Available online at www.sciencedirect.com
SCIENCE DIRECT®
Theoretical Computer Science 324 (2004) 147–160
Theoretical Computer Science
www.elsevier.com/locate/tcs

Linear time approximation schemes for vehicle scheduling problems

John E. Augustine[a,*], Steven Seiden[b]
[a] 444 CS Building, UCI, Irvine CA, USA
[b] Department of Computer Science, LSU, Baton Rouge, LA 70803, USA

Abstract

We consider makespan minimization for vehicle scheduling problems on trees with job requests that have release and handling times. 2-approximation algorithms were known for several variants of the single vehicle problem on a path. A $\frac{3}{2}$-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) that runs in time linear in the number of job requests for the single vehicle scheduling problem on trees that 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.

Keywords: Vehicle scheduling; Polynomial time approximation scheme; Dynamic programming; Approximation algorithm

1. Background

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 [17], scheduling flexible manufacturing systems [10], etc. MVSP is also equivalent to certain machine scheduling

* Corresponding author.
E-mail address: jea@ics.uci.edu (J.E. Augustine).

0304-3975/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2004.05.013

problems where there are costs for reconfiguring machines to perform different operations [2] and to power consumption minimization in CPU instruction scheduling [3]. Our main contribution is a polynomial time approximation scheme (PTAS) that runs in linear time for the MVSP.

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 $r_j \geqslant 0$ known as its *release time*. Job j requires a specific amount of time $h_j \geqslant 0$ 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 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 all the notations introduced in this paragraph are used throughout this paper.

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. 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) studied previously by Karuno et al. in [10], 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, denoted FO (fixed origin), is that each vehicle has a fixed origin.

Since even SVSP on a path is NP-hard [6,18], 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 [7]. 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

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

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 $\{\mathcal{A}_\varepsilon\}_{\varepsilon \geqslant 0}$ 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 [7] for a more comprehensive treatment of approximation algorithms.

Previous results: Psaraftis et al. [17] 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 [18] 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 $k \geqslant 2$ 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 [6]. If k is part of the input, then MVSP is strongly NP-complete. For paths, Karuno et al. develop a $\frac{3}{2}$-approximation algorithm for the RTO version of SVSP [14] and a 2-approximation for the version of MVSP where the origins and ending points are not pre-specified [10]. They also provide a solution to the ZMVSP whose maximum completion time is within twice that of the optimal. For SVSP on trees, [12] develop a 2-approximation algorithm. For SVSP on trees with zero handling times, Nagamochi et al. [15] 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 $\frac{5}{2}$-approximation algorithm. Karuno, Nagamochi and Ibaraki have provided some important hardness results to variants of the problems addressed in this paper. In [11] they show that the SVSP on a tree to minimize the maximum lateness is strongly NP-hard. In [13], they consider the TSP on a line with deadlines and general handling times and show that it is NP-complete when asked if there exists a schedule that meets the deadlines.

There is a large body of work on vehicle scheduling problems with different job requirements, metric spaces and objective functions. For instance, Nagamochi et al. [15] study the vehicle scheduling problem on a general graph, while Charikar et al. [3] 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. [5].

Our results: In this paper, we develop a PTAS that can be applied to many of the variants of SVSP. We begin in Section 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 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 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 4, we extend our algorithm to include ZMVSP using a dynamic programming approach. Essentially, this multiplies the running time by a factor of $O(n^t)$. We explore the relationship between the optimal zone and non-zone schedules. If multiple jobs can appear in the same position, then the

lower bound on the optimal cost ratio between the zone and non-zone schedules is k, but if jobs have to be in distinct positions the bound weakens to $2 - 1/t$. In Section 5, we show that an extension of SVSP to include deadlines is NP-hard, even when all release times are zero.

Note: Recently and independently, Karuno and Nagamochi [9] have also developed PTASs 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. Both solutions are exponential in the number of leaves.

