have been an awesome source of inspiration and encouragement in this endeavor. It is with great pleasure that I thank those people who have been supportive of my efforts.

I am greatly indebted to Dr. Steven Seiden, who introduced me to theoretical computer science. I could always count on Steve to answer all my questions — yes, even the stupid ones. He never tired of traveling the extra mile. I am grateful to Steve for co-advising me in this thesis, although he never knew that.

Of the many courses that have influenced my thinking, Dr. Jaganathan Ramanujam's courses stand apart. His take-home exams were a great challenge that inspired me. I am very thankful to Dr. Ram for co-advising me in this thesis and working into the wee hours providing me with some excellent comments.

The hardest courses often bring out the best in you. Without doubt, Dr. Jerry Trahan's course on algorithms gave me the foundation that sustains me today. His questions were always aimed at discovering the several blindspots in my thinking. I am glad that I think better today. I am also very grateful to Dr. R. Vaidyanathan for accepting my request to serve in my committee, given the unusual circumstances.

I am thankful to Dr. Kenny Rose of the Coastal Fisheries Institite, Department of Oceanography and Coastal Sciences for employing me as a computer modeler. He has been patient with me on several occassions when I needed some time to work on my thesis. I also thank my colleagues Shaye Sable, Aaron Adamack, Laura Althauser and Cheryl Murphy, who have made my work a fulfilling experience. I am thankful to Enrique Reyes for giving me the opportunity to play with the DEC-alpha machines.

My parents, though physically far away, have been with me all along in the spirit. Since the time I was born, they have nurtured me. I can never repay them for their sacrifices. My sisters and their families have been a continued source of encouragement. One of the greatest joys in life is to assume the role of an uncle and see the family grow.

Although I never had brothers, I have had several close friends who stuck close to me. I am thankful to Bradley for being a great pastor and friend, Jarrett, Donavan, Issarael, Tim, Steven, Mary, David Mukai, David Sathiaraj, Clay, Caleb, and the list goes on. Also, special thanks to my many friends in years past, whose lives still influence me. Of significant note are Prem Kumar, Prashan, Stephen, Enoch, Jell Singh, Solomon, and again, the list goes on.

Table of Contents

List of Figures

Abstract

In this thesis, we study several well-motivated variants of the Traveling Salesman Problem (TSP). First, we consider makespan minimization for vehicle scheduling problems on trees with release and handling times. 2-approximation algorithms were known for several variants of the single vehicle problem on a path. A $3/2$-approximation algorithm was known for the single vehicle problem on a path where there is a fixed starting point and the vehicle must return to the starting point upon completion. Karuno, Nagamochi and Ibaraki give a 2-approximation algorithm for the single vehicle problem on trees. We develop a Polynomial Time Approximation Scheme (PTAS) for the single vehicle scheduling problem on trees which have a constant number of leaves. This PTAS can be easily adapted to accommodate various starting/ending constraints. We then extended this to a PTAS for the multiple vehicle problem where vehicles operate in disjoint subtrees. We also present competitive online algorithms for some single vehicle scheduling problems.

Secondly, we study a class of problems called the Online Packet TSP Class (OP-TSP-CLASS). It is based on the online TSP with a packet of requests known and available for scheduling at any given time. We provide a $5/3$ lower bound on any online algorithm for problems in OP-TSP-CLASS. We extend this result to the related *k*-reordering problem for which a $3/2$ lower bound was known. We develop a $\kappa+1$-competitive algorithm for problems in OP-TSP-CLASS, where a κ-approximation algorithm is known for the offline version of that problem. We use this result to develop an offline $m(\kappa+1)$-

approximation algorithm for the Precedence-Constrained TSP (PCTSP) by segmenting the n requests into m packets. Its running time is $mf(n/m)$ given a κ-approximation algorithm for the offline version whose running time is $f(n)$.

1. Introduction

The Traveling Salesman Problem (TSP) is one of the most intriguing problems faced by researchers. It is easy to state, but hard to solve. Its decision version (term defined in Section 1.2) is a classic example of an NP-complete problem first studied by Karp in [25]. This thesis is concerned with variants of TSP that find practical application in several areas including low power computing, automated guided vehicles scheduling and routing ships on shorelines.

In the traditional definition of TSP, we are provided with *n cities* and a cost matrix C, where C_{ij} is the cost of traveling from city i to city j. The goal is to minimize the total cost of traveling to every city exactly once and returning to the starting city. The cities are typically represented by means of a graph but since most applications of the TSP allow for the triangle inequality to hold, metric spaces are also commonly studied. A metric space, often denoted $\mathcal{M}$ is a set of points P and a distance function $d : P \times P \rightarrow R^{+}$, where R^{+} denotes the set of non-negative reals, such that for any x, y and z belonging to P,

$$\begin{aligned} d(x,x) &= 0, \\ d(x,y) &= d(y,x), \text{ and} \\ d(x,y)+d(y,z) &\geq d(x,z) \text{ (triangle inequality).} \end{aligned}$$

We use the metric space representation for the problems in this thesis.

1.1 A Brief History

The precise origins of TSP are unclear. It has been studied in the first half of the twentieth century by several people world over primarily for agricultural reasons [34, 23]. An integer programming based solution to the TSP was provided by Dantzig, Fulkerson and Johnson in 1954 [14]. They solved a TSP instance with 49 cities, an impressive achievement for the fifties. Even before the theory of computational intractability and NP-completeness was established, researchers started looking for sub-optimal solutions. One the first heuristics was shown by Lin in [33]. In 1972, Karp showed the decision version of the TSP to be NP-complete along with several other problems [25]. Papadimitriou [36] showed in 1977 that even the Euclidean TSP is NP-hard. Quite recently, it has been shown that a polynomial time ρ-approximation algorithm is only possible for $\rho \geq 203/202$, unless P=NP [17]. A polynomial time $3/2$-approximation algorithm was developed by Christofides for the metric TSP in 1976. This positive theoretical result was not improved significantly for about two decades. In the mid-nineties, Arora provided a Polynomial Time Approximation Scheme (PTAS) for the Euclidean TSP [1]. In the meantime, several heuristic and metaheuristic approaches have been studied including genetic algorithms, tabu search, memetic algorithms, etc.

There are several variants of the TSP and the reader is referred to an excellent survey by Lawler, Lenstra, Rinnooy-Kan and Shmoys [31]. The Vehicle Scheduling Problem and the Precedence-Constrained TSP are two important problems that are studied in this thesis and we have provided their formulations at the appropriate sections. All the problems studied in this thesis have the TSP as a special case. Therefore, they are atleast as hard as the TSP.

1.2 Background

In this section, we endeavor to provide a brief explanation of several terms and concepts that we use in this thesis. A *problem* is a generalized question [19]. By "generalized," we mean that the problem may possess several free variables implying that the solution to the problem has to work for all possible values that those variables can take. Further, *optimization problems* either require the minimization or maximization of the cost or profit, which is provided by the objective function as part of the problem formulation. A decision problems are those problems that have an answer of either a "yes" or "no." Optimization problems can often be mapped onto corresponding decision problems simply by adding a threshold value and asking whether the cost (or profit) be below (or above) the threshold value." An *algorithm* is a series of steps that can be followed to solve a problem. Sometimes, perfect solutions are not achievable in practice. Several optimization problems fall into the category of problems that are more easily approximated than solved perfectly. Thus algorithm designers have resorted to "approximation algorithms," which are algorithms that can guarantee a solution that is within a factor, say k, of the perfect solution. More formally, for a problem $\mathcal{P}$, an algorithm $\mathcal{A}$ is a ρ-approximation algorithm if

$$\text{cost}_{\mathcal{A}}(\mathcal{P}) \leq \rho\, \text{cost}(\mathcal{P}),$$

where $\text{cost}_{\mathcal{A}}(\mathcal{P})$ is the cost of the solution provided by $\mathcal{A}$ for $\mathcal{P}$ and $\text{cost}(\mathcal{P})$ is the cost of the optimal solution to $\mathcal{P}$. The reader is referred to [21] for a more detailed treatment of the subject. There are several metrics that could be used to evaluate the quality of an algorithm. The obvious metric for approximation algorithms is the factor k. The other commonly used metric deals with the abstracted notion of time that the algorithm takes to solve a problem, referred to as its *time complexity function*. Informally, the time complexity function relates the size of the input to the time taken by the algorithm

to solve the problem. We often use the big-O notation for time complexity and other functions. Given functions $f(n)$ and $g(n)$ such that $f(n) \leq cg(n)$, where c is a constant, we say that $f(n)$ is $O(g(n))$. The reader is referred to an excellent introductory book by Cormen *et al.* [13] for a more detailed overview. The other commonly used metric is the memory space needed for an algorithm. We do not use this metric to evaluate our algorithms.

Problems can be classified on the basis of the extent of knowledge of the parameters that is available to the algorithm as it progresses in solving the problem. Traditionally, problems are formulated such that all parameters are known in advance and are called *offline* problems. They are commonly solved by a three step algorithm that reads the input, processes it and provides the output. However, we often encounter real life problems, where the parameters are not completely known in advance. For example, consider a stock market scenario. Let us assume that we will trade for one year. The question that is asked each day is whether to buy or sell. The answer to that question is easy if we know the future stock prices. Since we do not know the future, we resort to using past data and current market information in solving the problem. Also, note that the solution is progressive, i.e., the algorithm makes a series of decisions. This series of decisions that we make will have an impact on the overall quality of the solution that is obtained when the problem input is completely known and the final decision is made. In our example, our gain (or loss) at the end of the year is the result of individual decisions that we make each day. We use competitive analysis, first suggested by Sleator and Tarjan in [38], to evaluate online algorithms provided in this thesis. Consider an online problem $\mathcal{P}$ with a legal input sequence I. The corresponding offline problem, where I is known completely in advance have an optimal cost $OPT(I)$. Let $\mathcal{A}$ be an algorithm for $\mathcal{P}$ with cost $ALG(I)$ for an input sequence I. $\mathcal{A}$ is said to be c-competitive if there exists

a constant α such that for all input sequences I,

$$ALG(I) \leq c \cdot OPT(I) + \alpha.$$

The reader is referred to the book by Borodin and El-Yaniv [6] for a more comprehensive treatment of the subject of competitive analysis. The backgrounds of the problems considered in this thesis are provided in the appropriate chapters.

1.3 Problem Formulation

The TSP is a special case and basis for several practically occurring optimization problems in routing and scheduling, which is a very significant part of the work in operations research. The implication of TSP being a special case is that the problems we deal with are at least as hard as the TSP. We start this section by providing a motivational example that captures the usefulness of many problems discussed in this thesis. We then provide formulations for all the problems that we discuss in this thesis. Note that this list is provided with an intent to provide a single reference location for problem formulations. The notations provided in the sections dealing with a problem should take precedence over the notations provided in this list.

1.3.1 Motivating Example

Consider a company that requires its service people to travel by the public transportation system. Consider the city of Stockholm in Sweden. A map of the city metro transportation system [39] is shown in Figure 1.1 on page 8. We also provide a simplified schematic in Figure 1.2 on page 9 that conforms to the tree structure. We will use this schematic in the rest of this example. The service people are assumed to have passes. Service calls are placed in different parts of the city. We have an estimate of the time it requires for the service person to walk from the nearest station to the service location, serve the request and return to the station, which in total we call the service time or han-

dling time. We assume that the frequency of trains is good enough to ensure a constant travel time between any two stations. Having laid this basic framework, we will discuss several scenarios that lead to the problems discussed in this thesis.

In the first scenario, we have n service people. They all start at the same location, say the Centralstation at 8:00 AM. The service calls have been received and all service time estimates are known in advance. Further, each request has an associated release time requiring that service start only after the elapse of the release time. A sample of requests is provided in Table 1.1 on page 7. The service people are required to service all the requests and regroup at the Centralstation at the end of the day. The problem now is to partition the set of service requests between the n service people and provide a schedule for each person such that they can all come back to the Centralstation as soon as possible. The time elapsed from 8:00 AM till the last person gets back to the Centralstation corresponds to the *makespan* (defined in Chapter 2). Several variants of this can also be considered. One such variant is that the service people work from their homes and are not required to regroup anywhere. Thus, each person has an individual but fixed starting location and is required to return to that location. Another variant might require that each person be assigned a zone to work within. The zones can be restricted to not overlap with each other.

In the second scenario, we have a single service person who receives service requests in sets known as packets. At any time, he can handle requests from only one packet. As soon as he finishes serving all requests in one packet, he gets the subsequent packet. He does not have control over the packet size or its contents (such as location, release and service time of each job). He has no knowledge of future packets. The problem is to come up with the schedule that he has to follow in order to minimize the completion time. A sample list of jobs has been provided in Table 1.2 on page 7. Note that each packet is independent of each other. The release time, for instance, is provided with

Table 1.1: Sample job requests in the first scenario

Job ID	Location	Handling Time (in minutes)	Release Time (in minutes after 8:00 AM)
1	Karlaplan	15	0
2	Blakeberg	20	30
3	Hallonbergen	18	75
4	Telefonplan	22	40
5	Bagarmossen	32	82
6	Skogskyrkogården	39	150
7	Högdalen	28	120
⋮	⋮	⋮	⋮

Table 1.2: Sample job requests in the second scenario

Packet ID	Job ID	Location	Handling Time (in minutes)	Release Time (in minutes after start of packet)
1	1	Karlaplan	15	0
	2	Blakeberg	20	30
	3	Hallonbergen	18	75
2	1	Skogskyrkogården	39	10
	2	Högdalen	28	20
	3	Bergshamra	17	24
3	1	Sockenplan	23	1
	2	Västertorp	13	5
	3	Stadion	24	9
⋮	⋮	⋮	⋮	⋮

respect to the start of the packet. A special case of this scenario is when the release and handling times are restricted to be zero. In this case, each packet can be treated like a classic TSP problem. How does the service person schedule these packets of job requests so that he can complete all his packets at the earliest? The algorithms that we have provided in this thesis can be used to solve these problems and several other variants that arise in practice.

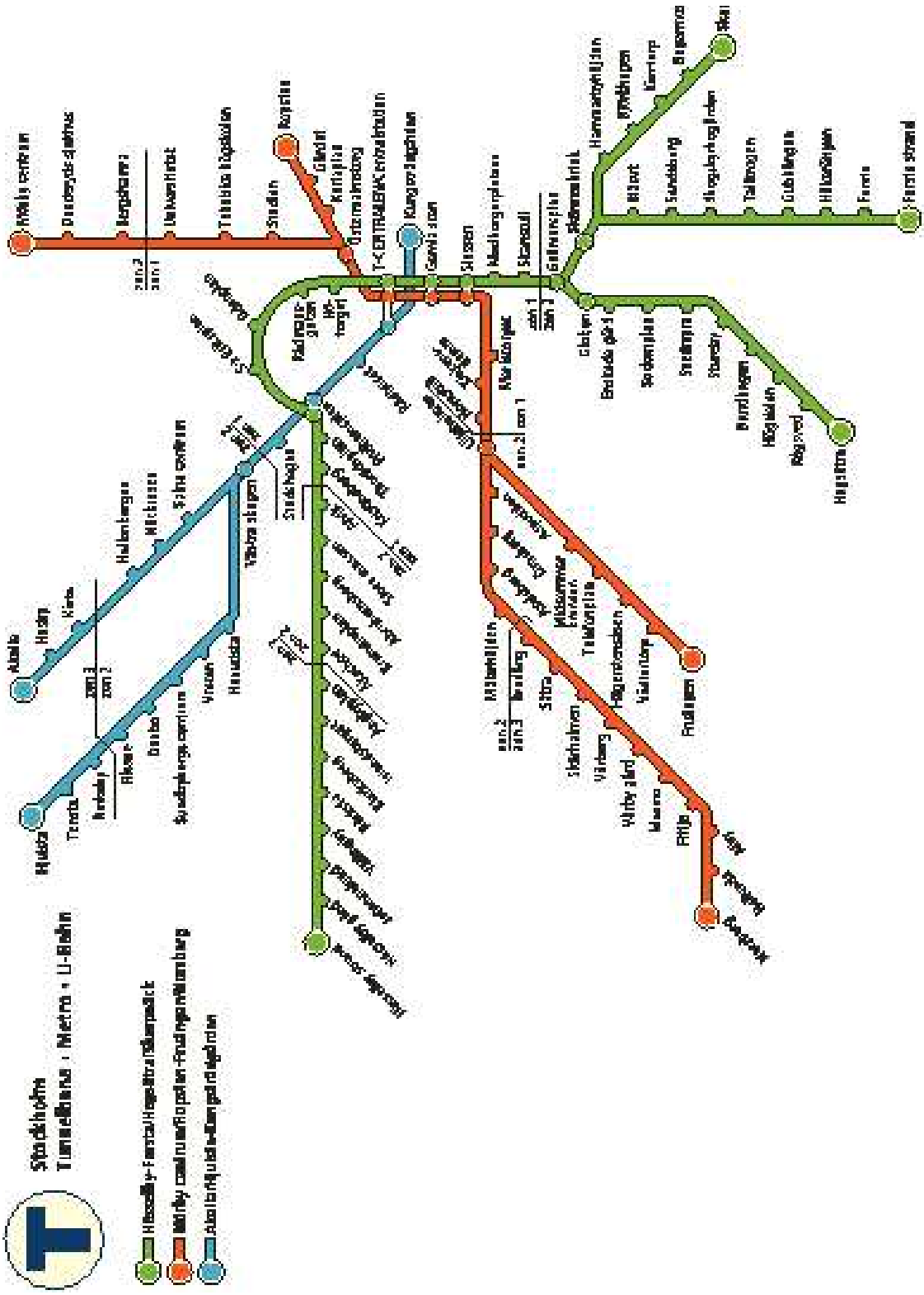

Figure 1.1: Map of Stockholm Tunnelbana used with permission. Please refer Appendix for details

