An approximability result of the multi-vehicle scheduling problem on a path with release and handling times

Yoshiyuki Karuno\textsuperscript{a,}\textsuperscript{*}, Hiroshi Nagamochi\textsuperscript{b}

\textsuperscript{a}Department of Mechanical and System Engineering, Kyoto Institute of Technology, Matsugasaki, Sakyo, Kyoto 606-8585, Japan
\textsuperscript{b}Department of Information and Computer Sciences, Toyohashi University of Technology, Toyohashi, Aichi 441-8580, Japan

Received 17 March 2003; received in revised form 22 August 2003; accepted 10 September 2003
Communicated by G. Ausiello

Abstract

In this paper, we consider a scheduling problem of vehicles on a path $G$ with $n$ vertices and $n-1$ edges. There are $m$ identical vehicles. Each vertex in $G$ has exactly one job. Any of the $n$ jobs must be processed by some vehicle. Each job has a release time and a handling time. With the edges, symmetric travel times are associated. The problem asks to find an optimal schedule of the $m$ vehicles that minimizes the maximum completion time of all the jobs. The problem is known to be NP-hard for any fixed $m \geq 2$. In this paper, we show that the problem with a fixed $m$ admits a polynomial time approximation scheme. Our algorithm can be extended to the case where $G$ is a tree so that a polynomial time approximation scheme is obtained if $m$ and the number of leaves in $G$ are fixed.

Keywords: Discrete optimization; Vehicle scheduling; Polynomial time approximation scheme; Dynamic programming

1. Introduction

The vehicle scheduling problem (VSP, for short) consists of a set of $n$ jobs (such as items to be picked up or facilities to be inspected) which are located at different
vertices in a given graph. Each job is characterized by a release time, a handling time and a deadline (or a due date). There are \( m (1 \leq m \leq n) \) identical vehicles on the graph to process the jobs. When a vehicle traverses an edge, it takes a travel time associated with the edge. The travel times are given as edge weights of the graph. Each job must be processed by exactly one vehicle. The problem asks to find a schedule of the \( m \) vehicles that minimizes (or maximizes) a specified objective function, such as the maximum completion time of jobs, the total completion time of all jobs, the maximum lateness from due dates, the number of tardy jobs, and so on. The VSP is an important topic encountered in a variety of industrial and service sector applications.

In this paper, when given graphs are restricted to paths (resp., trees), we denote the VSP by VSP-PATH (resp., VSP-TREE). The VSP with \( m = 1 \) is particularly called the single-vehicle scheduling problem. We refer to the VSP-PATH (resp., VSP-TREE) with \( m = 1 \) as the 1-VSP-PATH (resp., 1-VSP-TREE). The VSP-PATH treated in this paper has release and handling times, but no finite deadlines (i.e., deadlines are considered to be sufficiently large for all jobs). The objective is to minimize the maximum completion time of jobs.

It should be noted that if all edge weights in a given path (i.e., travel times of the vehicles) are zero, then the VSP-PATH is identical to the parallel machine scheduling problem with release time constraints to minimize the maximum completion time. According to the traditional notation for machine scheduling problems studied by Graham et al. [9], this machine scheduling problem is denoted by \( P/rj/C_{\text{max}} \).

When \( m = 1 \), problem \( P/rj/C_{\text{max}} \) becomes the single-machine scheduling problem denoted by \( 1/rj/C_{\text{max}} \). It can be solved in \( O(n \log n) \) time where jobs are scheduled in a non-decreasing order of release times. However, in the 1-VSP-PATH, a slight change in an order of processing jobs may affect the maximum completion time more dramatically due to the travel times. In fact, Tsitsiklis [21] proved the NP-hardness of the 1-VSP-PATH, which indicates that introducing travel times distinguishes the computational complexity of the 1-VSP-PATH from that of \( 1/rj/C_{\text{max}} \). Psaraftis et al. [20] showed that the 1-VSP-PATH is 2-approximable, and that, if all handling times are zero, then the 1-VSP-PATH can be solved in \( O(n^2) \) time. Afterward, Young and Chan [22] proved that the 1-VSP-PATH with general deadlines can be solved in \( O(n^2) \) time if all jobs have a common release time and all handling times are zero.