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 $O(R(m+1)^{(R-1)t+1}n)$. 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. It is easy to use induction to show that the number of vertices with degree greater than 2 is at most $t-2$ (refer [1] for proof). Also, the vertices containing requests, which is at most n cannot be eliminated. Therefore we have $m \leqslant n + t - 2$.

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 $\pi(0) = 0$. For some problem instance σ, 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 $\pi(\ell) \leqslant i$ 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 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. Consider some schedule π. This proof works by taking the non-eager schedule π and successively modify it until it is converted to ϖ, the eager schedule. In each modification step, a job that is non-eagerly serviced is made eager. We then show that in a finite number of these steps, we attain ϖ. Define

$$e(i,\ell) = c_\pi(i-1) + d(p_{\pi(i-1)}, p_\ell),$$
$$f_\ell = \min\{i \mid \min(a_\ell, r_\ell) \leqslant e(i,\ell),\ p_\ell \in p_{\pi(i-1)} \rightsquigarrow p_{\pi(i)}\}.$$

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 $\min(a_\ell, r_\ell) \leqslant e(i,\ell)$ 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

$$\pi'(i) = \pi(i) \quad \text{for } 1 \leqslant i < f_L,$$
$$\pi'(f_L) = L,$$
$$\pi'(i) = \pi(i-1) \quad \text{for } f_L < i \leqslant q,$$
$$\pi'(i) = \pi(i) \quad \text{for } q < i \leqslant n.$$

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

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

Using this fact, it is easy to show by induction that $c_{\pi'}(i) \leqslant c_\pi(i)$ for $q < i \leqslant n$. 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 $0 \leqslant u_1 < \cdots < u_R$ 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 $0 \leqslant i \leqslant R$. 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 $|L_i| \leqslant t$ 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 $1 \leqslant j \leqslant |L_i|$, let X_i^j be the jth leaf visited by the vehicle during phase i in schedule π. For $|L_i| < j \leqslant t$, define $X_i^j = X_i^{|L_i|}$. If X_i is empty then we define $X_i^j = -1$ for $0 \leqslant j \leqslant t$. Define $X_0^t = p_0$.

We claim that the structure of π is completely defined by X_i^j for $1 \leqslant j \leqslant t$, $0 \leqslant i \leqslant R$. 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 sweeps per phase are sufficient. If we sweep from X_{i-1}^t to X_i^1 serving X_i^0 on the way, 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 $1 \leqslant j \leqslant t$, $1 \leqslant i \leqslant R-1$ 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 $1 \leqslant i < R$ there are $m^t + 1$ possible choices for $X_i^0, \ldots, X_i^t$. Therefore, the total number of possible schedules is at most $(m+1)(m^t+1)^{R-1} \leqslant (m+1)^{t(R-1)+1}$, 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 $(m+1)^{t(R-1)+1}$, calculating the cost for each, and return the minimum cost schedule.

The calculation of the cost of a schedule, given X_i^j for $1 \leqslant j \leqslant t$, $0 \leqslant i \leqslant R$, 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.

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 $r_{\max} = \max_{1 \leqslant i \leqslant n} r_i$. 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 $\text{cost}(\sigma) \geqslant r_{\max}$, we have $\delta \leqslant \varepsilon\,\text{cost}(\sigma)/2$.

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, $\text{cost}(\sigma) \geqslant P$ and therefore $\Delta \leqslant \varepsilon\,\text{cost}(\sigma)/(4(t+1)a)$.