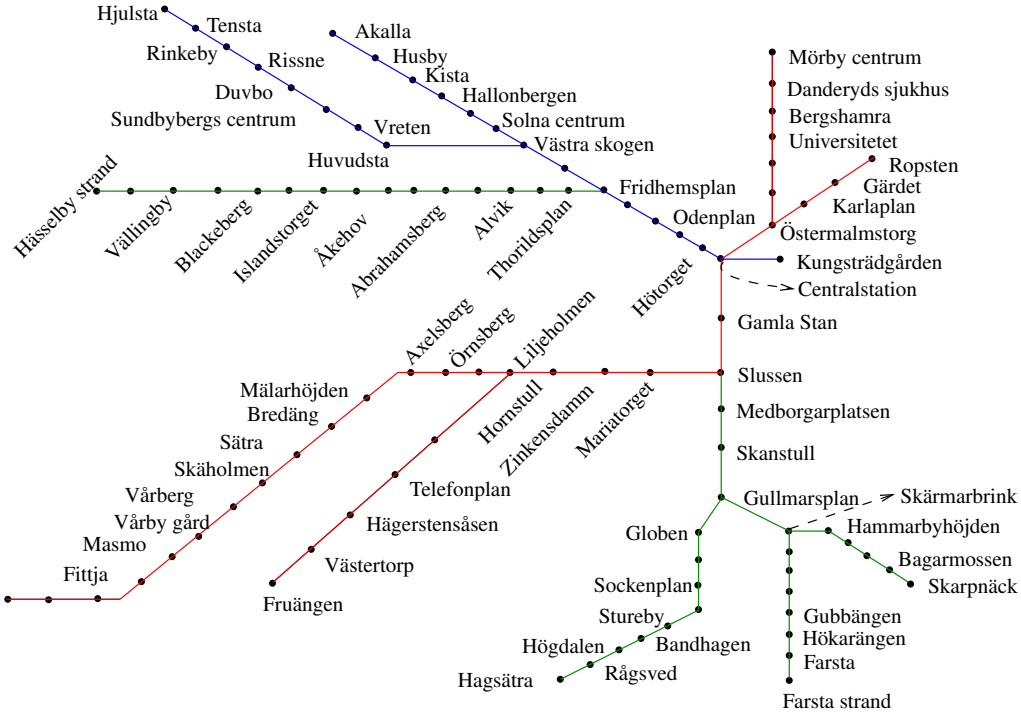

Figure 1.2: Schematic map of Stockholm Tunnelbana

1.3.2 Multiple Vehicle Scheduling Problem (MVSP)

Instance: A metric space $\mathcal{M}$ with distance $d(x,y)$ between any two points x and y in $\mathcal{M}$. A set J of *job requests* with n elements. In this thesis, we use the terms "request" and "job" interchangeably. For each $i \in J$, we have a release time $r_i \geq 0$, a handling time $h_i \geq 0$ and a position $p_i \in \mathcal{M}$. The number of *vehicles* is k.

Explanation: Intuitively, the problem consists of k vehicles that can traverse the metric space in question. We are required to provide k schedules for serving jobs each pertaining to one vehicle such that each job is served exactly once collectively. Although several objectives can be set up, we consider the cost in terms of time and the goal is to minimize the completion time of the last job served. The vehicle can spend time in three ways. Firstly, it could be traveling from one job request to another. Secondly, after reaching the position, it may be required to wait till the job is released and finally, it will have to spend time *handling* or *serving* the request. Note that we consider only the special case where $\mathcal{M}$ is a tree. Unless otherwise mentioned, $\mathcal{M}$ has t leaves and m vertices that are both considered to be constant.

A *valid schedule* s is a permutation of J denoted by Q_s and a list of $k+1$ integers u_i for $0 \leq i \leq k$ ordered by i such that $0 = u_0 < u_1 \leq u_2 \leq \cdots \leq u_k = n$. The i^{th} element of Q_s is denoted by $Q_s(i)$. Intuitively, the segment of the permutation ranging from u_{i-1} to u_i is the schedule of jobs served by the i^{th} vehicle. Let S be the set of all valid schedules. For any $s \in S$, and integers i, j such that $1 \leq i \leq j \leq n$, let the single vehicle cost for scheduling job requests i to j of the valid schedule s be given by

$$c(s,i,j) = \begin{cases} 0 & \text{if } i = j \\ \max \begin{pmatrix} r_{Q_s(j)}\,, \\ c(s,i,j-1) + \\ d(Q_s(j-1), Q_s(j)) \end{pmatrix} + h_{Q_s(j)} & \text{otherwise} \end{cases} \tag{1.1}$$

Now, the cost of a valid schedule is given by $C(s) = \max_{1 \leq i \leq k} c(s, u_{i-1}+1, u_i)$.

Goal: Find the valid schedule $s \in S$ that has the lowest cost $C(s)$.

1.3.3 Single Vehicle Scheduling Problem (SVSP)

It is a special case of the MVSP with the constraint that $k = 1$, that is, we have a single vehicle that serves all $j \in J$.

1.3.4 Fixed Origin MVSP (FO)

Instance: It is the same as MVSP with the following addition. Each vehicle i, where $1 \leq i \leq k$, is associated with an *origin* $O(i) \in \mathcal{M}$.

Explanation: The vehicle i is expected to start at $O(i)$. Hence, we add the cost of traveling from $O(i)$ to the first request served by i. Therefore, the new cost is given by

$$C^{FO} = \max_{1 \leq i \leq k} \{c(s, u_{i-1}+1, u_i) + d(O(i), Q_s(u_{i-1}+1))\}. \tag{1.2}$$

Goal: find the valid schedule $s \in S$ that has the lowest cost $C^{FO}(s)$.

1.3.5 Return to Origin MVSP (RTO)

Instance: This is the same as that of fixed Origin MVSP.

Explanation: The difference between the FO and RTO is that the vehicle is expected to return to the origin in the RTO version. This requires the cost to be altered to

$$C^{RTO} = \max_{1 \leq i \leq k} \{c(s, u_{i-1}+1, u_i) + d(O(i), Q_s(u_{i-1})+1) + d(Q_s(u_i), O(i))\}. \tag{1.3}$$

Goal: find the valid schedule $s \in S$ that has the lowest cost $C^{RTO}(s)$.

1.3.6 Zone MVSP (ZMVSP)

Instance: It is the same as MVSP. For the purposes of this thesis, the metric space $\mathcal{M}$ is restricted to be a tree.

Explanation: The *zone* Z_i of a vehicle i is the minimal subtree that includes the posi-

tions of job requests $Q_s(j)$, for $j = u_{i-1}$ to u_i. Each vehicle is required to stay within a zone that does not overlap with the zones of other vehicles. The algorithm, however, has the freedom to define the zones for the vehicles. We use $x \rightsquigarrow y$ to denote the set of points in $\mathcal{M}$ in the path from x to y, where x and y are in $\mathcal{M}$. Mathematically expressed, the zone constraint says that any point $p \in x \rightsquigarrow y$, where $x \in Z_i$ and $y \in Z_i$ implies that $p \in Z_i$ and for some $i' \neq i$ and $1 \leq i' \leq k$, $p \notin Z_{i'}$. This reduces the number of valid schedules.

Goal: The same as that of MVSP.

1.3.7 Online Packet TSP (OP-TSP)

Instance: This is an online problem. We have a set J of job requests and a partition on J with m elements. Each subset of J in the partition is called a *packet* and the packets are denoted by $\Psi(i)$ where $1 \leq i \leq m$. Each $j \in J$ is located in some position in $\mathcal{M}$. For this problem, we assume that each job in a given packet is located in a distinct position. So, we do not introduce a notation for its position. The first packet is released at time 0. The subsequent packets are released as soon as the current packet is scheduled. A single server is required to serve the requests as they are released in packets. Intuitively, the cost is calculated in terms of the time it takes for the server to complete all the requests. The details on the calculation of cost and the objective to be minimized are provided in Section 3.1.1 on page 42.

1.3.8 Precedence-Constrained TSP (PCTSP)

Instance: The PCTSP is defined on a metric space $\mathcal{M}$ with distance matrix d giving the distance between any two points x and y in $\mathcal{M}$. We are also provided set J of *job requests* with n elements. Each job $j \in J$ has a position p_j in $\mathcal{M}$. The jobs in J are also constrained by precedence rules that are modeled using a directed acyclic graph (DAG) whose vertices are jobs and directed edges are the precedence rules. A valid schedule

$\pi(j)$ is a permutation on J that is also a topological sort of the DAG. The cost is given by $C_\pi = \sum_{1 \leq j \leq n-1} d(\pi(j), \pi(j+1))$.

Goal: Find the valid schedule π that minimizes C_π.

1.4 Notation

Chapters 2 and 2.6 are self contained. All notations have been independently defined in both chapters. In this section, we provide listings of notations for the two chapters. Please note that the notations are provided for reference purpose. Therefore, we have used terms that have not yet been defined.

1.4.1 Notation for Chapter 1

$\mathcal{M}$ — Metric Space

n — Number of job requests

i, j — Job request identifiers

r_j — Release time of job j

h_j — Handling time of job j

p_j — Position of job j

$d(x,y)$ — distance from point x to point y in $\mathcal{M}$

k — Number of vehicles

m — Number of vertices in $\mathcal{M}$

t — Number of leaves in $\mathcal{M}$

σ — Problem instance

$(1+\varepsilon)$ — Approximation factor, where $\varepsilon \geq 0$ is user defined

R — Number of release times

π — A valid schedule

$a_\pi^\sigma(i)$ — Arrival time of vehicle at the i^{th} request according to π in problem instance σ

$c_\pi^\sigma(i)$ — Similarly, the completion time

ϖ — An eager schedule. Refer to Section 2.2
for definition of eager schedule.

X_i^j — j^{th} leaf visited by vehicle in phase i

r_{max} — $\max_{1\le i\le n} r_i$

P — Sum of all edge weights

a — $2\lceil 1/\varepsilon \rceil$

b — $2(t+1)a^2$

δ — r_{max}/a. Intuitively, this is the temporal grid distance

Δ — P/b. This is the spatial grid distance

$\mathcal{N}$ — Modified version of $\mathcal{M}$. Refer to Section 2.3

ϕ — Mapping of points from $\mathcal{M}$ to $\mathcal{N}$

$\sigma^\downarrow$ — Rounded down version of σ in $\mathcal{N}$

$\sigma^\uparrow$ — Rounded up version of σ in $\mathcal{N}$

σ^* — Rounded down version of σ in $\mathcal{M}$

$r_i^\downarrow$, $h_i^\downarrow$ and $p_i^\downarrow$— Release time, handling time and position of job i in $\sigma^\downarrow$

$r_i^\uparrow$, $h_i^\uparrow$ and $p_i^\uparrow$— Release time, handling time and position of job i in $\sigma^\uparrow$

r_i^*, h_i^* and p_i^*— Release time, handling time and position of job i in σ^*

$C^*(i,j)$ — Optimal cost of serving requests $i,\ldots,j$,
where $1 \le i \le j \le n$

$x^*(i,\ell)$ — Optimal zone schedule for serving requests $1,\ldots,i$
with ℓ vehicles

$\mathcal{A}$ — A ρ-approximation algorithm for SVSP

$C(i,j)$ — Cost of serving requests $i,\ldots,j$ using $\mathcal{A}$,
where $1 \le i \le j \le n$

$x(i,\ell)$ — Zone schedule for serving requests $1,\ldots,i$

with ℓ vehicles using $\mathcal{A}$

D — Global deadline for serving all requests

d_i — Deadline for handling job i

1.4.2 Notation for Chapter 3

$\mathcal{M}$ — Metric Space

$d(x,y)$ — distance from point x to point y in $\mathcal{M}$

m — Number of packets

n_i — Number of requests in the i^{th} packet

j^i_ℓ — The ℓ^{th} job in the i^{th} packet

$\Psi(i)$ — The set of requests in the i^{th} packet

$\mathcal{X}(i)$ — The set of all permutations on $\Psi(i)$

π_i — An element of $\mathcal{X}(i)$

Π — A valid schedule

$C_{opt}(o,\Psi(i))$ — Optimal cost of serving requests in $\Psi(i)$
starting from $o \in \Psi(i-1)$

*PacketLB*1
and — Packet level lower bounds
*PacketLB*2

t^π_i — Time to serve requests in $\Psi(i)$
starting from last request served in $\Psi(i-1)$
according to π_i

T_Π — Total time taken by the valid schedule Π

$\mathcal{F}$ — A κ-approximation algorithm for the offline problem

*LB*1 and *LB*2 — Lower bounds on T_Π

1.5 Some Properties of Trees

Property 1.1 *For a tree with t leaves, the number of vertices with degree greater than 2, denoted k, is at most $t-2$.*

Proof We provide a proof by induction for this property. For the base case, consider the tree with exactly two leaves. The tree has $t = 2$ and $k = 0 \leq t-2$. Define a branch to be a path of maximal length from a leaf such that the vertices in the path apart from the leaf are of degree two. Now, assume that the property holds for all trees that have $t = i$, where $i \geq 2$. We are interested in trees with $t = i+1$. Such a tree, say A_{i+1}, can be obtained by taking some appropriate tree A_i with $t = i$ and attaching a branch to it somewhere. Let k_i and k_{i+1} be the number of vertices with degree greater than two in A_i and A_{i+1} respectively. We know that $k_i \leq i-2$. Adding a single branch to such a tree might at most increase the number of vertices with degree greater than two by one. Therefore, $k_{i+1} \leq i-1$, which means that the property holds. ∎

Define an *essential path* P in a tree T to be a path in T, whose end points are either leaves or vertices of degree greater than 2 and all other vertices in P are of degree 2 with respect to T.

Property 1.2 *A tree T with t leaves has a unique decomposition of at most $2t-3$ essential paths.*

Proof It suffices to show the result for trees that have no vertices of degree two because such vertices affect neither the number of essential paths nor the number of leaves. So, now the number of essential paths is equal to the number of edges in the tree. Also, from Property 1.1 the number of vertices with degree greater than two is at most $t-2$. Adding such vertices to the number of leaves, we get the total number of vertices to be at most $2t-2$. Therefore, the number of edges is at most $2t-3$, which we established earlier is the number of essential paths. ∎

Property 1.3 *A tree T with n vertices and t leaves has at most n^t subtrees.*

Proof Consider a subtree T' of T. The vertices in T, but not in T' that have a neighbor in T' are designated *bounding vertices*. Please refer to Figure 1.3, where the vertices in T' are shown in red and the bounding vertices are shown in blue. We claim that the number of bounding vertices cannot exceed the number of leaves in T given by t. In order to show that this claim holds, consider the forest produced by removing the vertices of T' from T. Each tree in the forest has exactly one bounding vertex and at least one vertex that was a leaf in T. Therefore the claim holds. There are n^t subsets of size t that could potentially be valid bounding vertices. Also, each set of bounding vertices uniquely defines a single subtree. Therefore there are at most n^t subtrees. ■

Property 1.4 *Let T be a tree with n vertices and t leaves of which one is designated the root. There are at most n^{t-1} subtrees of T that contain the root.*

Proof The argument is very similar to Property 1.3. In this case, there are only $t-1$ leaves outside the bounding vertices because the root leaf is inside. Therefore there are at most n^{t-1} rooted subtrees. ■

1.6 Thesis Outline

This thesis is organized as follows. In Chapter 2, we study the Vehicle Scheduling Problem with release and handling time. Our results are primarily for the offline SVSP. In Section 2.2, we provide a linear time algorithm for a restriceted version of the SVSP. We, then, use that algorithm to derive a PTAS for the more general problem in Section 2.3. Our PTAS runs in time linear in n, but exponential in both $1/\varepsilon$ and t, where ε is the approximation guaranteed by the PTAS. We extend this PTAS to the ZMVSP in Section 2.4.