When \( m \geq 2 \), problem \( P/rj/C_{\text{max}} \) is NP-hard in the strong sense since it contains 3-PARTITION as a special case, and the problem for any fixed \( m \geq 2 \) is even NP-hard since it contains PARTITION (e.g., see [7]). Hall and Shmoys [11] observed that there exist a 2-approximation algorithm and a polynomial time approximation scheme for problem \( P/rj/C_{\text{max}} \) (but the running times are not available in their paper). Therefore, the VSP-PATH is NP-hard in the strong sense for \( m \) arbitrary, since it can be viewed as a generalization of \( P/rj/C_{\text{max}} \). For the similar reason, the VSP-PATH is NP-hard even for any fixed \( m \geq 2 \). However, the VSP-PATH or the VSP-TREE has a more intractable situation. In an optimal schedule for an instance of the VSP-TREE with \( m \geq 2 \), some edges may not be traversed by any vehicle. Such an edge is called a gap. This makes difficult for us to derive a lower bound on the total travel times. The first constant factor approximation algorithm to the VSP-PATH with \( m \geq 2 \) has been obtained by Karuno and Nagamochi [12]. They first designed a 2-approximation algorithm for the
problem of finding an optimal gapless schedule (i.e., the one with the minimum of the maximum completion time among all schedules with no gaps), and applied the result to the general case to obtain an $O(mn^2)$ time 2-approximation algorithm and a nearly linear time $(2 + \varepsilon)$-approximation algorithm for any fixed $\varepsilon > 0$ [12].

In this paper, we consider the VSP-PATH from a different viewpoint of approximability. A polynomial time approximation scheme to the VSP-TREE is a family of algorithms $\{A_{\varepsilon}\}$ such that for any $\varepsilon > 0$, $A_{\varepsilon}$ delivers a schedule with the maximum completion time at most $(1 + \varepsilon)$ times the optimal. By noting that the approximability of the bin-packing problem, which can be regarded as a special case of the VSP-TREE in a star, has been well studied (e.g., see [2]), investigating the approximability of the VSP-TREE is a theoretically important issue to categorize approximation classes of related NP-hard problems. Recently, Augustine and Seiden [1] proposed a polynomial time approximation scheme to the 1-VSP-TREE if a given tree has a fixed number of leaves. The running time is bounded by a linear function in $n$ (but by an exponential in $1/\varepsilon$). This can be extended to a polynomial time approximation scheme to the VSP-PATH of finding an optimal zone schedule (i.e., the one with the minimum of the maximum completion time among all schedules with $m - 1$ gaps) [1]. However, this is not a polynomial time approximation scheme for obtaining the optimal attained by general schedules.

In this paper, we show that the VSP-PATH with a fixed number $m$ of vehicles and symmetric edge weights admits a polynomial time approximation scheme whose running time is bounded by a polynomial in $n$, but by an exponential in $1/\varepsilon$. The presented approximation scheme is based on the approximation of the problem by rounding given release times, and on the fact that any schedule with $\lambda$ gaps consists of $(\lambda + 1)$ gapless schedules on subpaths of a given path. The scheme is a two-fold dynamic programming. One is for computing an optimal schedule to the problem with rounded release times, and the other for finding the best schedule to the original problem by combining several gapless schedules over all choices of gaps on the path. Our algorithm can be extended to the VSP-TREE so that a polynomial time approximation scheme is obtained if $m$ and the number of leaves in a given tree are fixed.

Since paths and trees are important network topologies from both practical and graph theoretical views, the 1-VSP-PATH and the 1-VSP-TREE with respect to different objective functions have been studied (e.g., see [8,13–16,18,19]). Related problems on path and tree network topologies such as the traveling salesmen problem (TSP) and the delivery men problem (DMP) have also been considered (e.g., see [3–6]).

The remainder of this paper is organized as follows. In Section 2, we provide a mathematical description of the VSP-PATH. In Section 3, we explain a basic property of the VSP-PATH with a fixed number of distinct release times, which is obtained by rounding given release times, and present a dynamic programming to the restricted VSP-PATH. In Section 4, we discuss a polynomial time approximation scheme to the VSP-PATH of finding an optimal gapless schedule, and in Section 5, we present a polynomial time approximation scheme to the original problem. We also mention that our algorithm can be extended to the VSP-TREE, showing that a polynomial time approximation scheme exists if $m$ and the number of leaves in a tree are fixed. Finally, in Section 6, we give some concluding remarks.
2. Problem description

2.1. Instance and solution

The VSP-PATH is formulated as follows. Let $G=(V,E)$ be a path which consists of a set $V = \{v_1,v_2,\ldots,v_n\}$ of $n$ vertices and a set $E = \{\{v_j,v_{j+1}\} \mid j = 1,2,\ldots,n-1\}$ of $n-1$ edges. In this paper, we assume that vertex $v_1$ is the left end of $G$, and $v_n$ the right end of it. There is a job $j$ at each vertex $v_j \in V$. The job set is denoted by $J = \{j \mid j = 1,2,\ldots,n\}$. There are $m$ vehicles on $G$ ($1 \leq m \leq n$), which are assumed to be identical. Each job must be processed by exactly one vehicle.