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 Fig. 1 on page 8. 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

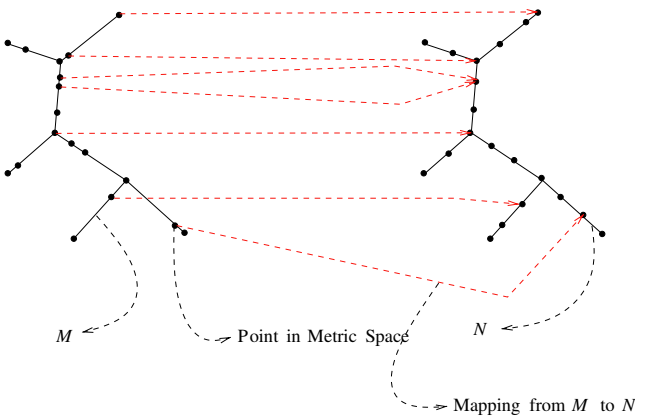

Fig. 1. Mapping of points from $\mathcal{M}$ to $\mathcal{N}$.

leaves or junctions and other vertices are of degree 2. $\mathcal{M}$ has a unique decomposition into a set E of at most $2t-3$ essential paths (Proof provided in [1]). Note that the tree formed by preserving the junctions and leaves of $\mathcal{M}$ and using the corresponding essential paths as edges is topologically similar to $\mathcal{M}$. We find this decomposition E 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 $x'(v) = \min\{|p|, \Delta\lfloor x(v)/\Delta + \frac{1}{2}\rfloor\}$. 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

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

Using $\mathcal{N}$, we define two new problem instances $\sigma^{\uparrow}$ and $\sigma^{\downarrow}$. Fig. 2 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 by downward directed arrows in Fig. 2) is defined in

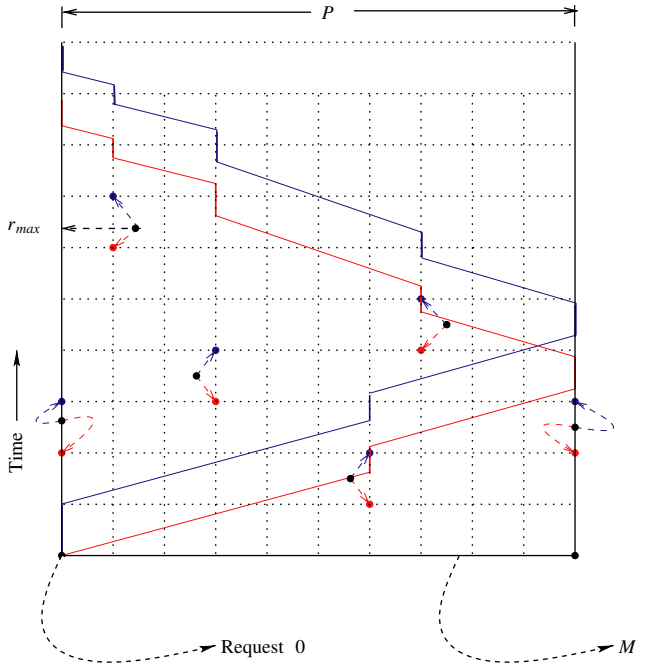

Fig. 2. Relating $\sigma^{\downarrow}$ and $\sigma^{\uparrow}$.

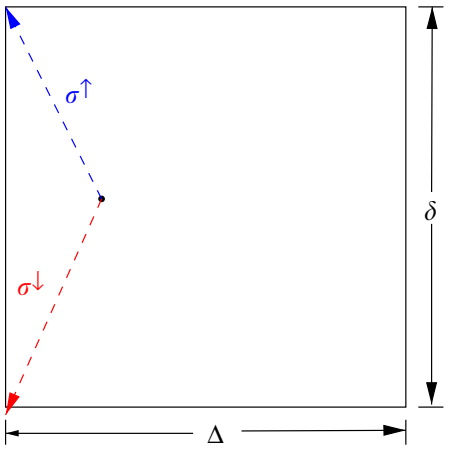

Fig. 3. Magnified view of unit temporal and spatial grid.

terms of σ as follows:

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

for $1 \leqslant i \leqslant n$. 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 by upward directed arrows in Fig. 2) is defined by $r_i^{\uparrow} = r_i^{\downarrow} + \delta$, $p_i^{\uparrow} = p_i^{\downarrow}$ and $h_i^{\uparrow} = h_i$ for $1 \leqslant i \leqslant n$. 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 $O(n(a+1)(b+2t-2)^{ta+1}) = O(n(8(t+1)\lceil 1/\varepsilon\rceil^2 + 2t - 2)^{2t\lceil 1/\varepsilon\rceil+1}/\varepsilon)$, which is linear in n for constant t and ε.