We introduce a new class of problems called Online Packet TSP Class (OP-TSP-CLASS) in Chapter 3. We provide lower and tight upper bounds for the problems in

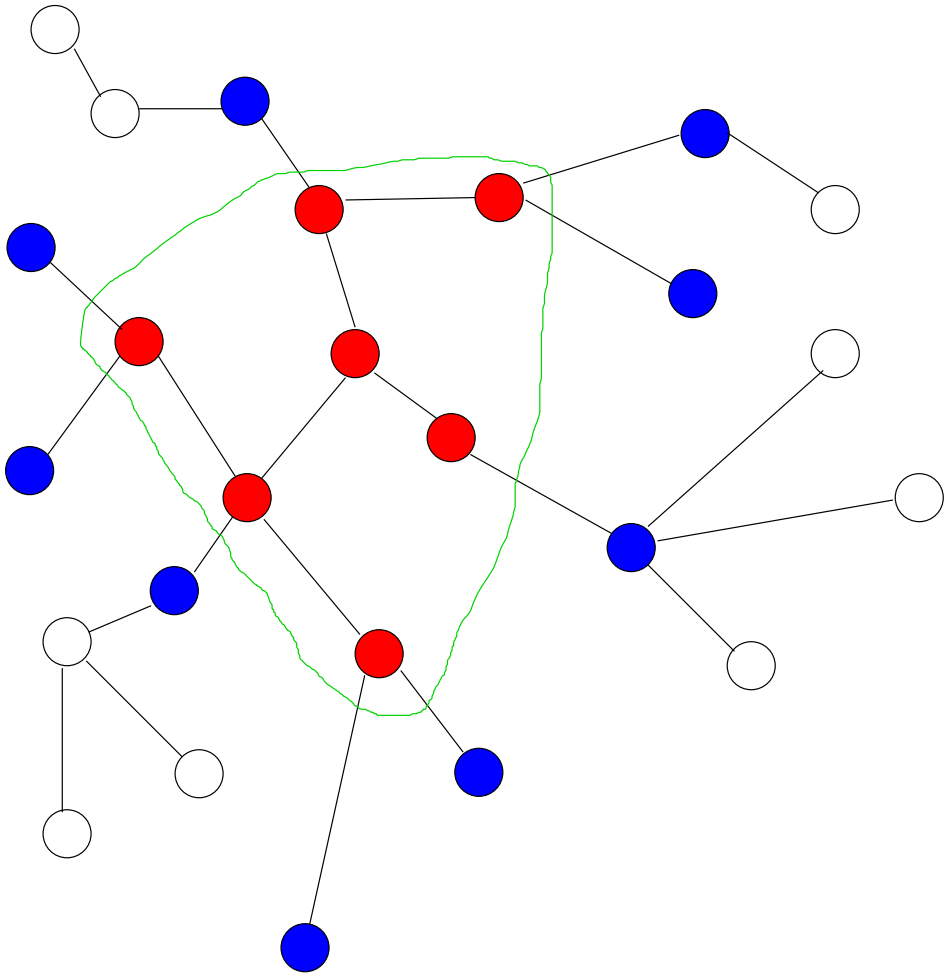

Figure 1.3: Subtree can be defined by its bounding vertices.

this class in Sections 3.2 and 3.3 respectively. We go on to derive an approximation algorithm for the offline Precedence-Constrained TSP in Section 3.4, which finds application in low power computing.

2. Offline Vehicle Scheduling*

2.1 Background

In this chapter we study the Multiple Vehicle Scheduling Problem (MVSP), which involves scheduling a set of vehicles to handle jobs at different sites. There are a large number of applications for such problems, for instance, scheduling automated guided vehicles [26], scheduling delivery ships on a shoreline [37], scheduling flexible manufacturing systems [26], etc. MVSP is also equivalent to certain machine scheduling problems where there are costs for reconfiguring machines to perform different operations [5] and to power consumption minimization in CPU instruction scheduling [7]. Our main contribution is a Polynomial Time Approximation Scheme (PTAS) that runs in linear time for the MVSP.

2.1.1 Problem Description

MVSP is a variation of the well-known Traveling Salesman Problem. In the most general formulation, the problem consists of a metric space $\mathcal{M}$ along with n jobs. Each job j becomes available for processing at a time $r_j \geq 0$ known as its *release time*. Job j requires a specific amount of time $h_j \geq 0$ for its completion known as its *handling time*. Job handling is non-preemptive in nature meaning that if the vehicle starts processing a job, it is required to complete the processing without interruption. Finally, each job has

*Published work done in collaboration with Dr. Steven Seiden. Reprinted with permission.

a *position* p_j in $\mathcal{M}$. We are given a set of k vehicles that can traverse $\mathcal{M}$ and handle or *serve* these jobs. Our goal is to minimize the maximum completion time over all jobs, called the *makespan*, using the given set of vehicles.

Note that without loss of generality $\mathcal{M}$ is finite. $\mathcal{M}$ can be represented by a weighted graph, where the points are vertices, and the distance from $p \in \mathcal{M}$ to $q \in \mathcal{M}$ is the length of the shortest path from p to q in the graph. In an abuse of notation, we also use $\mathcal{M}$ to denote this graph. We are interested in the case where $\mathcal{M}$ is a tree; unless stated otherwise, all the discussions that follow pertain to this case. We use m and t to denote the number of vertices and leaves in $\mathcal{M}$, respectively. We have provided a mathematically precise formulation in Section 1.3 on page 10. Note that a particularly interesting special case is $t = 2$, where $\mathcal{M}$ is a path.

Several different variants of this problem are possible:

- The single vehicle scheduling problem (SVSP) is just the special case $k = 1$.
- In the zone multiple vehicle scheduling problem (ZMVSP), the vehicles all operate in disjoint subtrees of G called *zones*. Part of the problem is to specify the zones.
- There are a large number of possibilities for vehicle starting/ending constraints. It is possible that the starting positions of the vehicles are given as part of the problem, or that they can be chosen by the algorithm. We call a problem-specified starting point for a vehicle the *origin* of that vehicle. There are analogous possible constraints on vehicle ending positions. The most common variant when an ending position is specified is that the origin and the ending position are the same. We denote this variant as RTO (return to origin). When no ending position is specified, the most common variant is that each vehicle has a fixed origin. We denote this variant as FO (fixed origin).

- In the *online* variants of these problems, jobs are unknown before their release, and even the number of jobs is a priori unspecified.

Since even SVSP on a path is NP-hard [19, 43], we shift our focus from finding an optimal solution to finding an approximate solution with cost that is guaranteed to be within a certain bound relative to the optimal cost. Suppose we have an algorithm $\mathcal{A}$ for problem $\mathcal{P}$. We define $\text{cost}_{\mathcal{A}}(\sigma)$ to be the cost of the solution produced by $\mathcal{A}$ on instance σ of $\mathcal{P}$. Let $\text{cost}(\sigma)$ be minimum possible cost for σ. A *polynomial time* ρ-*approximation algorithm* $\mathcal{A}$ guarantees that

$$\text{cost}_{\mathcal{A}}(\sigma) \leq \rho\,\text{cost}(\sigma)$$

for every instance σ of $\mathcal{P}$ and that $\mathcal{A}$ runs in time that is polynomial in $|\sigma|$. A *polynomial time approximation scheme* (PTAS) for problem $\mathcal{P}$ is a family of approximation algorithms $\{\mathcal{A}_\varepsilon\}_{\varepsilon \geq 0}$ such that each is a polynomial time $(1+\varepsilon)$-approximation algorithm for $\mathcal{P}$. A *fully polynomial time approximation scheme* (FPTAS) is a PTAS whose running time is polynomial in both $|\sigma|$ and $1/\varepsilon$. The reader is referred to [21] for a more comprehensive treatment of approximation algorithms.

2.1.2 Previous Results

Psaraftis *et al.* [37] consider SVSP on a path when all handling times are zero. They show that the RTO version can be solved exactly in $O(n)$ time, while the FO version can be solved exactly in $O(n^2)$ time. Psaraftis *et al.* further give 2-approximation algorithms for both these versions of SVSP with positive handling times. Tsitsiklis [43] shows that the FO and RTO versions of SVSP on paths with release and handling times are NP-complete. MVSP is NP-complete for all $k \geq 2$ even if all release times are zero and there is only a single point in $\mathcal{M}$, since this is exactly the multiprocessor scheduling problem [19]. If k is part of the input, then MVSP is strongly NP-complete. For paths, Karuno *et al.* develop a $3/2$-approximation algorithm for the RTO version of SVSP [29]

and a 2-approximation for the version of MVSP where the origins and ending points are not pre-specified [26]. For SVSP on trees, Karuno, Nagamochi and Ibaraki [28] develop a 2-approximation algorithm. For SVSP on trees with zero handling times, Nagamochi, Mochizuki and Ibaraki [35] give an exact algorithm that runs in time $O(n^t)$ and show strong NP-hardness. For SVSP on general metrics, they give an exact algorithm which runs in time $O(n^2 2^n)$, and a $5/2$-approximation algorithm.

There has also been interest in the online version of SVSP. Ausiello *et al.* [2] investigate the online RTO and FO versions of the problem, where all handling times are zero. For a general class of metrics spaces, they show 2.5 and 2-competitive algorithms for the FO and RTO variants, respectively. For the FO version, they give a $7/3$-competitive online algorithm, and a lower bound of 2 on the competitive ratio of any online algorithm. For the RTO variant, their upper and lower bounds are $7/4$ and $(9+\sqrt{17})/8 > 1.64038$, respectively.

There is a large body of work on vehicle scheduling problems with different job requirements, metric spaces and objective functions. For instance, Tsitsiklis [43] considers job deadlines, while Charikar *et al.* [7] consider precedence constraints. We do not give a comprehensive treatment of all variations here, but refer the reader to the survey of Desrosiers *et al.* [15].

2.1.3 Our Results

In this chapter, we develop a PTAS that can be applied to many of the variants of SVSP. We begin in Section 2.2 by giving an exact algorithm for solving the FO variant of SVSP on a tree when the number of distinct release times is at most R. This algorithm runs in time $O(R(m+1)^{(R-1)(t+1)+1}n)$. In Section 2.3, we use this result to provide an $O(f(1/\varepsilon,t)n)$ time $(1+\varepsilon)$-approximation algorithm for the FO variant of SVSP, where $f(1/\varepsilon,t)$ is a function exponential in both t and $1/\varepsilon$. This is accomplished

by running the algorithm of Section 2.2 on a modified problem on a modified metric space. In this modified problem, R and m are constants depending only on ε. Our PTAS is easily adapted to all the other starting/finishing constraints previously mentioned, as well as others. In Section 2.4, we extend our algorithm to include ZMVSP using a dynamic programming approach. Essentially, this multiplies the running time by a factor of $O(n^t)$. In Section 2.5, we show that an extension of SVSP to include deadlines is NP-hard, even when all release times are zero. Finally, in Section 2.6, we show how to adapt the algorithms of Ausiello *et al.* [2] to get competitive online algorithms for some SVSP variants. Recently and independently, Karuno and Nagamochi [27] have also developed PTAS's for the vehicle scheduling problems described here. Their approach is different than ours. They develop an exact pseudopolynomial time algorithm for MVSP. The running time of this algorithm is exponential in k and polynomial in $\sum_j h_j$. We get a linear time PTAS for SVSP where they do not. Our PTAS for ZMSVP runs in time polynomial in k, whereas all their algorithms have running times exponential in k.

2.2 A Special Case

In this section, we consider the single vehicle scheduling problem on trees when there are a constant number R of distinct release times. We show that this problem can be solved exactly in time $O(R(m+1)^{(R-1)(t+1)+1}n)$. We assume the FO variant, but the algorithm given here can easily be adapted to handle all of the different starting and ending conditions described in the introduction.

We denote the origin by p_0. We assume that in the input, $\mathcal{M}$ is in adjacency list form. We use $d(x,y)$ to mean the distance from point x to point y in $\mathcal{M}$. We use $x \rightsquigarrow y$ to denote the set of vertices on the shortest path from x to y, including x and y. It is easy to see that vertices of degrees one and two containing no request can be eliminated from $\mathcal{M}$. However, vertices with degree greater than 2 cannot be eliminated. We know from

Property 1.1 on page 16 that the number of vertices with degree greater than 2 is at most $t-2$. Also, the vertices containing requests, which is at most n cannot be eliminated. Therefore we have $m \leq n+t-2$.

A *schedule* for the single vehicle problem is just a permutation π on $\{1,\ldots,n\}$, that is, $\pi(i) = j$ implies that the jth job is served in the ith position in π. We also use the notation $\pi^{-1}(j)$ to refer to i. In an abuse of notation, we define $\pi(0) = 0$. The *arrival time* $a^{\sigma}_{\pi}(i)$ and *completion time* $c^{\sigma}_{\pi}(i)$ of the vehicle at the ith request in π are defined

$$\begin{aligned} a^{\sigma}_{\pi}(i) &= c^{\sigma}_{\pi}(i-1) + d(p_{\pi(i-1)}, p_{\pi(i)}), \\ c^{\sigma}_{\pi}(0) &= 0, \\ c^{\sigma}_{\pi}(i) &= \max\{r_{\pi(i)}, a^{\sigma}_{\pi}(i)\} + h_{\pi(i)}. \end{aligned}$$

If the problem instance is clear from the context, we drop the σ superscript. The cost of π is $c^{\sigma}_{\pi}(n)$. We say that schedule π *eagerly* serves request ℓ if for all i such that $p_{\ell} \in p_{\pi(i-1)} \rightsquigarrow p_{\pi(i)}$ either $\pi(\ell) \leq i$ or $r_{\ell} > c_{\pi}(i-1) + d(p_{\pi(i-1)}, p_{\ell})$. If π eagerly serves all requests, we say that π is *eager*. Intuitively, an eager schedule never passes through the location of an available request without serving the request.

Lemma 2.1 *For any finite metric $\mathcal{M}$, if there is a schedule π for a single vehicle scheduling problem σ with cost x then there is also an eager schedule ϖ for σ with cost at most x.*

Proof In order to provide an intuition for this lemma, we provide Figure 2.1 showing a non-eager schedule and Figure 2.2 showing the corresponding eager schedule on pages 26 and 27 respectively. Please note that the figures illustrate the case where the metric space is a line, i.e., $t = 2$. The dots in the time-space plane show the release time and position of the job requests.