The travel time for a vehicle to traverse edge $\{v_j,v_{j+1}\} \in E$ from $v_j$ to $v_{j+1}$ (resp., from $v_{j+1}$ to $v_j$) is $w(v_j,v_{j+1}) \geq 0$ (resp., $w(v_{j+1},v_j) \geq 0$). Edge weight $w(v_j,v_{j+1})$ for $\{v_j,v_{j+1}\} \in E$ is called symmetric if $w(v_j,v_{j+1}) = w(v_{j+1},v_j)$ holds. In this paper, we assume that all edge weights are symmetric. The travel time for a vehicle to move from vertex $v_i$ to vertex $v_j$ on $G$ is the sum of edge weights belonging to the unique path from $v_i$ to $v_j$. Each job $j \in J$ has a release time $r_j \geq 0$ and a handling time $h_j \geq 0$. That is, a vehicle cannot start processing job $j$ before time $r_j$, and it takes $h_j$ time units to process job $j$ (no interruption of the processing is allowed). A vehicle at vertex $v_j$ may wait until time $r_j$ to process job $j$, or move to other vertices without processing job $j$ if it is more advantageous (in this case, the vehicle must come back to $v_j$ later to process job $j$, or another vehicle must come to $v_j$ to process it). An instance of the problem VSP-PATH is denoted by $(G,r,h,w,m)$.

An entire schedule is completely specified by a set of $m$ sequences of jobs $\pi = \{\pi^{[1]},\pi^{[2]},\ldots,\pi^{[m]}\}$, each sequence in which is denoted by $\pi^{[p]} = (j_{1}^{[p]},j_{2}^{[p]},\ldots,j_{n_{p}}^{[p]})$, $p = 1,2,\ldots,m$, where $n_{p}$ is the number of jobs to be processed by vehicle $p$ (hence, it holds that $\sum_{p=1}^{m} n_{p} = n$) and $j_{i}^{[p]}$ is its $i$th job. Vehicle $p$ is initially situated at vertex $v_{j_{i}^{[p]}}$, and hence it starts processing job $j_{1}^{[p]}$ at time $\max\{0,r_{j_{1}^{[p]}}\}$ (after completing $j_{1}^{[p]}$, the vehicle moves to vertex $v_{j_{2}^{[p]}}$ to process job $j_{2}^{[p]}$, and so on, until it completes the last job $j_{n_{p}}^{[p]}$). The completion time of vehicle $p$ (i.e., the working time of it) is defined as the completion time of its last job $j_{n_{p}}^{[p]}$, which is denoted by $C(\pi^{[p]})$. The objective is to find a $\pi$ that minimizes the maximum completion time of all the jobs, i.e.,

$$C_{\text{max}}(\pi) = \max_{1 \leq p \leq m} C(\pi^{[p]}).$$

In this paper, we denote by $\pi^*$ an optimal schedule and by $C_{\text{max}}^*$ the minimum of the maximum completion time $C_{\text{max}}(\pi^*)$.

2.2. Subpath and subinstance

Let $V(i,j)$ denote $\{v_i,v_{i+1},\ldots,v_j\} (\subseteq V)$, where $i \leq j$. Define $G(i,j) = (V(i,j),E(i,j))$ be a subpath of a given path $G = (V,E)$ induced by $V(i,j)$, where $E(i,j) = \{(v_j,v_{j+1}) \mid j' = i,i+1,\ldots,j-1\} (\subseteq E)$. Let $J(i,j) = \{i,i+1,\ldots,j\} (\subseteq J)$ be the corresponding subset of jobs on the subpath $G(i,j)$. In particular, $G(1,n) = G$ and $J(1,n) = J$. 
We consider the scheduling problem of \( v (\leq m) \) vehicles on \( G(i,j) \), where release and handling times, and edge weights remain unchanged. This scheduling problem is called a subinstance of the original instance \((G, r, h, w, m)\), and is denoted by \((G(i,j), r, h, w, v)\) (hence, the original instance is denoted by \((G, r, h, w, m)\) as well as \((G(1,n), r, h, w, m)\)).

2.3. Zone schedule and gapless schedule

For a schedule \( QEM \), we refer to a maximal subpath \( G(i,j) \) of a given path which is traversed by a certain vehicle as its zone (hence, the vehicle processes jobs \( i \) and \( j \), but there may be some jobs \( j' \) \((i < j' < j)\) processed by other vehicles).

A feasible schedule \( QEM \) using \( m' \) vehicles \((m' \leq m)\) is referred to as a zone schedule if any two zones in \( QEM \) do not intersect and thus there are \( m' - 1 \) edges which are not traversed by any vehicle. Such an edge that is not traversed by any vehicle is called a gap. A schedule \( QEM \) is called gapless if each edge \( \{v_j, v_{j+1}\} \in E \) is traversed at least once (either from \( v_j \) to \( v_{j+1} \) or from \( v_{j+1} \) to \( v_j \)) by some vehicle. Define

\[
Z = W + H,
\]
where \( W = \sum_{j=1}^{n-1} w(v_j, v_{j+1}) \) and \( H = \sum_{j=1}^{n} h_j \).