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

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

for $1 \leqslant i \leqslant n$. We first observe that clearly, $\text{cost}(\sigma) \geqslant \text{cost}(\sigma^*)$. The optimal schedule π^* for σ^* has the structure that we have explained in the preceding section; i.e. at most $t+1$ sweeps are contained in each phase. We consider an extra sweep from the previously mentioned t sweeps because the sweeps contained in phase i are X_i^0 to X_i^1, X_i^1 to $X_i^2, \ldots, X_i^t$ to X_{i+1}^0 . 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, $\text{cost}(\sigma^*) + (t+1)a\Delta \geqslant \text{cost}(\sigma^{\downarrow})$. We conclude that the cost incurred by the algorithm is at most

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

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 [10] 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 $1 \leqslant i \leqslant j \leqslant n$ to be the optimal cost for serving requests $i, \ldots, j$ using a single vehicle. Further define $x^*(i,\ell)$ for $1 \leqslant i \leqslant n$ and $1 \leqslant \ell \leqslant k$ 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:

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

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 $1 \leqslant i \leqslant j \leqslant n$ to be the cost incurred by $\mathcal{A}$ for serving requests $i, \ldots, j$ with a single vehicle. Define $x(i,\ell)$ for $1 \leqslant i \leqslant n$ and $1 \leqslant \ell \leqslant k$ 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

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

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

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

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

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

(1) Calculate the values $C(i,j)$ for $1\leqslant i\leqslant j\leqslant n$, 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 (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 $\mathcal{M}$. The total number of subtrees is at most n^t (refer [1]), so the time to do this is $O(n^t g(n))$.

Note that any tree that is partitioned into two or more zones will have at least two zones such that the removal of either one will not disconnect the rest of the tree and we designate them *non-disconnecting zones*. This can be shown easily by induction with the obviously true base case being any tree T split into two zones. Now assume that any tree T partitioned into some i zones, $2\leqslant i<n$, has at least two non-disconnecting zones. Now consider any tree T with n zones. Pick a zone that disconnects (note that if this cannot be done, then it follows that there are at least two non-disconnecting zones). Each of these disconnected parts should have atleast one zone that does not disconnect the original tree T. Note also that a non-disconnecting zone will be the tree formed by some node and its descendants.

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. In each step of the dynamic programming, we try to find an optimal non-disconnecting zone. 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 root. The DFS should follow the rule that it visits the children of each node before its parents with respect to the root. This ensures

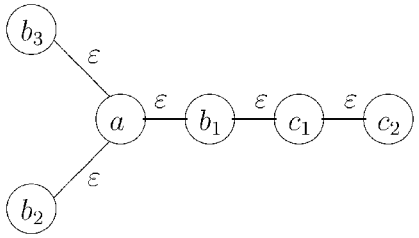

Fig. 4. Tree for the bad instance with $t=3$ and $k=4$.

that atleast one DFS visits the subtree of the descendants of each node in T before visiting its parent. Each time the depth first search has completed the descendants of a node and the next step would be to visit its parent, 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 $x(T,\ell)$ is $O(tn)$, since T has at most t leaves. From [1], 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 $1\leqslant j\leqslant k$. 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. *For all $k\geqslant 2$ and $t\geqslant 2$, 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 $1\leqslant i\leqslant t$, from b_1 to c_1 and from c_i to c_{i+1} for $1\leqslant i\leqslant k-3$. There are no other edges. An example is shown in Fig. 4 on page 13. 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. □

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 [18] 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 $1 \leqslant j \leqslant n$. 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 [18] shows that this problem is NP-hard, even when $\mathcal{M}$ is a path. Nagamochi et al. [15] 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 $1 \leqslant j \leqslant n$. 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 $p'_j = p_j$, $h'_j = h_j$, and $d'_j = D - r_j$ for $1 \leqslant j \leqslant n$. 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.