Consider some schedule π. Define

$$e(i,\ell) = c_{\pi}(i-1) + d(p_{\pi(i-1)}, p_{\ell}),$$

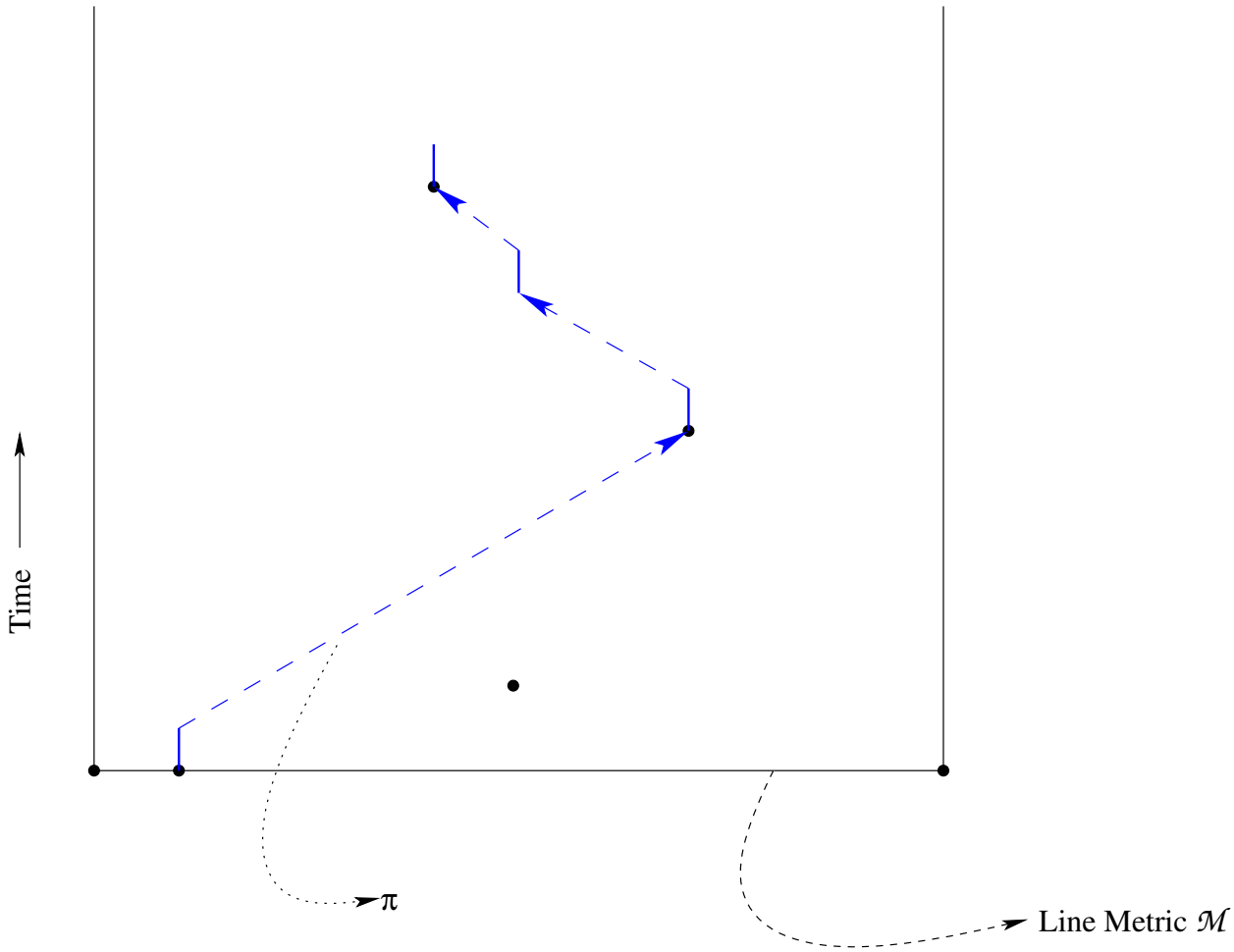

Figure 2.1: Non-eager schedule (π) on a line metric

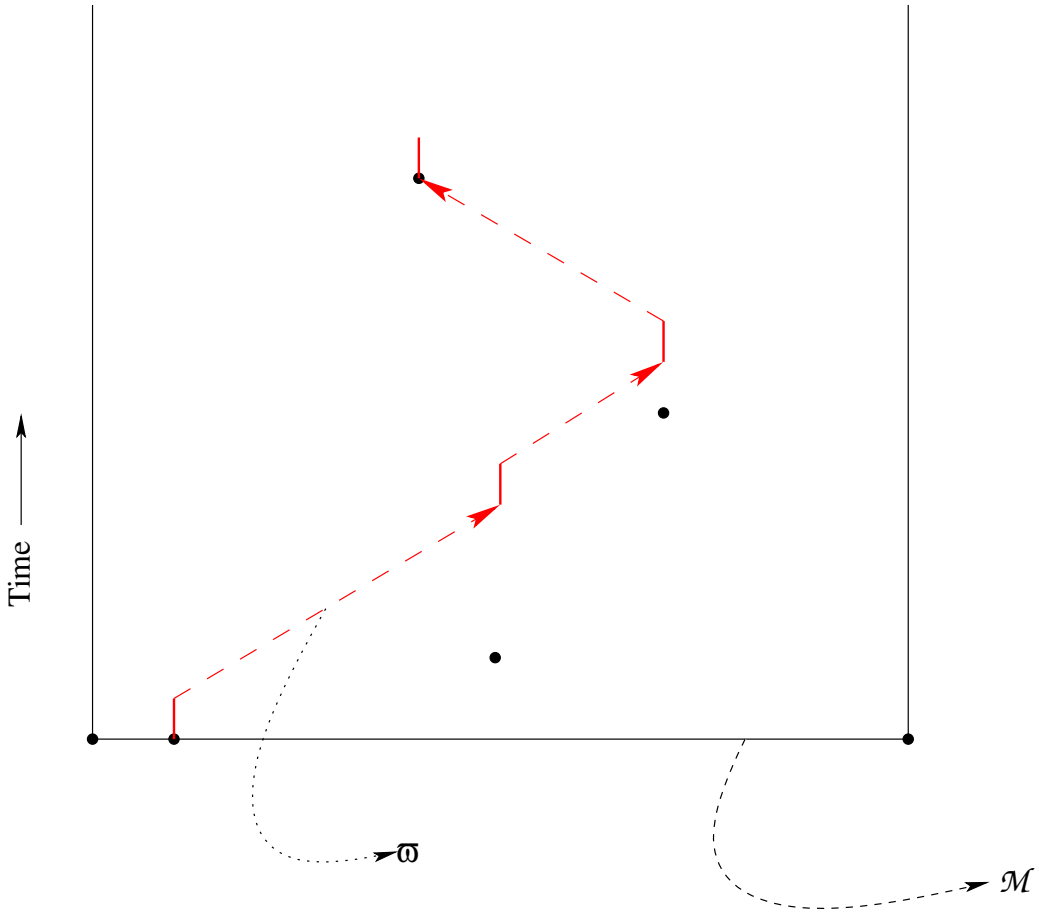

Figure 2.2: Eager schedule (ϖ) on a line metric

$$f_\ell \;=\; \min\{i \mid \min(a_\ell, r_\ell) \le e(i,\ell), p_\ell \in p_{\pi(i-1)} \rightsquigarrow p_{\pi(i)}\}.$$

Intuitively, $e(i,\ell)$ is the earliest point in time that position p_ℓ can be reached after servicing requests $\pi(1),\ldots,\pi(i-1)$. The vehicle crosses request ℓ for the first time after it becomes available when traveling from request $\pi(f_\ell - 1)$ to request $\pi(f_\ell)$. f_ℓ is well defined since $p_\ell \in p_{\pi(i-1)} \rightsquigarrow p_{\pi(i)}$ and $\min(a_\ell, r_\ell) \le e(i,\ell)$ for $i = \pi^{-1}(\ell)$. If $f_\ell = \pi^{-1}(\ell)$ for all ℓ, then π is eager. Otherwise, there is some request ℓ with $f_\ell < \pi^{-1}(\ell)$. Among these requests, let L be the one which minimizes $e(f_\ell, \ell)$. L is the first request crossed by π which is not eagerly served. Define $q = \pi^{-1}(L)$. Basically, we modify π to get π' by removing L from its current position in the order defined by π and inserting it between requests $\pi(f_L - 1)$ and $\pi(f_L)$. This causes the service of requests $\pi(f_L),\ldots,\pi(q-1)$ to be delayed by at most h_L. However, in the modified schedule, we go directly from request $\pi(q-1)$ to $\pi(q+1)$, and so we arrive at $\pi(q+1)$ at least as early as before. More formally, we define

$$\begin{aligned}
\pi'(i) &= \pi(i), && \text{for } 1 \le i < f_L;\\
\pi'(f_L) &= L;\\
\pi'(i) &= \pi(i-1), && \text{for } f_L < i \le q;\\
\pi'(i) &= \pi(i), && \text{for } q < i \le n.
\end{aligned}$$

We first note that since $p_L \in p_{\pi(f_L-1)} \rightsquigarrow p_{\pi(f_L)}$ and $r_L \le e(f_L, L)$ we have $a_{\pi'}(f_L+1) = a_\pi(f_L) + h_L$. By induction, it is easy to show that

$$c_{\pi'}(i) \le c_\pi(i-1) + h_L,$$

for $f_L < i \le q$. Now note that

$$\begin{aligned}
a_{\pi'}(q+1) &= c_{\pi'}(q) + d(p_{\pi'(q)}, p_{\pi'(q+1)})\\
&= c_{\pi'}(q) + d(p_{\pi(q-1)}, p_{\pi(q+1)})\\
&\le c_\pi(q-1) + h_L + d(p_{\pi(q-1)}, p_{\pi(q+1)})
\end{aligned}$$

$$
\begin{aligned}
&\leq c_\pi(q-1) + h_L + d(p_{\pi(q-1)}, p_{\pi(q)}) + d(p_{\pi(q)}, p_{\pi(q+1)}) \\
&= a_\pi(q) + h_L + d(p_{\pi(q)}, p_{\pi(q+1)}) \\
&\leq c_\pi(q) + d(p_{\pi(q)}, p_{\pi(q+1)}) \\
&= a_\pi(q+1).
\end{aligned}
$$

Using this fact, it is easy to show by induction that $c_{\pi'}(i) \leq c_\pi(i)$ for $q < i \leq n$. Therefore, the cost of π' is at most the cost of π. We have increased the number of eagerly served requests by one. By iterating this process, we eventually reach an eager schedule ϖ. ■

We use $0 \leq u_1 < \cdots < u_R$ to denote the possible release times. Define $u_0 = 0$ and $u_{R+1} = \infty$. Define *phase i* to be the time interval $[u_i, u_{i+1})$ for $0 \leq i \leq R$. We show that it is possible to construct the optimal schedule in polynomial time with respect to n, when t is fixed. We do so by establishing a one-to-one correspondence between an optimal eager schedule and a structure that it possesses. We show that these structures are enumerable in polynomial time, t being fixed.

Let π be an optimal schedule. Without loss of generality, π is eager. For the remainder of the paragraph, let i be in $\{1, \ldots, R\}$. Let X_i be the set of requests whose service is initiated during phase i. If X_i is non-empty, define T_i to be the minimal subtree of $\mathcal{M}$ which contains all the requests in X_i. Define L_i to be the set of leaves of T_i. Note that $|L_i| \leq t$ since T_i is subtree of $\mathcal{M}$, and $\mathcal{M}$ has at most t leaves. Let X_i^0 be the position of the first request served during phase i in schedule π. For $1 \leq j \leq |L_i|$, let X_i^j be the jth leaf visited by the vehicle during phase i in schedule π. For $|L_i| < j \leq t$, define $X_i^j = X_i^{|L_i|}$. If X_i is empty then we define $X_i^j = -1$ for $0 \leq j \leq t$. Define $X_0^t = p_0$.

We claim that the structure of π is completely defined by X_i^j for $1 \leq j \leq t, 0 \leq i \leq R$. This follows from the fact that π is eager and all requests released during phase i are released at the beginning of the phase. X_i consists of exactly those requests that lie in T_i and that are released at or before time u_i. Define a *sweep* to be a time period

during which the vehicle travels along some path, possibly stopping to serve requests, but without changing direction. Essentially, $t+1$ sweeps per phase are sufficient. If we sweep from X_{i-1}^t to X_i^0, then sweep from X_i^0 to X_i^1, sweep from X_i^1 to X_i^2 etc, we pass through all requests in X_i. We take this route and service all the requests in X_i when they are first encountered. Clearly, this is the optimal route that serves all request in X_i visiting $X_i^0,\ldots,X_i^{|L_i|}$ in order.

If we fix X_i^j for $1 \leq j \leq t$, $0 \leq i \leq R-1$ then note that this determines X_R and T_R, since all requests not served in phases $0 \ldots R-1$ must be served during phase R. The number of choices for X_R^0 is $m+1$. Once X_R^0 is fixed, it is easy to determine the remaining schedule in $O(n)$ time, since this is just the Hamiltonian path problem on a tree. For $1 \leq i < R$ there are $m^{t+1}+1$ possible choices for $X_i^0,,\ldots,X_i^t$. Therefore, the total number of possible schedules is at most $(m+1)(m^{t+1}+1)^{R-1} \leq (m+1)^{(t+1)(R-1)+1}$, which is independent of n.

From these observations, we conclude that there is a polynomial time algorithm for finding the optimal schedule if t is a fixed constant: We enumerate the possible schedules, of which there are at most $(m+1)^{(t+1)(R-1)+1}$, calculating the cost for each, and return the minimum cost schedule.

The calculation of the cost of a schedule, given X_i^j for $1 \leq j \leq t$, $0 \leq i \leq R$, can be accomplished in time $O(Rn)$: We first determine π. This can be accomplished by using depth first search on each sweep to determine the requests served. This takes time $O(Rn)$. From π we can calculate the cost in time $O(n)$.

Therefore, the total running time of the algorithm is $O(R(m+1)^{(t+1)(R-1)+1}n)$. If t can be fixed, the running time is linear in n and polynomial in m, since R is assumed to be a constant.

2.3 The Offline Single Vehicle Problem

In this section, we present a $(1+\varepsilon)$-approximation algorithm for SVSP, for all $\varepsilon > 0$. This paragraph provides an intuitive overview of the argument. Denote the input problem instance as σ. We first provide a framework for discretizing the problem instance σ. More specifically, we discretize two aspects of σ — temporal and spatial. We define three variants $\sigma^{\downarrow}$, $\sigma^{\uparrow}$ and σ^{*} of σ that use the framework. We then apply our algorithm described in the previous section to $\sigma^{\downarrow}$ and argue that the optimal schedule for $\sigma^{\downarrow}$, when applied to σ is upper bounded by $(1+\varepsilon)\text{cost}(\sigma)$, where $\text{cost}(\sigma)$ is the cost of the optimal schedule for σ. The use of $\sigma^{\uparrow}$ and σ^{*} will become apparent as the argument unfolds. Define $r_{\max} = \max_{1 \le i \le n} r_i$. Note that $r_{\max}$ is a lower bound on $\text{cost}(\sigma)$. Let $a = 2\lceil 1/\varepsilon \rceil$ and $\delta = r_{\max}/a$. Intuitively, δ is the duration of each discrete time unit and a is the number of discrete time units that constitute $r_{\max}$. Since $\text{cost}(\sigma) \ge r_{\max}$, we have $\delta \le \varepsilon\,\text{cost}(\sigma)/2$.

Let P be the sum of all edge weights in $\mathcal{M}$. Define $b = 2(t+1)a^2$ and $\Delta = P/b$. Intuitively, Δ is the discretized distance unit and b is the number of those units that make P. Since every edge must be traversed to serve all requests, $\text{cost}(\sigma) \ge P$ and therefore $\Delta \le \varepsilon\,\text{cost}(\sigma)/(4(t+1)a)$.

We define a new metric $\mathcal{N}$ with a constant number of points, which we use to approximate $\mathcal{M}$. In other words, unlike $\mathcal{M}$, the number of points in $\mathcal{N}$ is a constant with respect to n, but maintains the same structure as that of $\mathcal{M}$. For an illustrative example refer to Figure 2.3 on page 33. A *junction* of $\mathcal{M}$ is defined to be a vertex of degree three or more. Define an *essential path* of $\mathcal{M}$ to be a path whose endpoints are either leaves or junctions and other vertices are of degree 2. From Property 1.2 on page 16, $\mathcal{M}$ has a unique decomposition into a set E of at most $2t-3$ essential paths. We find this decomposition and perform the following operation on each essential path $p \in E$: We embed

p in the real line, with an arbitrary endpoint at position 0. The other endpoint lies at position $|p|$. This assigns each vertex v in p a non-negative coordinate value $x(v)$. We get a new path p' by rounding the coordinates to get $x'(v) = \min\{|p|, \Delta\lfloor x(v)/\Delta + 1/2\rfloor\}$. p' consists of $y = \lceil |p|/\Delta \rceil$ vertices, $y-1$ edges of length Δ, and one edge of length $|p| - \Delta(y-1)$. There is an obvious mapping from vertices in p to those in p'. From this, we get a mapping ϕ from points in $\mathcal{M}$ to points in $\mathcal{N}$. Note that the number of points in $\mathcal{N}$ is at most

$$\begin{aligned}\sum_{p\in E} \lceil |p|/\Delta \rceil &\leq \sum_{p\in E} (|p|/\Delta + 1) \\ &\leq P/\Delta + 2t - 3 \\ &= b + 2t - 3.\end{aligned}$$

Using $\mathcal{N}$, we define two new problem instances $\sigma^{\uparrow}$ and $\sigma^{\downarrow}$. Figure 2.4 is provided as an example illustrating $\sigma^{\downarrow}$ and $\sigma^{\uparrow}$ in a line metric, which is a special case of the general tree metric. Problem $\sigma^{\downarrow}$ (shown in red in Figure 2.4) is defined in terms of σ as follows:

$$r_i^{\downarrow} = \delta\lfloor r_i/\delta \rfloor, \qquad p_i^{\downarrow} = \phi(p_i), \qquad h_i^{\downarrow} = h_i,$$

for $1 \leq i \leq n$. For purposes that shall become clear, we add a request at the origin to $\sigma^{\downarrow}$, the 0th request, with $p_0^{\downarrow} = p_0$, $r_0^{\downarrow} = 0$ and $h_0^{\downarrow} = 0$. Clearly, this additional request does not affect the solution of $\sigma^{\downarrow}$ in any way, since the vehicle is already at p_0 at time 0, and the request has zero handling time. Note that in $\sigma^{\downarrow}$ there are at most $a+1$ distinct release times and $b+2t-3$ distinct job positions (not including p_0). Problem $\sigma^{\uparrow}$ (shown in blue in Figure 2.4) is defined by $r_i^{\uparrow} = r_i^{\downarrow} + \delta$, $p_i^{\uparrow} = p_i^{\downarrow}$ and $h_i^{\uparrow} = h_i$ for $1 \leq i \leq n$. As with $\sigma^{\downarrow}$, in $\sigma^{\uparrow}$ there is an additional request at the origin, the 0th request, with $p_0^{\uparrow} = p_0$, $r_0^{\uparrow} = 0$ and $h_0^{\uparrow} = \delta$.