3. With a fixed number of distinct release times

In this section, we provide a dynamic programming approach to the VSP-PATH with a fixed number of distinct release times, based on a similar notion to that in [17]. The distinct release times are obtained by rounding given release times. We explain in Section 4 how to round given release times. The dynamic programming becomes a basis of our polynomial time approximation scheme proposed in Section 5.

3.1. A basic property

We restrict our attention to the VSP-PATH that has a fixed number of distinct release times. Let \( 0 \leq \rho_1 < \rho_2 < \cdots < \rho_K \) be the \( K \) distinct release times. Define \( \rho_{K+1} = \infty \). For a feasible schedule \( \pi \), the set of jobs \( j \) that are processed by vehicle \( p \) at the starting times \( \sigma_j \) with \( \rho_j \leq \sigma_j < \rho_{j+1} \) is referred to as the \( k \)th interval set of vehicle \( p \), and is denoted by \( J_{p,k}(\pi) \) (if the schedule is obvious from the context, the \( \pi \) may be omitted). Note that a job \( j \) can belong to the \( k \)th interval set only if \( r_j \leq \rho_k \), since there is no other release time between \( \rho_k \) and \( \rho_{k+1} \).

We say that a vehicle \( p \) processes a set \( J' = \{j_1, j_2, \ldots, j_l\} \) \((j_1 < j_2 < \cdots < j_l)\) of jobs in a \( 1 \)-way if jobs in \( J' \) are processed in the order of \( j_1, j_2, \ldots, j_l \) or \( j_l, j_{l-1}, \ldots, j_1 \). The following observation is a key property to reduce the complexity for finding a schedule for each vehicle to minimize the travel cost.

**Lemma 1.** For the VSP-PATH with \( K \) distinct release times, there is an optimal schedule \( \pi \) such that for each interval set \( J_{p,k}(\pi) \neq \emptyset \), the jobs in \( J_{p,k}(\pi) \) are processed in a \( 1 \)-way.
Proof. Let \( \pi \) be an optimal schedule. Consider an interval set \( J_{p,k}(\pi) = \{ j_1, j_2, \ldots, j_i \} \) \((j_1 < j_2 < \cdots < j_i)\). Suppose that job \( j_1 \) is processed before job \( j_i \) (the other case can be treated in a symmetric way). Thus, vehicle \( p \) moves from \( j_1 \) to \( j_i \) on the subpath \( G(j_1, j_i) \) during \( \pi \). Since all release times of jobs in \( J_{p,k}(\pi) \) are no more than \( \rho_k \), the completion time of the schedule of vehicle \( p \) never increases by changing the processing order of jobs in \( J_{p,k}(\pi) \) into the order \( j_1, j_2, \ldots, j_i \). This proves the lemma.

In a schedule \( \pi \), when an interval set \( J_{p,k}(\pi) = \{ j_1, j_2, \ldots, j_i \} \) \((j_1 < j_2 < \cdots < j_i)\) of jobs is processed in a 1-way, the schedules \( \pi^{\{p\}}_k = (j_1, j_2, \ldots, j_i) \) and \( \mu^{\{p\}}_k = (j_i, j_{i-1}, \ldots, j_1) \) are called 1-way subschedules. We denote the total of traveling and handling times in these subschedules by \( T(\pi^{\{p\}}_k) \) and \( T(\mu^{\{p\}}_k) \), respectively. That is, \( T(\pi^{\{p\}}_k) = h_{j_1} + \sum_{i=2}^i (w_{v_{j_i-1}, v_{j_i}} + h_{j_i}) \) and \( T(\mu^{\{p\}}_k) = h_{j_i} + \sum_{i=1}^{i-1} (w_{v_{j_i+1}, v_{j_i}} + h_{j_i}) \), where \( T(\pi^{\{p\}}_k) = T(\mu^{\{p\}}_k) \) holds due to the symmetry of edge weights.

3.2. Dynamic programming

For the VSP-PATH with \( K \) distinct release times, there are at most \( K^n \) assignments of \( n \) jobs to \( K \) interval sets, and at most \( m^n \) assignments of \( n \) jobs to \( m \) vehicles. We call each of the \( K^n m^n \) ways of assigning \( n \) jobs to \( K \) interval sets and \( m \) vehicles as a job assignment. By Lemma 1, if we know the assignment of \( n \) jobs to all interval sets \( J_{p,k}(\pi) \) that induces an optimal schedule \( \pi \), the schedule of each vehicle \( p \) in \( \pi \) can be constructed from such an assignment by choosing the best way of concatenating 1-way subschedules \( \pi^{\{p\}}_k \) or \( \mu^{\{p\}}_k \) of \( J_{p,k}(\pi) \) for all \( k = 1, 2, \ldots, K \) (some of which may be empty schedules). Since there are \( O(2^K) \) ways of concatenations for each vehicle, choosing the best schedule among all these possible cases would take \( \Omega(2^K n m^n) \) time to find an optimal schedule.