6. Conclusion

We have presented the first approximation schemes for single and multiple vehicle scheduling problems on trees. Such problems are well motivated, having a large number of applications [10]. We believe that this is just an initial step in the exploration of such problems, and therefore we list some open problems that we feel are important: Karuno and Nagamochi [10] 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?

References

[1] J.E. Augustine, Offline and online variants of the traveling salesman problem, Masters Thesis, Louisiana State University, 2002.

[2] J. Bruno, P. Downey, Complexity of task sequencing with deadlines, set-up times and changeover costs, SIAM J. Comput. 7 (4) (1978) 393–404.
[3] M. Charikar, R. Motwani, P. Raghavan, C. Silverstein, Constrained TSP and low-power computing, in: Proc. 5th Internat. Workshop on Algorithms and Data Structures, (1997) 104–115.
[4] N. Christofides, Vehicle routing, The Traveling Salesman Problem, Wiley, New York, 1985, 431–448.
[5] J. Desrosiers, Y. Dumas, M. Solomon, F. Soumis, Time constrained routing and scheduling, in: M.O. Ball, T.L. Magnanti, C.L. Monma, G.L. Nemhauser (Eds.), Network Routing, Handbooks in Operations Research and Management Science, Vol. 8, Elsevier, Amsterdam, 1995.
[6] M.R. Garey, D.S. Johnson, Computers and Intractability: a Guide to the Theory of NP-completeness, Freeman and Company, San Francisco, 1979.
[7] D. Hochbuam, Approximation Algorithms for NP-hard Problems, PWS Publishing Company, 1997.
[8] R.M. Karp, Reducibility among Combinatorial Problems, Plenum Press, New York, 1972, pp. 85–103.
[9] Y. Karuno, H. Nagamochi, A polynomial time approximation scheme for the multi-vehicle scheduling problem on a path with release and handling times, in: Proc. 12th Internat. Sympos. Algorithms and Comput. (2001) 36–47.
[10] Y. Karuno, H. Nagamochi, 2-approximation algorithms for the multi-vehicle scheduling problem on a path with release and handling times, Discrete Appl. Math. 129 (2003) 433–447.
[11] Y. Karuno, H. Nagamochi, T. Ibaraki, Vehicle scheduling on a tree to minimize maximum lateness, J. Oper. Res. Soc. Japan 39 (3) (1996) 345–355.
[12] Y. Karuno, H. Nagamochi, T. Ibaraki, Vehicle scheduling on a tree with release and handling times, Ann. Oper. Res. 69 (1997) 193–207.
[13] Y. Karuno, H. Nagamochi, T. Ibaraki, Computational complexity of the traveling salesman problem on a line with deadlines and general handling times, Memoirs of the Faculty of Engineering and Design, Kyoto Institute of Technology, Vol. 45, 1997, pp. 19–22.
[14] Y. Karuno, H. Nagamochi, T. Ibaraki, A 1.5-approximation for single-vehicle scheduling problem on a line with release and handling times, in: Japan—USA Symp. on Flexible Automat., July 1998, pp. 1363–1366.
[15] H. Nagamochi, K. Mochizuki, T. Ibaraki, Complexity of the single vehicle scheduling problem on graphs, INFOR: Inform. Systems Oper. Res. 35 (4) (1997) 256–276.
[16] C.H. Papadimitriou, The Euclidean traveling salesman problem is NP-complete, Theor. Comput. Sci. 4 (3) (1977) 237–244.
[17] H. Psaraftis, M. Solomon, T. Magnanti, T. Kim, Routing and scheduling on a shoreline with release times, Manage. Sci. 36 (2) (1990) 212–223.
[18] J. Tsitsiklis, Special cases of traveling salesman and repairman problems with time windows, Networks 22 (1992) 263–282.