Using the algorithm described in the preceding section, we can solve $\sigma^{\downarrow}$ exactly in time $O(n(a+1)(b+2t-2)^{(t+1)a+1}) = O(n(8(t+1)\lceil 1/\varepsilon \rceil^2 + 2t - 2)^{2(t+1)\lceil 1/\varepsilon \rceil + 1}/\varepsilon)$, which is linear in n for constant t and ε.

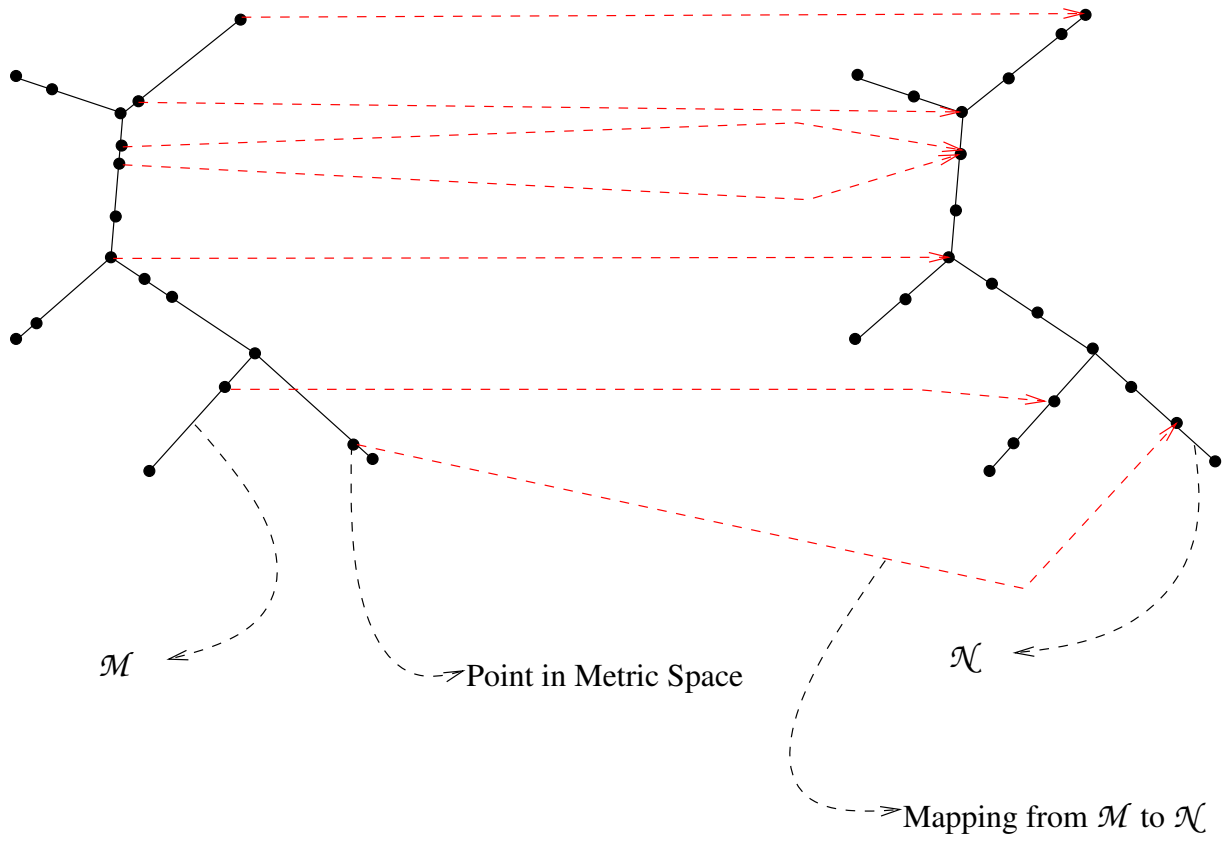

Figure 2.3: Mapping of points from $\mathcal{M}$ to $\mathcal{N}$

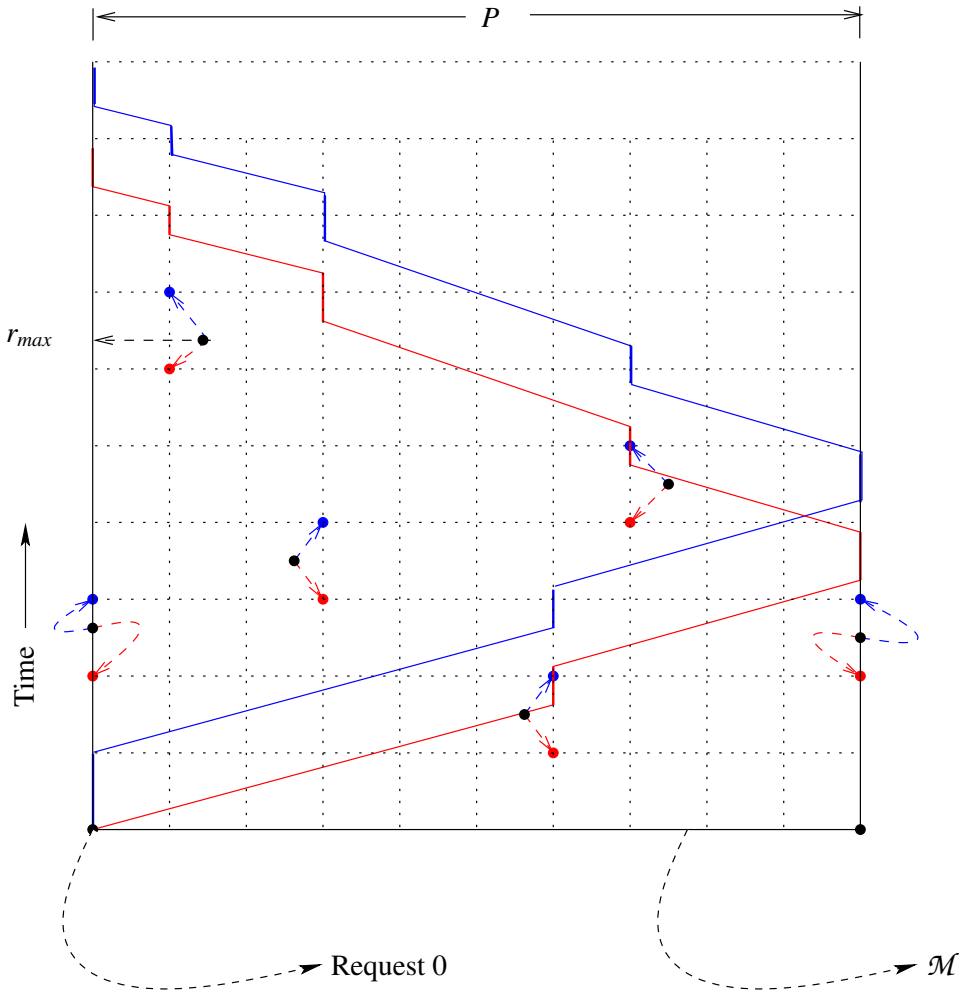

Figure 2.4: Relating $\sigma^{\downarrow}$ and $\sigma^{\uparrow}$.

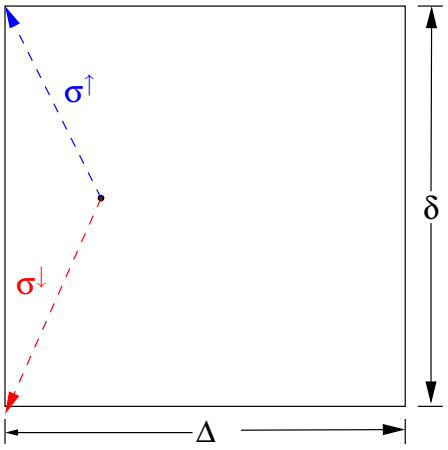

Figure 2.5: Magnified view of unit temporal and spatial grid.

We now observe that an optimal schedule π for $\sigma^{\downarrow}$ is also an optimal schedule for $\sigma^{\uparrow}$. Intuitively, problem $\sigma^{\uparrow}$ is the same as problem $\sigma^{\downarrow}$ but with all requests except 0 shifted back δ time units. Applied to $\sigma^{\uparrow}$ schedule π stays at p_0 until time δ, since $\pi(0) = 0$ and $h_0^{\uparrow} = \delta$, and then travels the same route as for $\sigma^{\downarrow}$, except that each point is reached δ time units later. The cost incurred by π on $\sigma^{\uparrow}$ is therefore $\text{cost}(\sigma^{\downarrow}) + \delta$. Note that when π is used for $\sigma^{\uparrow}$ every request is served after its release time in σ. With a bit of care, we can also use π as a schedule for σ. To ensure that the vehicle reaches all jobs, we have to increase the length of each sweep, but by at most Δ each. Therefore π is also a schedule for σ with cost at most $\text{cost}(\sigma^{\downarrow}) + \delta + (t+1)a\Delta$.

We now relate $\text{cost}(\sigma^{\downarrow})$ to $\text{cost}(\sigma)$. To accomplish this, we consider a third modified instance, which we denote σ^*. This instance is defined in terms of the original metric $\mathcal{M}$ by

$$r_i^* = r_i^{\downarrow}, \qquad p_i^* = p_i, \qquad h_i^* = h_i,$$

for $1 \leq i \leq n$. We first observe that clearly, $\text{cost}(\sigma) \geq \text{cost}(\sigma^*)$. The optimal schedule π^* for σ^* has the structure that we have explained in the preceding section; i.e. at most $t+1$ sweeps per phase are sufficient. Note that if we apply π^* to $\sigma^{\downarrow}$, we have a feasible schedule for $\sigma^{\downarrow}$. Each sweep still covers the same jobs, since the rounding scheme used to obtain $\mathcal{N}$ does not change the order of points along any essential path. Further, we increase the length of each sweep by at most Δ. Therefore, $\text{cost}(\sigma^*) + (t+1)a\Delta \geq \text{cost}(\sigma^{\downarrow})$.

We conclude that the cost incurred by the algorithm is at most

$$\begin{aligned} \text{cost}(\sigma^{\downarrow}) + \delta + (t+1)a\Delta &\leq \text{cost}(\sigma^*) + \delta + 2(t+1)a\Delta \\ &\leq \text{cost}(\sigma) + \delta + 2(t+1)a\Delta \\ &\leq (1+\varepsilon)\,\text{cost}(\sigma). \end{aligned}$$

2.4 The Offline Zone Multiple Vehicle Problem

In this section, we show that if we have a ρ-approximation algorithm $\mathcal{A}$ for SVSP which runs in time $O(g(n))$, then we also have a ρ-approximation algorithm $\mathcal{B}$ for ZMVSP which runs in time $O(tkn^t + n^t g(n))$. The basic idea is to generalize the dynamic programming algorithm given by Karuno and Nagamochi [26] for computing the optimal one-way zone schedule for the multiple vehicle scheduling problem. The general case is quite complicated, so we begin by looking at the special case of $t = 2$, where $\mathcal{M}$ is a path. We assume in this section that the starting and finishing positions of each vehicle can be selected by the algorithm.

Since the requests are all on a single path, we assume that they are given in order along this path, i.e. request 1 is at one end of the path, request 2 is adjacent to request 1, etc. Define $C^*(i,j)$ for $1 \leq i \leq j \leq n$ to be the optimal cost for serving requests $i,\ldots,j$ using a single vehicle. Further define $x^*(i,\ell)$ for $1 \leq i \leq n$ and $1 \leq \ell \leq k$ to be the cost of the optimal zone schedule for serving requests $1,\ldots,i$ with ℓ vehicles. Then the cost of the optimal zone schedule for the entire problem is given by $x^*(n,k)$. We calculate x^* using the following recurrence

$$\begin{aligned} x^*(i,1) &= C^*(1,i), \\ x^*(i,\ell) &= \min_{1\leq j<i} \max\left\{x^*(j,\ell-1), C^*(j+1,i)\right\}. \end{aligned}$$

Of course, we do not know how to calculate $C^*(i,j)$ in polynomial time. We are therefore led to consider the following modified recurrence. Define $C(i,j)$ for $1 \leq i \leq j \leq n$ to be the cost incurred by $\mathcal{A}$ for serving requests $i,\ldots,j$ with a single vehicle. Define $x(i,\ell)$ for $1 \leq i \leq n$ and $1 \leq \ell \leq k$ to be minimum cost of a zone schedule for serving requests $1,\ldots,i$ with ℓ vehicles using $\mathcal{A}$ to serve requests in each zone. Similar to the situation with x^*, we calculate $x(n,k)$ using

$$x(i,1) = C(1,i),$$

$$x(i,\ell) \;=\; \min_{1\le j<i} \max\{x(j,\ell-1), C(j+1,i)\}. \tag{2.1}$$

We show that $x(i,\ell) \le \rho\, x^*(i,\ell)$ for $1 \le i \le n$ and $1 \le \ell \le k$. This is simple to accomplish by induction. For the base case, from the definition of $\mathcal{A}$ we have $x(i,1) = C(1,i) \le \rho\, C^*(1,i) = \rho\, x^*(i,1)$. For the inductive case, assume that $x(j,\ell-1) \le \rho\, x^*(j,\ell-1)$ for all $1 \le j < i$. Then

$$\begin{aligned} x(i,\ell) &= \min_{1\le j<i} \max\{x(j,\ell-1), C(j+1,i)\} \\ &\le \min_{1\le j<i} \max\{\rho\, x^*(j,\ell-1), \rho\, C^*(j+1,i)\} \\ &= \rho \min_{1\le j<i} \max\{x^*(j,\ell-1), C^*(j+1,i)\} = \rho\, x^*(i,\ell). \end{aligned}$$

In particular, this means that $x(n,k) \le \rho\, x^*(n,k)$, which leads us to a ρ-approximation algorithm $\mathcal{B}$:

1. Calculate the values $C(i,j)$ for $1 \le i \le j \le n$, storing them in an array.
2. Calculate $x(n,k)$ using dynamic programming (i.e. store x in an array).
3. From x find the zone partition and use the schedule of $\mathcal{A}$ within each zone.

Step 1 takes $O(n^2 g(n))$ time. Step 2 takes $O(kn^2)$ time. Step 3 can be accomplished in $O(k)$ time if we record the values of j minimizing (2.1) in Step 2.

We now sketch the general solution for tree metrics. In an abuse of notation, we use $C(T)$ to represent the cost of serving requests in a tree T with a single vehicle and $x(T,\ell)$ to represent the cost of serving requests in tree T with ℓ vehicles. To begin, we calculate the cost $C(T)$ for each subtree T of $\mathcal{M}$. From Property 1.1 on page 16, the total number of subtrees is at most n^t, so the time to do this is $O(n^t g(n))$.