We reduce the time complexity by using a dynamic programming. In the rest of this section, we assume that all handling times and all edge weights are integers (we discuss in Section 4 how to round given handling times and edge weights into integers). Now \( Z \) in (2) is an integer.

Again by Lemma 1, for a fixed job assignment, an optimal schedule of each vehicle \( p \) obeying the job assignment can be constructed by concatenating 1-way subschedules for all interval sets. During the concatenation, if the starting time of the last job in the \( k \)-interval becomes equal to or larger than \( \rho_{k+1} \) due to the finishing time of the previous job, then we can abort the construction for the job assignment since it violates the definition of interval sets. We call such a job assignment violating. Notice that, in this computation for constructing an optimal schedule of vehicle \( p \), we only need to compute the least time to resume the first job in the \((k+1)\)-interval set after deciding the finishing time of the last job in the \( k \)-interval set for each \( k = 1, 2, \ldots, K - 1 \). For this, it suffices to maintain the total of traveling and handling times \( T_{p,k} = T(\pi^{\{p\}}_k) = T(\mu^{\{p\}}_k) \) of two 1-way subschedules to process all jobs in each interval set \( J_{p,k} \), and the first and last jobs to be processed in the interval set \( J_{p,k} \) (the set of jobs in each interval set is not necessarily stored as long as \( T_{p,k} \) for all \( p \) and \( k \) are stored and the \( n \) jobs...
are assumed to be correctly assigned according to a job assignment). By maintaining these, concatenating $K$ 1-way subschedules for vehicle $p$ can be done in $O(K)$ time, and thus the completion time of each vehicle obeying a job assignment can be computed in $O(2^K K)$ time since there are $O(2^K)$ possible ways of concatenations for each vehicle.

To facilitate the above computation, we define a table $X^{(n)}$ for $(G(1, n), r, h, w, m)$ as the following $3Km$-tuple $X$:

$$
X = (T_{p,1}, T_{p,2}, \ldots, T_{p,k}; T_{1,1}, T_{1,2}, \ldots, T_{1,K}; T_{2,1}, T_{2,2}, \ldots, T_{2,K}; \ldots; T_{m,1}, T_{m,2}, \ldots, T_{m,k};
$$

$$
L_{11}, L_{12}, \ldots, L_{1K}; L_{21}, L_{22}, \ldots, L_{2K}; \ldots; L_{m1}, L_{m2}, \ldots, L_{mk};
$$

$$
R_{11}, R_{12}, \ldots, R_{1K}; R_{21}, R_{22}, \ldots, R_{2K}; \ldots; R_{m1}, R_{m2}, \ldots, R_{mk}),
$$

where $L_{p,k}$ (resp., $R_{p,k}$) denotes the index of the left (resp., right) end vertex of the zone of $\pi_k^{[p]}$ (which is also the zone of $\mu_k^{[p]}$), implying that $L_{p,k}$ and $R_{p,k}$ are, respectively, the first and last jobs to be processed during $\pi_k^{[p]}$ while they are, respectively, the last and first jobs during $\mu_k^{[p]}$. If no job is assigned to the $k$th interval of vehicle $p$, we set $T_{p,k} = 0$ and $L_{p,k} = R_{p,k} = 0$. As stated above, from such a table $X^{(n)}$, an optimal value of each vehicle $p$ obeying the job assignment which produces the table can be constructed by concatenating 1-way subschedules by choosing one of $\pi_k^{[p]}$ and $\mu_k^{[p]}$ for each $k$, taking $O(2^K K)$ time to find the best way of concatenation. We here assume that a table $X^{(n)}$ has been constructed from a job assignment. We remark that different job assignments may produce the same table $X^{(n)}$. This reduces the number of different tables to at most $(2 + 1)^{K(n + 1)}^{K(n + 1)^m}$ since $T_{p,k} \leq Z$, $L_{p,k} \leq n$ and $R_{p,k} \leq n$.