We now have to build a method of employing dynamic programing to trees. We pick an arbitrary leaf r and designate it to be the root. The partial solutions we build are subtrees of $\mathcal{M}$ containing r. To find the minimum cost $x(T,\ell)$ of an ℓ vehicle solution for a rooted tree T, we use depth first search starting from each leaf of T, excluding the

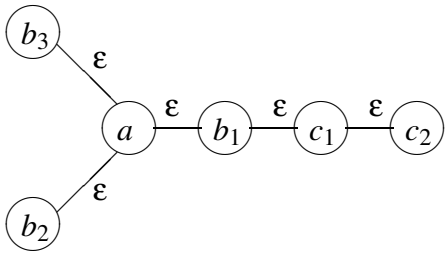

Figure 2.6: Tree for the bad instance with $t = 3$ and $k = 4$.

root. At each point in the depth first search, we have a decomposition of T into two disjoint subtrees: the portion of T visited in the depth first search, which we call U, and the remainder, which we call $V = T - U$. V contains the root. The minimum of $\max\{x(V, \ell - 1), C(U)\}$ over all possible U and V gives us the minimum cost for T. The time required to calculate $x(T, \ell)$ in $O(tn)$, since T has at most t leaves. From Property 1.4, the number of rooted trees T is at most $O(n^{t-1})$. The total time used is therefore $O(n^t g(n) + tkn^t)$, as claimed.

We now make a number of remarks on the relationship between the cost of the optimal zone schedule, and the optimal non-zone schedule. If we allow multiple requests to appear at a single location, then clearly the cost of the optimal zone schedule can be k times the cost of the optimal non-zone schedule: Consider an input where $n = k$, $r_j = 0$, $h_j = 1$ and $p_j = p_1$ for $1 \leq j \leq k$. Then in a zone schedule, a single vehicle must serve all requests, whereas in a non-zone schedule we can devote a vehicle per request. If requests must occur at distinct locations, then we get a weaker bound.

Lemma 2.2 *For all $k \geq 2$ and $t \geq 2$, there exists an MVSP problem instance σ where the cost of the optimal zone schedule is $2 - 1/t$ times the cost of the optimal non-zone schedule.*

Proof Consider an instance with $n = t + k - 1$ jobs on a tree with n vertices. We name the vertices a, $b_1, \ldots, b_t$ and $c_1, \ldots, c_{k-2}$. All edges have weight ε. There is an edge

from a to b_i for $1 \leq i \leq t$, from b_1 to c_1 and from c_i to c_{i+1} for $1 \leq i \leq k-3$. There are no other edges. An example is shown in Figure 2.6. All requests are released at time 0. There is one request at each vertex. The requests at a and $c_1, \ldots, c_{k-2}$ all have handling time 1. The requests at $b_1, \ldots, b_t$ all have handling time $1/t$. The optimal non-zone schedule completes in time $1 + (2t-2)\varepsilon$: We use a vehicle at each of a and $c_1, \ldots, c_{k-2}$, while a single vehicle serves $b_1, \ldots, b_t$. In any schedule that completes before time 2, a vehicle must be used at each of a and $c_1, \ldots, c_{k-2}$. In a zone schedule that completes before time 2, the vehicle that serves a must be used to serve the requests at $b_2, \ldots, b_t$. Therefore, the completion time of this vehicle is at least $2 - 1/t + (2t-4)\varepsilon$. Since we can choose ε to be arbitrarily small, we get the desired result. ∎

2.5 The Single Vehicle Problem with Deadlines

In this section we consider an extension of SVSP, where each job j has a deadline d_j, by which it must complete. We say that a schedule is *feasible* if it completes all jobs before their deadlines. The objective is to find the feasible schedule with minimum makespan, if it exists. Tsitsiklis [43] shows that SVSP with deadlines on paths is strongly NP-hard, but leaves open the complexity of SVSP with general deadlines and zero release times. In this section, we show that this problem is NP-hard on paths and strongly NP-hard on trees.

To begin, we define our problems precisely. In the problem *decision RTO-SVSP*, we are given a bound D, a metric $\mathcal{M}$, an origin $o \in \mathcal{M}$, and n jobs specified by (p_j, r_j, h_j) for $1 \leq j \leq n$. We are to determine if there is a schedule where the vehicle starts at o, handles all jobs and returns to o by time D. Tsitsiklis [43] shows that this problem is NP-hard, even when $\mathcal{M}$ is a path. Nagamochi, Mochizuki and Ibaraki [35] show that this problem is strongly NP-hard when $\mathcal{M}$ is a tree. (Actually, both these sets of authors consider the FO variant of SVSP, however, the problem instances they use in

their reductions all force the vehicle to return to the origin at the end of processing).

In the problem *decision deadline RTO-SVSP*, we are given a metric $\mathcal{M}$, a bound D, an origin $o \in \mathcal{M}$, and n jobs specified by (p_j, d_j, h_j) for $1 \leq j \leq n$. The goal is to determine if there is a schedule where the vehicle starts at o, handles all jobs before their deadlines and returns to o by time D.

The basic idea is very simple: decision RTO-SVSP and decision deadline RTO-SVSP are dual. By this we mean that if we reverse time in one of these problems, we get exactly the other. More precisely, we have a YES schedule π for an instance σ of decision RTO-SVSP if and only if the reverse of π is a YES schedule for the instance σ' of decision deadline RTO-SVSP defined by

$$p'_j = p_j, \qquad h'_j = h_j, \qquad d'_j = D - r_j,$$

for $1 \leq j \leq n$. The metrics, origins, and bounds D are the same in both problems. Since this is clearly a polynomial time reduction in both directions, the problems are equivalent.

2.6 The Online Single Vehicle Problem

The problem of scheduling a single vehicle online when all handling times are zero is investigated by Ausiello *et al.* [2]. The FO and RTO variants of this problem are of interest. For both FO and RTO, they obtains results for general metrics, and improved results for paths.

We show that if one has a c-competitive online algorithm for zero handling times, then it is possible to get a $(c+1)$-competitive online algorithm for non-negative handling times.

Suppose we have a c competitive algorithm $\mathcal{A}$ for some variant of SVSP with zero handling times. We show how to adapt $\mathcal{A}$ to the situation where there are handling times. The idea is very simple. Given a problem instance σ with positive handling times, we

create modified instance ς which is the same as σ, except that all handling times are zero. We simulate the actions of $\mathcal{A}$ on ς, creating a schedule π. Obviously, we can do this online. We schedule our vehicle using π. Naturally, the vehicle incurs extra delays because of the handling times, however, we stress that the information we provide to $\mathcal{A}$ is exactly the information available if it were to serve ς online. Let $\text{cost}_{\mathcal{A}}(\varsigma)$ be the cost of schedule π for problem instance ς. Then the cost of this schedule for σ is just

$$\begin{aligned} \text{cost}_{\mathcal{A}}(\varsigma) + \sum_{i=1}^{n} h_j &\leq c\,\text{cost}(\varsigma) + \sum_{i=1}^{n} h_j \\ &\leq c\,\text{cost}(\sigma) + \text{cost}(\sigma) \\ &= (c+1)\,\text{cost}(\sigma). \end{aligned}$$

2.7 Chapter Summary

In this chapter, we have studied several vehicle scheduling problems. Our main contribution was a PTAS to the SVSP. In Section 2.2, we provided an exact algorithm for SVSP restricted to have a constant number of release times and leaves. In Section 2.3, we applied this algorithm to the more general problem and derived a polynomial time approximation guarantee of $1+\varepsilon$, where $\varepsilon \geq 0$. In Section 2.4, we extend this PTAS to the ZMVSP using a dynamic programming approach. In Section 2.5, we show that an extension of SVSP to include deadlines is NP-hard, even when all release times are zero. Finally, in Section 2.6, we show how to adapt the algorithms of Ausiello *et al.* [2] to get competitive online algorithms for some SVSP variants.

3. Online Packet Traveling Salesman Problem

3.1 Introduction

We introduce a new class of problems that we call the Online Packet TSP class (OP-TSP-CLASS). Our primary motivation to study this class of problems is the practical situation that arises in serving requests that arrive in *packets*. Let us briefly re-visit the Stockholm service person example. Consider the scenario where a service person is given a set or a packet of jobs that he has to serve. We assume that he has complete knowledge of the jobs that he has to visit in his current packet, but has no knowledge of the jobs that he has to visit beyond that packet. He can improve his income by serving more requests. Also, everytime he finishes his current packet, he is guaranteed a new packet of jobs. We call this problem the Online Packet Traveling Salesman Problem (OP-TSP). We provide a formulation and extend it into a class of problems for which several of our results hold.

3.1.1 Problem Description

Let J be a set of job requests located in a metric space $\mathcal{M}$. The distance between two locations a and b in $\mathcal{M}$ is given by $d(a,b)$. The requests are served by a single vehicle which moves at a constant unit speed. The handling time of each request is assumed to be zero. The set J is partitioned into m packets of requests. For $1 \leq i \leq m$, the i^{th} packet has n_i requests in it and each request is denoted by j_l^i, where $1 \leq l \leq n_i$. Where

obvious, the superscript is omitted. We assume that there is at most one request in a location per packet. Where required, we treat a request synonymously with its location. We are justified in doing so because within a packet, serving a request in a particular location implies that we can, without increasing the makespan, serve the other requests in that location. This justification does not hold for some of the variants that we will be discussing, but this does not affect our arguments. We denote the set of requests in the i^{th} packet by $\Psi(i)$. After the first packet is released, the subsequent packets are released as soon as the previous packet has been completely served. The server has no knowledge of the subsequent packets. The server may or may not be restricted to start or end in specified locations. The object is to serve all the jobs minimizing the total makespan. Let $\mathcal{X}_i$ be the set of all permutations on $\Psi(i)$ and π_i be an element of $\mathcal{X}_i$. Define ":" to be the concatenation operator. A valid schedule Π is given by $\Pi = \pi_1 : \pi_2 : \cdots : \pi_m$. Let t_i^{π} be the time taken by the server to start from the position of the last request served in π_{i-1} and complete serving all requests in $\Psi(i)$ according to π_i. The superscript of t_i^{π} may be omitted, where obvious. Let $\mathcal{X}$ be the set of all valid schedules and T_Π be the completion time of schedule Π given by

$$T_\Pi = \sum_{i=1}^{m} t_i \tag{3.1}$$

The goal is to find the schedule $\Pi_{opt} \in \mathcal{X}$ such that

$$T_{\Pi_{opt}} = \min_{\Pi \in \mathcal{X}} T_\Pi. \tag{3.2}$$

This problem can be viewed at two levels.

Macro Level: Seen overall, the problem is online. The server is unaware of the future packets that will arrive and in fact is unaware of even how many more packets will need to be served.

Packet Level: At any given time though, the server has complete knowledge of the

current packet of requests that need to be serviced. We assume the availability of a κ-approximation algorithm $\mathcal{F}$ for this offline problem that runs in time $O(f(n))$.

3.1.2 OP-TSP-CLASS

Using the OP-TSP as the basis, we can define a large number of problems which we call OP-TSP-CLASS. Many of the results discussed in this chapter hold for this entire class of problem.

Definition 3.1 *The class of problems* OP-TSP-CLASS *is the set of all problems that are characterized by two parts in their formulation:*

- *The OP-TSP with m packets, and*
- *a set ζ_i of constraints on each packet i possibly reducing the number of valid schedules.*

Note that ζ can always be empty implying that OP-TSP is always a special case of the problem. This is important because we are interested in those problems whose complexity is at least that of OP-TSP. The following are a few problems in the OP-TSP-CLASS that are of special interest to us.

OP-TSP-PRECEDENCE: The requests within a packet are constrained by precedence constraints that can be represented by a Directed Acyclic Graph (DAG). The OP-TSP is a special case when the DAG has no edges. Let the offline version of this problem, wherein the packets are all known in advance, be called PACKET-PCTSP. Strictly speaking, PACKET-PCTSP can be reformulated as a more general PCTSP. However, we maintain this name to preserve the relationship between the online problem and its corresponding offline problem.

OP-SVSP: In this variation, each request j_k^i has a release time r_k^i which is relative to the release time of the i^{th} packet. That is, if the packet i is released at time t, then

j_k^i can be served any time after $t + r_k^i$. In addition, the request can also have non-negative handling time h_k^i. This problem is interesting because it is a variation on the Online Single Vehicle Problem discussed in Section 2.6. The case where release and handling times are all zero is equivalent to the OP-TSP.

Our lower bound result can be extended to a problem that Feder *et al.* have studied in [18], which they call the k-reordering problem. Here, $J = 1, 2, \cdots, n$ is a set of job requests located on $\mathcal{M}$ and we are allowed to reorder the requests to minimize the makespan as long as for any two jobs i and j, $i \prec j$ if $j \geq i + k$, where k is the window size. The online algorithm has knowledge of requests from i to $i + k - 1$ where i is an unserved request and all requests prior to i have been served.

The rest of this chapter is organized as follows. In Section 3.2, we analyze the lower bound on the performance of any online algorithm for the class of problems. We provide upper bounds in Section 3.3. We apply our results to the Precedence-Constrained TSP in Section 3.4 providing approximation algorithms.

3.2 Lower Bound

Theorem 3.1 *If $\mathcal{A}$ is a ρ-competitive algorithm on a generic metric space $\mathcal{M}$ for solving the OP-TSP, then $\rho \geq 5/3$. This result extends to all problems in OP-TSP-CLASS and the k-reordering problem.*

Proof The lower bound is achieved on the line. Although the server may not be required to start from any particular location for the first packet, this does not hold for subsequent packets. This is because the server's starting position for each subsequent packet is the location of the last request served in the previous packet. Let o be the starting position of the server and let p be the packet that has just arrived. Let packet p contain requests a and b at unit distance from o on either side. Since the two jobs are equidistant from o, we can assume without loss of generality that the algorithm serves b followed by a.

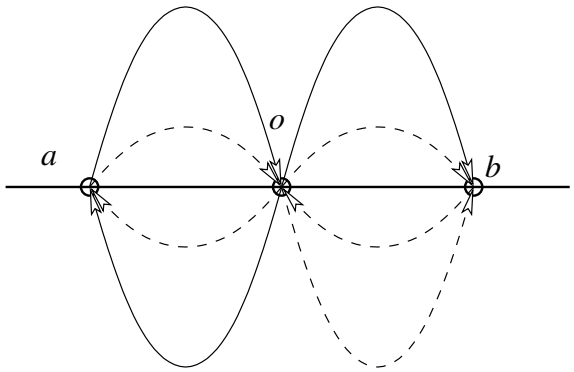

Figure 3.1: Lower bound on line

Now the adversary releases packet $(p+1)$ containing a single request at location b. This forces the server to travel from a to b. This traversal of the vehicle serving requests in packets p and $(p+1)$, shown by dotted line in Figure 3.1, costs five units of time. On the other hand, with prior knowledge that the $(p+1)^{th}$ packet has a single job request at b would allow us to optimally schedule the job at a followed by b as shown by the continuous line. This will cost us three units of time. Repetition can be established by making b the origin and $(p+2)^{th}$ packet containing two requests at unit distance from b on either side, and so on. Thus we see a $5/3$ lower bound on the competitive ratio.

This result extends to OP-TSP-CLASS because OP-TSP is a special case of these problems. However, establishing the $5/3$ lower bound for the k-reordering problem is not as straightforward. We use the same idea as before, in that we consider $\mathcal{M}$ to be a line and o to be the origin. The window size k is two. The two jobs initially in the window are a followed by b. Now we will have to choose between a and b. Since these are not equivalent choices, we will have to treat each case separately. If the algorithm chooses a, the adversary releases the third and fourth jobs at location a. This will cost five units of time. With prior knowledge, the algorithm could have chosen b followed by a. This would have cost only three units of time. On the other hand, if the algorithm had initially chosen b, then it is forced to serve a right after b. Now the adversary can

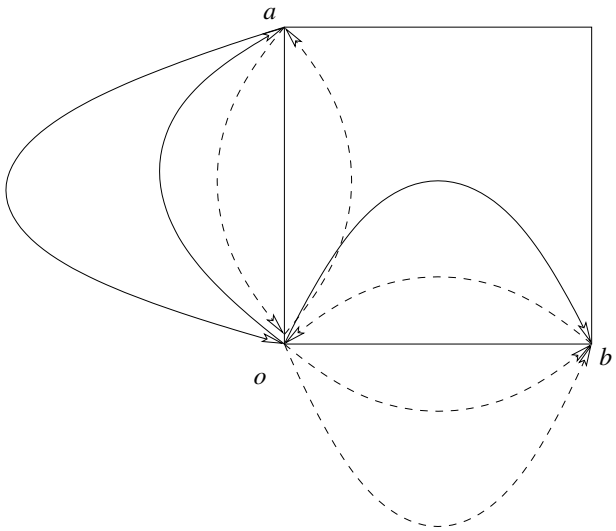

Figure 3.2: Lower bound on square

release jobs three and four in location b requiring the server to move a total of five units of distance. Again, with prior knowledge, the optimal algorithm would have served a before b requiring just three units of time. As before, the adversary can repeat this sequence with the location of the fourth request as the origin. Therefore, we cannot guarantee a competitive ratio better than $5/3$. ■

It is also interesting to see that these results extend to unit hypercubes. This is especially important to establish since scheduling for low power involves the precedence constrained traveling salesman problem on unit hypercubes. Interestingly, this lower bound holds for a unit square shown in Figure 3.2. The line that we discussed earlier is an infinite metric space, whereas the unit square is not, implying that our result is applicable to finite metric spaces as well.

The argument follows in a manner similar to Theorem 3.1. The server starts off at position o. The first packet contains two requests at a and b. Without loss of generality, we can assume that the request at b is served first followed by a. The adversary issues the second packet with a single job at b. Thus our algorithm takes 5 units of time, while

the optimal offline algorithm can perform this in 3 units of time. The paths traced by the online algorithm and the offline optimal algorithm are shown in dotted and continuous lines respectively. Thus we get a $5/3$ lower bound.

3.3 Upper Bound

We provide upper bounds for OP-TSP-CLASS in this section. These problems can be viewed from two levels. We have mentioned earlier that at the macro level, they are online problems. On the other hand, at the packet level, these are offline problems. Let the optimal offline cost of serving requests in some packet $\Psi(i)$, $1 \leq i \leq m$, with the server starting at some $o \in \mathcal{M}$ be denoted by $C_{opt}(o, \Psi(i))$. The competitive ratio of the solution that we provide is dependent on the approximation provided by the packet-level offline algorithm. Therefore we will assume that a κ-approximation algorithm $\mathcal{F}$ is available for the offline problem. We do this in spite of the fact that there exist optimal offline algorithms possibly taking exponential time in order to ensure generality. When a packet $\Psi(i)$ is released, $\mathcal{F}$ can be used to find a schedule through some $o_{alt} \in \Psi(i-1)$ that minimizes $d(o, o_{alt}) + \kappa C_{opt}(o_{alt}, \Psi(i))$, where $o \in \Psi(i-1)$ is the position of the last request served in $\Psi(i-1)$. Let the schedule obtained by this algorithm be denoted by $\Pi = \{\pi_1 : \pi_2 : \cdots : \pi_m\}$. Note that although o_{alt} is used in the algorithm, by the definition of a valid schedule it does not appear in the schedule. This does not affect the algorithm's competitive ratio adversely because of triangle inequality. The reason for using o_{alt} will become apparent as the proof unfolds. Therefore, the time taken to serve the requests in $\Psi(i)$ starting from the last request served in $\Psi(i-1)$ is

$$t_i^{\pi} \leq \min_{o_{alt} \in \Psi(i-1)} \{d(o, o_{alt}) + \kappa C_{opt}(o_{alt}, \Psi(i))\}. \tag{3.3}$$

Note that this adds a factor of n_{i-1} to the time complexity of the packet level offline algorithm.

Theorem 3.2 *The algorithm outlined above is* $(\kappa+1)$*-competitive for OP-TSP and this upper bound is tight.*

Proof For this proof, we assume that the starting location of the server is given as part of the problem and is denoted by o_{start}. The proof can be easily adapted to other starting conditions. When a new packet arrives, the server has to start from one of the requests in the previous packet and traverse through each request in the current packet.The cost of this partial schedule for the i^{th} packet given schedule Π is called its packet-level cost denoted by $C_P^{\Pi}(i)$. Since we are given a starting position, $t_1 \leq \kappa C_{opt}(o_{start}, \Psi(1))$. The overall cost of the schedule Π is given by

$$T_{\Pi} = \sum_{i=1}^{m} t_i. \tag{3.4}$$

Let T_{opt} be the total time taken by the optimal offline schedule of serving the requests. We give two lower bounds on t_i denoted by $PacketLB1(i)$ and $PacketLB2(i)$, where $1 \leq i \leq m$. We then use them to derive two lower bounds on T_{opt} denoted by $LB1$ and $LB2$. For the first lower bound on t_i, we consider the optimal cost of starting at any position $o \in \Psi(i-1)$ and traversing every request in $\Psi(i)$ without taking the $(i+1)^{th}$ packet into account. So, the first lower bound on t_i is given by

$$\begin{aligned} PacketLB1(1) &= C_{opt}(o_{start}, \Psi(1)) \\ PacketLB1(i) &= \min_{o \in \Psi(i-1)} \{C_{opt}(o, \Psi(i))\}, \text{ for } 1 < i \leq m. \end{aligned}$$

Let $o_{opt} \in \Psi(i-1)$ be such that $PacketLB1(i) = C_{opt}(o_{opt}, \Psi(i))$. Intuitively, o_{opt} is one of the best requests that could have been served last in $\Psi(i-1)$. The lower bound on T_{opt} is given by

$$LB1 = \sum_{i=1}^{m} PacketLB1(i). \tag{3.5}$$

For a packet $\Psi(i)$ we can derive a second offline lower bound by taking the distance between the two jobs that are farthest apart from each other. This is a lower bound

because the server has to traverse from one of these requests to the other. For the packet $\Psi(i)$, it is

$$PacketLB2(i) = \max_{a\in\Psi(i),b\in\Psi(i)} \{d(a,b)\}. \tag{3.6}$$

In an abuse of notation, let $PacketLB2(0) = 0$. Using this we can derive a second offline lower bound given by

$$LB2 = \sum_{i=0}^{m} PacketLB2(i) \geq \sum_{i=0}^{m-1} PacketLB2(i). \tag{3.7}$$

Consider our algorithm to have served $(i-1)$ packets and the server is now at o. From Equation (3.3), we have

$$\begin{aligned} t_i &\leq \min_{o_{alt}\in\Psi(i-1)} \{d(o,o_{alt}) + \kappa C_{opt}(o_{alt},\Psi(i))\} \\ &\leq d(o,o_{opt}) + \kappa C_{opt}(o_{opt},\Psi(i)), \\ &= d(o,o_{opt}) + \kappa PacketLB1(i) \\ &\leq PacketLB2(i-1) + \kappa PacketLB1(i). \end{aligned}$$

Now the total time taken by our algorithm to serve all requests is given by the summation of the t_i, where $1 \leq i \leq m$. That is,

$$\begin{aligned} T &= \sum_{i=1}^{m} t_i \\ &\leq \sum_{i=1}^{m} \{PacketLB2(i-1) + \kappa PacketLB1(i)\} \\ &= \sum_{i=0}^{m-1} \{PacketLB2(i)\} + \kappa \sum_{i=1}^{m} \{PacketLB1(i)\} \\ &\leq LB2 + \kappa LB1 \\ &\leq (\kappa+1)T_{opt}. \end{aligned}$$

This establishes the $(\kappa+1)$-competitive ratio.

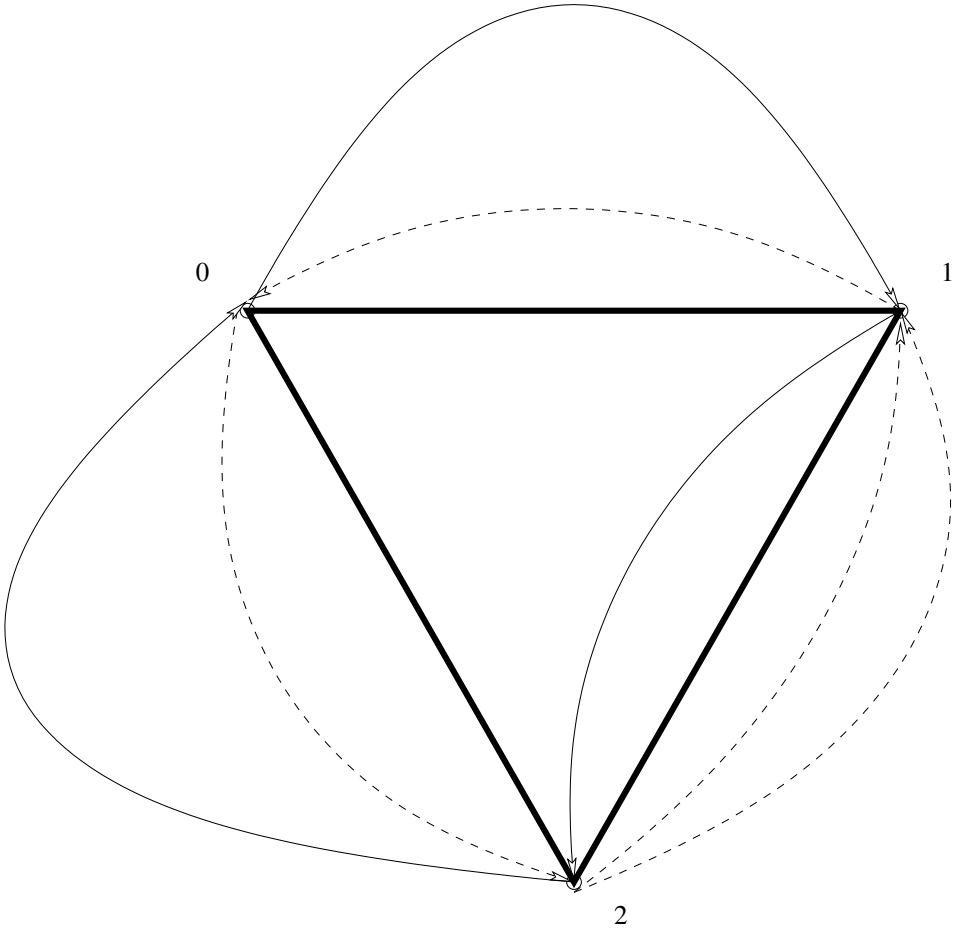

Figure 3.3: Tightness of 2-competitive algorithm

In order to establish the tightness of the above competitive ratio, consider the triangular metric space shown in Figure 3.3 with sides of unit length. The server starts off at position 2. Each packet contains two requests to be served. The i^{th} packet has two jobs in locations $(i-1) \pmod 3$ and $i \pmod 3$. We can assume that our algorithm chooses to schedule the requests in the counter-clockwise direction whenever the two requests to be served are located in the vertices opposite the server's location. It is an optimal offline algorithm implying that $\kappa = 1$, therefore requiring us to show that we have a tight 2-competitive offline algorithm. Therefore, the server moves a distance of two units from $2 \rightarrow 1 \rightarrow 0$. The adversary continues to issue packets with requests on the vertices of the side opposite to the position of the server. Therefore at the start of

the i^{th} packet, the server is always at $(i+1) \pmod 3$. This implies that for a total of m packets, the server has traveled $2m$ units of distance. In the optimal schedule, the server travels in the clockwise direction traveling a total of $m+1$ units of distance giving us a competitive ratio of $\frac{2m}{m+1}$, which is 2 as $m \to \infty$. Therefore the upper bound 2 for the competitive ratio is tight. ■

Note that when the optimal offline algorithm is used, i.e., when $\kappa = 1$, we get a competitive ratio of 2. Also, even if we don't look for a convenient o_{alt}, we get

$$t_i = C_{opt}(o, \Psi(i)) \leq d(o, o_{opt}) + C_{opt}(o_{opt}, \Psi(i)). \tag{3.8}$$

Hence, if $\kappa = 1$, we do not have to look for the o_{alt} that minimizes t_i in Equation 3.3. This reduces the time complexity of obtaining a partial schedule when the i^{th} packet is released by a factor n_{i-1}. When $\kappa > 1$, it is unclear if we can save the factor of n_{i-1} on the time complexity of the packet-level offline algorithm implied by Equation 3.3 by not searching for a o_{alt} without losing the $(\kappa+1)$ competitiveness. What we can guarantee is a 2κ-competitive ratio using an argument similar in manner to the 2-competitive algorithm shown above.

Theorem 3.2 again extends to OP-TSP-CLASS as shown in Corollary 3.3 but cannot be extended to the *k*-reordering problem.

Corollary 3.3 *There exists a* $(\kappa+1)$*-competitive algorithm for all problems in OP-TSP-CLASS.*

Proof The cost T_p and lower bounds remain the same as in Equations (3.3), (3.5) and (3.7) except that C_{opt} is the cost of the optimal offline problem with the corresponding constraints. Hence the rest of the argument follows. ■

3.4 Precedence-Constrained TSP

In this section, we explore the use of our results in Precedence-Constrained Traveling Salesman Problem (PCTSP). This problem finds application in instruction scheduling for low power computing. The jobs are constrained by precedence relations and the object is to find the schedule that minimizes the makespan without violating the precedence constraints [40].

In a valid schedule Π, a *setup* or *jump* is a pair of adjacent requests (a,b) such that a is not an immediate predecessor of b in the dag. A special case of the PCTSP was studied by Duffus *et al.* [16] and Chein *et al.* [9], where they consider minimizing the number of setups. An exact algorithm is provided in [9] for the case where the maximum width of the dag is 2. This result is generalized by Colbourn and Pulleyblank [12] by providing a polynomial time exact solution for the case where the maximum width is a constant. They use a dynamic programming approach. Bianco *et al.* give exponential time exact algorithms, also using dynamic programming [4] for the general PTSP. Charikar *et al.* [7] provide an excellent theoretical analysis of the problem. They relate it to the Shortest Common Supersequence problem and show that even for PTSP on a line, there does not exist a polynomial time k^{α}-approximation algorithm, where $\alpha > 0$ and k is the number of points on the line, unless $P = NP$. They also note that the polynomial time algorithm due to Colbourn and Pulleyblank [12] can be extended to PTSP, provided again that the width of the dag is a constant. They also give several other approximation algorithms. However, the best approximation they can guarantee on the hypercube is $|\mathcal{M}|/2$. This is prohibitively high for most applications. In situations like this researchers have resorted to mildly exponential approximation algorithms in order to ensure implementability [22]. We provide a similar mildly exponential approximation algorithm.

We propose that a suitable topological sort of the DAG be segmented into m packets. Each packet is associated with a DAG that incorporates precedence constraints which are a subset of the constraints in the original dag. Thus, it is an instance of the PACKET-PTSP, whose set of valid schedules is a subset of the valid schedules of the PTSP. This instance can be solved as if it were online producing a schedule Π^{on}. Let $T_{\Pi^{on}}$ be the completion time of schedule Π^{on} and let T^{w}_{opt} and T^{ptsp}_{opt} be the completion times of the optimal schedules for PACKET-PTSP and PTSP respectively. A valid schedule for OP-PTSP is also valid for PACKET-PTSP. which is in turn valid for PTSP. If we can establish a relationship between their completion times, we can design an approximation algorthm for PTSP.

Theorem 3.4 *Let m be the number of packets in the OP-PTSP derived from PTSP. Then* $T_{\Pi^{on}} \leq m(\kappa+1)T^{ptsp}_{opt}$.

Proof From Theorem 3.2,

$$T_{\Pi^{on}} \leq (\kappa+1)T^{w}_{opt}. \tag{3.9}$$

There are m packets in PACKET-PTSP and the optimal completion time of each packet is a lower bound on T^{ptsp}_{opt}. But we know from Equation (3.4) that

$$\begin{aligned} T^{w}_{opt} &= \sum_{i=1}^{m} t_i \\ &\leq \sum_{i=1}^{m} T^{ptsp}_{opt} \\ &= mT^{ptsp}_{opt}. \end{aligned}$$

Therefore, from Equations (3.9) and (3.10), we get $T_{\Pi^{on}} \leq m(\kappa+1)T^{ptsp}_{opt}$, which is the required result. ∎

If we use an exact algorithm such as the one provided by Bianco *et al.* in [4], we get a $2m$-approximation algorithm. Let us suppose that our exact algorithm takes time $f(n)$. If we segment the n requests into m segments, each segment can be solved in

$f(\lceil n/m \rceil)$ units of time. Since we have m segments, the total time to solve the m segments is $mf(\lceil n/m \rceil)$. Further, let w_i be the maximum width of the i^{th} segment and $W = \max_{1 \leq i \leq m} w_i$. We know from [12, 7] that each segment can be solved in $O(n^W)$ implying that the $2m$-approximation algorithm takes only $O(mn^W)$, which is polynomial if W is a constant. Considering

- exact algorithm taking time $f(n)$, which is known to be exponential, and
- the polynomial time $|\mathcal{M}|/2$-approximation,

which are our known choices, the algorithm that we have provided is a significant improvement.