Now consider how to compute the set $\{X^{(j)}\}$ of such tables. Let $\mathcal{X}^{(j)}$ denote the set of tables $X^{(j)}$ defined for the subinstance $(G(1, j), r, h, w, m)$. Suppose that the set $\mathcal{X}^{(j-1)}$ of tables has been computed. We add the $j$th job to the $k$th interval set of vehicle $p$ for all $p$ and $k$ with $r_j \leq \rho_k$, and update each $X \in \mathcal{X}^{(j-1)}$ into a table in $\mathcal{X}^{(j)}$ by computing

$$
T_{p,k} := T_{p,k} + w(v_{R_{p,k}}, v_j) + h_j, \quad R_{p,k} := j,
$$

where we extend $w(u, v)$ to the path length between $u$ and $v$, and we let $L_{p,k} := j$ if $L_{p,k} = 0$ and use $w(v_{R_{p,k}}, v_j) = 0$ if $R_{p,k} = 0$. We denote by $X_{p,k}$ the updated table. Supposing that $w(u, v)$ for all $u, v \in V$ have been computed, table $X_{p,k}$ can be obtained in $O(1)$ time for each $p$ and $k$.

We are now ready to describe an algorithm for computing $\mathcal{X}^{(1)}, \mathcal{X}^{(2)}, \ldots, \mathcal{X}^{(n)}$ by a dynamic programming and for constructing an optimal schedule from $\mathcal{X}^{(n)}$.

**Algorithm $A'$**.

Step 1 (Initialization): Set $\mathcal{X}^{(0)}$ to be $\{(0, 0, \ldots, 0; \ldots; 0, 0, \ldots, 0)\}$, which contains only a $3Km$-tuple with zero entries.

Step 2 (Generation of $\mathcal{X}^{(1)}, \mathcal{X}^{(2)}, \ldots, \mathcal{X}^{(n)}$): For $j = 1, 2, \ldots, n$, perform the following computations:

(i) Initialize $\mathcal{X}^{(j)}$ by $\mathcal{X}^{(j)} := \emptyset$.

(ii) For each table $X \in \mathcal{X}^{(j-1)}$, $k = 1, 2, \ldots, K$ and $p = 1, 2, \ldots, m$,
we add job $j$ to the $k$th interval set of vehicle $p$ in the $X$ and update $X$ into $X_{p,k}$ according to (3):

$$X^{(j)} := X^{(j)} \cup \{X_{p,k}\} \text{ if } X_{p,k} \notin X^{(j)}.$$ 

**Step 3 (Determination of an optimal schedule):** For each table $X \in \mathcal{A}^{(n)}$, compute the maximum completion time of the corresponding schedule, where we discard table $X$ if it turned out to be violating during the computation (recall that the completion time of each vehicle can be obtained in $O(2^K)$ time, and hence the maximum completion time $C_X$ of each table $X \in \mathcal{A}^{(n)}$ can be computed in $O(2^K)$ time). The minimum of the maximum completion times $C_X$ over all tables $X \in \mathcal{A}^{(n)}$ is the optimal value to the problem, and an optimal schedule can be constructed by backtracking.

The time complexity of this algorithm is evaluated as follows. As already observed, $|\mathcal{A}^{(j-1)}| \leq ((n+1)(Z+1))^{Km}$, $j = 2, \ldots, n$. In Step 2(ii), we need to test whether $\mathcal{A}^{(j)}$ already contains a table which is identical with the table $X_{p,k}$ just generated. By preparing a complete set $M$ of all possible tables of $3Km$-tuples, where $M$ records which table $X$ belongs to the current $\mathcal{A}^{(j)}$, we can answer this query in $O(1)$ time and $O((n^2Z)^{Km})$ space. (More precisely, we encode each $3Km$-tuple $X$ into an entry of a $3Km$-dimensional array $M$ in such a way that $X$ belongs to if and only if the corresponding entry has 1. We can access the encoded entry from $X$ in $O(1)$ time provided that the index for each dimension of $M$ is specified, which usually takes $O(Km)$ time to prepare. However, we know the set of indices of a table $X$ before generating $X_{p,k}$, and we only need to change at most three indices, those for $T_{p,k}$, $R_{p,k}$ and $L_{p,k}$, among all $3Km$ indices for $M$. Therefore, we can answer the query in $O(1)$ time.) Hence an iteration of Step 2 takes $O(Km(n^2Z)^{Km})$ time. Step 2 repeats $n$ times, and Step 3 takes $O(2^K Km(n^2Z)^{Km})$ time for computing the minimum of the maximum completion times over all tables $X \in \mathcal{A}^{(n)}$. Therefore, the time complexity of algorithm $A'$ is at most

$$O(Km(n + 2^K)(n^2Z)^{Km}).$$

(4)

Note that this is a pseudopolynomial time if the number of vehicles $m$ and the number of release times $K$ are constant.

### 4. An approximation scheme for gapless schedules

In this section, we first restrict ourselves to gapless schedules in a given instance $(G, r', h, w, m)$ with $K$ distinct release times. Let $\pi^*_g$ be an optimal gapless schedule in $(G, r', h, w, m)$, i.e., a gapless schedule minimizing the maximum completion time over all gapless schedules. Since each edge in $G$ is traversed at least once by some vehicle in a gapless schedule, the following lower bound on the minimum of the maximum completion time $C_{\max}(\pi^*_g)$ is immediately obtained:

$$LB = \frac{Z}{m} \left(\frac{W + H}{m}\right).$$

(5)
Consider the VSP-PATH with $K$ distinct release times, where handling times and edge weights may not be integers. Given any $\varepsilon > 0$, we define

$$\delta = (\varepsilon Z)/8Kmn.$$  

Note that $LB \leq C_{\text{max}}(\pi_g^*)$ implies $4\delta Kn \leq (\varepsilon/2)C_{\text{max}}(\pi_g^*)$ for an optimal gapless schedule $\pi_g^*$ to a given instance $(G, r', h, w, m)$. We now modify the given instance $(G, r', h, w, m)$ into $(G, r', h', w', m)$ by replacing the given handling times $h_j$ and edge weights $w(v_j, v_{j+1})$ by scaled handling times and scaled edge weights

$$h'_j = \lfloor h_j/\delta \rfloor \quad \text{and} \quad w'(v_j, v_{j+1}) = \lfloor w(v_j, v_{j+1})/\delta \rfloor,$$  

where $\lfloor x \rfloor$ is the largest integer no greater than $x$.

**Lemma 2.** For the VSP-PATH with a fixed number of distinct release times of finding an optimal gapless schedule, any exact algorithm to the scaled problem instance $(G, r', h', w', m)$ according to (7) yields a $(1 + \varepsilon/2)$-approximation solution to the problem instance $(G, r', h, w, m)$ with original handling times and original edge weights.

**Proof.** Let $C_{\text{max}}'(\pi_g^*)$ denote the maximum completion time of the schedule $\pi_g^*$ with respect to the scaled instance $(G, r', h', w', m)$.

Suppose that we have found an optimal schedule $\pi'$ and its maximum completion time $C_{\text{max}}'(\pi')$ for the scaled instance $(G, r', h', w', m)$. Let $C_{\text{max}}(\pi')$ denote the maximum completion time of this schedule with respect to the instance $(G, r', h, w, m)$ with original handling times and original edge weights. If $h_j = \delta h'_j$ for all $j = 1, 2, \ldots, n$ and $w(v_j, v_{j+1}) = \delta w'(v_j, v_{j+1})$ for all $j = 1, 2, \ldots, n - 1$, it holds that $C_{\text{max}}(\pi') = \delta C_{\text{max}}'(\pi')$. Moreover, notice that, for any schedule, if $h_j$ and $w(v_j, v_{j+1})$ are increased by some additive value $\beta$, then the maximum completion time of the schedule increases at most $4\beta Kn$ (since we can assume that each vertex or each edge is visited at most $2K - 1$ times by a vehicle in a schedule consisting of $K$ 1-way subschedules for each vehicle). Therefore, by making use of the inequalities

$$\delta h'_j \leq h_j < \delta(h'_j + 1),$$  

$$\delta w'(v_j, v_{j+1}) \leq w(v_j, v_{j+1}) < \delta(w'(v_j, v_{j+1}) + 1),$$

we obtain

$$C_{\text{max}}(\pi') < \delta C_{\text{max}}'(\pi') + 4\delta Kn \leq \delta C_{\text{max}}'(\pi_g^*) + 4\delta Kn,$$

$$\leq C_{\text{max}}(\pi_g^*) + 4\delta Kn \leq (1 + \varepsilon/2)C_{\text{max}}(\pi_g^*),$$

proving the lemma.

Next, we explain how to round given release times $r_j$ into the $K$ distinct release times $r'_j, j = 1, 2, \ldots, n$. Given any $\varepsilon > 0$, let $r^* = \max\{r_j \mid j \in J\}$ and $\Delta = \varepsilon r^*/2$. We round each release time $r_j$ down to the nearest multiple of $\Delta$, i.e.,

$$r'_j = \Delta \lfloor r_j/\Delta \rfloor, \quad j = 1, 2, \ldots, n.$$  

(10)
The number $K$ of distinct release times $r_j$ is bounded as follows:

$$K \leq 1 + r^*/A \leq 1 + 2/\varepsilon.$$  \hfill (11)

We can enjoy the next lemma due to Hall and Shmoys \cite{11} (which we can also find in \cite{10}).