3.5 Chapter Summary

In this section, we introduced a class of problems called the Online Packet TSP Class (OP-TSP-CLASS). It is a generalization of the online TSP [2], with a packet of requests known and available for scheduling at any given time. We provide a $5/3$ lower bound on any online algorithm for problems in OP-TSP-CLASS. We extend this result to the related k-reordering problem for which a $3/2$ lower bound was known [18]. We develop a $\kappa + 1$-competitive algorithm for problems in OP-TSP-CLASS, where a κ-approximation algorithm is known for the offline version of that problem. We use this result to develop an offline $m(\kappa + 1)$-approximation algorithm for the Precedence-Constrained TSP (PCTSP) by segmenting the n requests into m packets. Its running time is $mf(n/m)$ given a κ-approximation algorithm for the offline version whose running time is $f(n)$.

4. Conclusion and Future Work

We have presented the first approximation schemes for single and multiple vehicle scheduling problems on trees. We have also defined a new class of problems that we call OP-TSP-CLASS. Such problems are well motivated, having a large number of applications [26, 40]. We believe that this is just an initial step in the exploration of such problems, and therefore we list a number of open problems that we feel are important:

- Karuno and Nagamochi [26] give a 2-approximation for the non-zone multiple vehicle problem on a path. Can their result be extended to trees?
- For what other metrics is a PTAS possible? Is an FPTAS possible for SVSP on paths or trees with a constant number of leaves?
- In [2], lower bounds are given for online vehicle scheduling problems with zero handling costs. Can these lower bounds be increased using handling costs?
- We provide a common lower bound for all problems in the OP-TSP-CLASS. Can this bound be increased under other constraints?
- We have provided a theoretical framework for solving the PTSP within a constant upper bound of $m(\kappa+1)$ in a reasonable amount of time. Choosing the right values for m and κ in order to achieve good and implementable solutions is left open. An experimental work on this might provide significant results for low power instruction scheduling. Finding the right segmentation can also give us significant

reductions in the time complexity if we can restrict the maximum widths of the segments. This, again, is an open problem.

Bibliography

[1] ARORA, S. Polynomial-time approximation schemes for euclidean TSP and other geometric problems. *Journal of the ACM* 45(5), (1998) 753-782.

[2] AUSIELLO, G., FEUERSTEIN, E., LEONARDI, S., STOUGIE, L., AND TALAMO, M. Algorithms for the on-line travelling salesman. *Algorithmica 29*, 4 (2001), 560–581.

[3] BIANCO, L., MINGOZZI, A., RICCIARDELLI, S. Dynamic programming strategies and reduction techniques for the travelling salesman problem with time windows and precedence constraints. *Operations Research*, 45 (3), (May-June 1997), pp. 365–377.

[4] BIANCO, L., MINGOZZI, A., RICCIARDELLI, S. AND SPADONI, M. Exact and heuristic procedures for the traveling salesman problem with precedence constraints, based on dynamic programming. *INFOR* 32(1), (1994), 19–31.

[5] BRUNO, J., AND DOWNEY, P. Complexity of task sequencing with deadlines, set-up times and changeover costs. *SIAM Journal on Computing 7*, 4 (Nov. 1978), 393–404.

[6] BORODIN, A., AND EL-YANIV, R. Online Computation and Competitive Analysis. Cambridge University Press, (1998).

[7] CHARIKAR, M., MOTWANI, R., RAGHAVAN, P., AND SILVERSTEIN, C. Constrained TSP and low-power computing. In *Proceedings of the 5th International Workshop on Algorithms and Data Structures* (Aug. 1997), pp. 104–115.

[8] CHARIKAR, M. AND RAGHAVACHARI, B. The finite capacity dial-A-ride problem. In *Proceedings of the 39th Annual IEEE Symposium on the Foundations of Computer Science*, (1998).

[9] CHEIN, M., AND HABIB, M. Jump number of dags having Dilworth number 2. *Discrete Applied Mathematics*, Vol 7, pp. 243–250.

[10] CHRISTOFIDES, N. Worst-case analysis of a new heuristic for the travelling salesman problem. Tech. Rep. CS-93-13, Carnegie Mellon University, Graduate School of Industrial Administration, 1976.

[11] CHRISTOFIDES, N. Vehicle routing. Chapter in *The Traveling Salesman Problem*, John Wiley and Sons Ltd., (1985), pp. 431-448.

[12] COLBOURN, C. J., PULLEYBLANK, W. R. Minimizing setups in ordered sets of fixed width. *Order*, 1, (1985), pp. 225-229.

[13] CORMEN, T. H., LEISERSON, C. E., AND RIVEST, R. L. Introduction to Algorithms. MIT press, (1990).

[14] DANTZIG, G. B., FULKERSON, R., AND JOHNSON, S. M. Solution of a large-scale traveling salesman problem. *Operations Research* 2, 1954, pp. 393-410.

[15] DESROSIERS, J., DUMAS, Y., SOLOMON, M., AND SOUMIS, F. Time constrained routing and scheduling. In *Network Routing, Volume 8 of Handbooks in Operations Research and Management Science*, M. O. Ball, T. L. Magnanti, C. L. Monma, and G. L. Nemhauser, Eds. Elsevier Science, 1995.

[16] DUFFUS, D., RIVAL, I., AND WINKLER, P. Minimizing setups for cycle-free ordered sets. In *Proceedings of the American Mathematical Society*, Vol 85, Issue 4, (Aug. 1982), pp. 509–513.

[17] ENGEBRETSEN, L., AND KARPINSKI, M. Approximation hardness of TSP with bounded metrics. In *Proceedings of the 28th Annual International Colloquium on Automata, Languages and Programming* (July 2001), pp. 201–212.

[18] FEDER, T., MOTWANI, R., PANIGRAHY, R., AND ZHU, A., Web caching with request reordering. In *13th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, Jan. 2002, San Francisco.

[19] GAREY, M. R., AND JOHNSON, D. S. *Computers and Intractability: A Guide to the Theory of NP-completeness*. Freeman and Company, San Francisco, 1979.

[20] GAREY, M. R., GRAHAM, R. L. AND JOHNSON, D. S. Performance guarantees for scheduling algorithms. *Operations Research*, Vol. 26 No. 1 (1978) pp. 3–21.

[21] HOCHBUAM, D. *Approximation Algorithms for NP-hard Problems*. PWS Publishing Company, 1997.

[22] JERRUM, M., AND VAZIRANI, U. A mildly exponential approximation algorithm for the permanent. In 33^{rd} *Annual Symposium on Foundations of Computer Science*, (Oct. 1992), pp. 320-326.

[23] JESSEN, R. J. *Statistical investigation of a sample survey for obtaining farm facts*. *Research Bulletin #304*, Iowa State College of Agriculture, 1942.

[24] JOHNSON, D. S., MCGEOCH, L. A. The traveling salesman problem: a case study in local optimization. In E. H. L. Aarts and J. K. Lenstra, editors, *Local Search in Combinatorial Optimization*, John Wiley and Sons, pp 215–310, 1997.

[25] KARP, R. M. *Reducibility among Combinatorial Problems*. Plenum Press, NY, 1972, pp. 85–103.

[26] KARUNO, Y., AND NAGAMOCHI, H. A 2-approximation algorithm for the multi-vehicle scheduling problem on a path with release and handling times. In *Proceedings of the 9th Annual European Symposium on Algorithms* (Aug. 2001), pp. 218–229.

[27] KARUNO, Y., AND NAGAMOCHI, H. A polynomial time approximation scheme for the multi-vehicle scheduling problem on a path with release and handling times. In *Proceedings of the 12th International Symposium on Algorithms and Computation* (Dec. 2001), pp. 36–47.

[28] KARUNO, Y., NAGAMOCHI, H., AND IBARAKI, T. Vehicle scheduling on a tree with release and handling times. *Annals of Operations Research 69* (1997), 193–207.

[29] KARUNO, Y., NAGAMOCHI, H., AND IBARAKI, T. A 1.5-approximation for single-vehicle scheduling problem on a line with release and handling times. In *Japan-U.S.A. Symposium on Flexible Automation* (July 1998), pp. 1363–1366.

[30] KISSIN, G. Energy consumption in VLSI circuits: a foundation. In *ACM Symposium on Theory of Computing*. 1982.

[31] LAWLER E. L., LENSTRA J. K., RINNOOY-KAN A. H. G., SHMOYS D. B. , editors. *The Traveling Salesman Problem*. John Wiley, 1985.

[32] LENSTRA, J. K., RINNOOY KAN, A. H. G. Complexity of scheduling under precedence constraints. *Operations Research*, Vol. 26 No. 1 (1978) pp. 22–35.

[33] LIN, S. Computer solutions of the traveling salesman problem. *Bell System Technical Journal* Vol. 44 (1965) pp. 2245–2269.

[34] MAHALANOBIS, P. C. A sample survey of the acreage under jute in bengal. *Sankhyu* 4 (1940), pp. 511-530.

[35] NAGAMOCHI, H., MOCHIZUKI, K., AND IBARAKI, T. Complexity of the single vehicle scheduling problem on graphs. *INFOR: Information Systems and Operational Research 35*, 4 (1997), 256–276.

[36] PAPADIMITRIOU, C. H. The Euclidean traveling salesman problem is NP-complete. *Theoretical Computer Science 4*, 3 (June 1977), 237–244.

[37] PSARAFTIS, H., SOLOMON, M., MAGNANTI, T., AND KIM, T. Routing and scheduling on a shoreline with release times. *Management Science 36*, 2 (1990), 212–223.

[38] SLEATOR, D. D., AND TARJAN, R. E. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2) (1985), 202–208.

[39] "Stockholm Tunnelbana." Map of the Stockholm Tunnelbana transportation system from `http://www.sl.se/ficktid/karta/vinter/vTub.pdf` visited on August 16, 2002. Used with permission — see appendix.

[40] SU, C. L., TSUI, C. Y., AND DESPAIN, A. M. Low power architecture design and compilation techniques for high-performance processors. In *Proceedings of the IEEE COMPCON* (Feb 1994), 489–498.

[41] TIWARI, V., MALIK, S., AND WOLFE, A. Compilation techniques for low energy: An overview. In *Proc. Symp. Low Power Electronics*, (Oct 1994), pp. 38–39.

[42] TOBUREN, M. C., CONTE, T., AND REILLY, M. Instruction scheduling for low power dissipation in high performance microprocessors. In *Proc. of the Power Driven Microarchitecture Workshop*, (1998).

[43] TSITSIKLIS, J. Special cases of traveling salesman and repairman problems with time windows. *Networks 22* (1992), 263–282.

Appendix: Letters of Permission

The following is the written permission for including the Article titled "Linear Time Approximation Schemes for Vehicle Scheduling" as Chapter 2 of this thesis. The article was presented at the 8^{th} Scandinavian Workshop on Algorithm Theory, Turku, Finland, July 3-5, 2002. The proceedings from the conference appeared in the series "Lecture Notes in Computer Science" published by Springer-Verlag Berlin/Heidelberg©, Volume 2368 titled "Algorithm Theory — SWAT 2002." This title was edited by Penttonen, M., University of Kuopio, Finland and Meineche Schmidt, E., University of Aarhus, Denmark. Its ISBN number is 3-540-43866-1. The written permission, shown in Figure 4.1, was obtained in the form of an email from Ms. Caroline Davis of Springer. Figure 4.2 shows the first page of the article.

Figure 4.3 shows the written permission from Susanne Nilsson of Stockholm Transport (http://www.sl.se/international/). The permission was provided to use the map of the Stockholm city transportation. The map was available at

http://www.sl.se/ficktid/karta/vinter/vTub.pdf

as of August 16, 2002.

RE: Including paper in Master's thesis.

From: C.Davis@Springer.de
To: augustine@ieee.org
Date: Wed, 19 Jun 2002 16:10:31 +0200

Dear John,

Thank you very much for your permission request.

We are pleased to grant you the permission requested, provided that full credit (Springer-Verlag book or journal title, article title, name(s) of author(s), volume, page, figure number(s), year of publication, copyright notice of Springer-Verlag) is given to the publication in which the material was originally published.

This message is also being sent to you per fax and letter as requested.

Sincerely,
Caroline
****

Caroline Davis (Ms.)
Springer-Verlag
Rights and Permissions
Tiergartenstr. 17
D - 69121 Heidelberg
Germany
Fax: +6221-487223
Tel.: +6221-487270
E-mail: c.davis@springer.de
Home Page: http://www.springer.de/rights
Rights News: http://www.springer.de/rights/news/index.html

Figure 4.1: Written permission

Linear Time Approximation Schemes for Vehicle Scheduling

John E. Augustine[1] and Steven S. Seiden[2]*

[1] Dept. of Electrical & Computer Eng., Louisiana State University, Baton Rouge, LA 70803, USA,
augustine@ieee.org
[2] Department of Computer Science, 298 Coates Hall, Louisiana State University, Baton Rouge, LA 70803, USA,
sseiden@acm.org,

Abstract. We consider makespan minimization for vehicle scheduling problems on trees with release and handling times. 2-approximation algorithms were known for several variants of the single vehicle problem on a path [16]. A 3/2-approximation algorithm was known for the single vehicle problem on a path where there is a fixed starting point and the vehicle must return to the starting point upon completion [13]. Karuno, Nagamochi and Ibaraki give a 2-approximation algorithm for the single vehicle problem on trees. We develop a linear time PTAS for the single vehicle scheduling problem on trees which have a constant number of leaves. This PTAS can be easily adapted to accommodate various starting/ending constraints. We then extended this to a PTAS for the multiple vehicle problem where vehicles operate in disjoint subtrees. For this problem, the only previous result is a 2-approximation algorithm for paths [10]. Finally, we present competitive online algorithms for some single vehicle scheduling problems.

1 Introduction

In this paper we study the multiple vehicle scheduling problem (MVSP), which involves scheduling a set of vehicles to handle jobs at different sites. There are a large number of applications for such problems, for instance, scheduling automated guided vehicles [10], scheduling delivery ships on a shoreline [16], scheduling flexible manufacturing systems [10], etc.... MVSP is also equivalent to certain machine scheduling problems where there are costs for reconfiguring machines to perform different operations [2], and to power consumption minimization in CPU instruction scheduling [3].

Problem Description: MVSP is a variation of the well-known traveling salesman problem. In the most general formulation, the problem consists of a metric space $\mathcal{M}$ along with n jobs. Each job j becomes available for processing

* Contact author. This research was partially supported by the Research Competitiveness Subprogram of the Louisiana Board of Regents.

Figure 4.2: First page of article

VB: Permission to use map.
From: Susanne.Nilsson@sl.se
To: augustine@ieee.org
Date: Wed, 14 Aug 2002 09:52:11 +0200

Dear John,

We are happily giving you permission to use our subway map for your research. Good luck with your studies.

Best regards,

Susanne Nilsson
Kommunikation
SL AB

-----Ursprungligt meddelande-----
Från: Laurén Marie HK
Skickat: den 14 augusti 2002 09:05
Till: Nilsson Susanne
Ämne: VB: Permission to use map.

Hej Susanne.
Här kommer ännu en kartfråga som jag ber dig svara på.
Hälsn Ulla

-----Ursprungligt meddelande-----
Från: John Augustine [mailto:augustine@ieee.org]
Skickat: den 14 augusti 2002 04:11
Till: registrator@sl.se
Ämne: Permission to use map.

Hello,
First of all, let me commend you on the excellence of your transportation network. I had the pleasure of visiting Stockholm in July 2002 and am very impressed by the facility.

I am a graduate student at Louisiana State University, USA and my research involves a vehicle scheduling problem. I would like to use your map published
at <http://www.sl.se/ficktid/karta/vinter/vTub.pdf> as part of my thesis to give a motivational example to my problem. I will properly reference your website in my thesis. I would like to get your written permission for the same in the form of a reply to this email. Your response is greatly appreciated.

Sincerely,
John Augustine.

- -
~~~~~~
John Augustine
CFI, Weland Resources Bldg

Figure 4.3: Written permission
~~~~~~

Vita

John Ebenezer Augustine was born on July 8, 1977 in Madras, India. He received his Bachelor of Engineering (B.E) degree from the Univeristy of Madras in June 1998, majoring in computer science and engineering. He received a master's degree in systems science from Louisiana State University in May 2001. He will receive his master's degree in electrical engineering from the department of Electrical and Computer Engineering in December 2002. He will be pursuing his doctoral studies, starting September 2002, in the department of Information and Computer Science at the University of California at Irvine, California.