**Lemma 3** (Hall and Shmoys \cite{11}). *For the VSP-PATH of finding an optimal gapless schedule, given an instance $(G, r, h, w, m)$, let $(G', r', h, w, m)$ be the instance with a fixed number of distinct release times obtained by rounding the given release times according to (10). Then, any $(1 + \varepsilon/2)$-approximation solution to $(G, r', h, w, m)$ is a $(1 + \varepsilon)$-approximation solution to $(G, r, h, w, m)$.***

We now define a family of algorithms $\{A'_\varepsilon\}$ as follows. Given an instance $(G, r, h, w, m)$ of the VSP-PATH of finding an optimal gapless schedule and any $\varepsilon > 0$, we first make the instance $(G', r', h', w', m)$ with rounded release times $r_j'$ (see (10)), scaled handling times $h_j'$ and scaled edge weights $w'(v_j, v_{j+1})$ (see (7)). Then, $A'_\varepsilon$ applies the algorithm $A'$ in Section 3.2 to obtain an exact solution to the modified instance $(G, r', h', w', m)$. We regard the resulting schedule as an approximation solution.

From Lemma 2, for any $\varepsilon > 0$, algorithm $A'_\varepsilon$ yields a $(1 + \varepsilon/2)$-approximation solution to the instance $(G, r', h, w, m)$ with original handling times and original edge weights. Thus, from Lemma 3, $A'_\varepsilon$ is a $(1 + \varepsilon)$-approximation algorithm to the given instance $(G, r, h, w, m)$.

Next we examine the time complexity of $A'_\varepsilon$. We define $Z' = \sum_{j=1}^{n-1} w'(v_j, v_{j+1}) + \sum_{j=1}^{n-1} h_j'$. Obviously, $Z' \leq Z/\delta = 8Kmn/\varepsilon$. Since $K \leq 1 + 2/\varepsilon$ (see (11)), the running time of algorithm $A'_\varepsilon$ is $O((1 + 2/\varepsilon)m(n + 2^{1+2/\varepsilon}))(8mn^3(1 + 2/\varepsilon)/\varepsilon)^{m(1+2/\varepsilon)})$ by (4). Therefore, we have the following theorem.

**Theorem 1.** *The family of algorithms $\{A'_\varepsilon\}$ is a polynomial time approximation scheme to the VSP-PATH of finding an optimal gapless schedule, whose time complexity is $O((1 + 2/\varepsilon)m(n + 2^{1+2/\varepsilon}))(8mn^3(1 + 2/\varepsilon)/\varepsilon)^{m(1+2/\varepsilon)})$.***

5. An approximation scheme for general schedules

Unfortunately, the optimal schedule $\pi^*$ for a problem instance $(G, r, h, w, m)$ is not always a gapless schedule, and hence $LB = Z/m = (W + H)/m$ (see (5)) cannot be used as a lower bound on the minimum of the maximum completion time $C^*_{\text{max}}$ attained by general schedules. However, any schedule consists of several gapless schedules for subinstances of $G$. Thus, to find a $(1 + \varepsilon)$-approximation solution, we take into account all configurations of gaps on $G$ which are possible to be incurred by an optimal schedule. This can be executed efficiently by a dynamic programming as follows. For gaps $e'_1, e'_2, \ldots, e'_f \in E$, each of maximal subpaths $G_1, G_2, \ldots, G_{i+1}$ of $G$ induced by non-gap edges will be processed by a gapless schedule. We consider a configuration of gaps on $G$ that minimizes the maximum completion times obtained by algorithm $A'_\varepsilon$ on subpaths $G_1, G_2, \ldots, G_{i+1}$. For a subinstance $(G(i, j), r, h, w, v)$, let
$C_c(G(i, j), r, h, w, v)$ denote the maximum completion time of a schedule computed by algorithm $A'_c$, which is at most $(1 + \varepsilon)$ times the optimum for the instance. For given jobs $i, j \in J$ ($i \leq j$), a number $v$ of vehicles and an upper bound $\lambda$ ($< v$) on the number of gaps, we denote by $Q(i, j, v, \lambda)$ the minimum of the maximum $C_c(G_t, r, h, w, v_t)$ of instances $(G_t, r, h, w, v_t)$, $t = 1, 2, \ldots, \lambda' + 1$ over all possible gaps $e'_1, e'_2, \ldots, e'_\lambda' \in E$, where $\lambda' \leq \lambda$ and $\sum_{1 \leq t \leq \lambda' + 1} v_t = v$. Note that $Q(1, n, m, m - 1)$ is the minimum of the maximum of the $(1 + \varepsilon)$-approximate maximum completion time of a gapless schedule for a subinstance over all possible configurations of gaps.

The following dynamic programming algorithm computes $Q(1, n, m, m - 1)$:

**Algorithm $A_c$.**

Input: A path $G = (V, E)$, where $V = \{v_1, v_2, \ldots, v_n\}$ is its set of $n$ vertices and $E = \{\{v_j, v_{j+1}\} | j = 1, 2, \ldots, n - 1\}$ is its set of edges, release times $r_j$ for $j \in J$, handling times $h_j$ for $j \in J$, edge weights $w(v_j, v_{j+1})$ for $\{v_j, v_{j+1}\} \in E$, the number of vehicles $m$, and a real number $\varepsilon > 0$.

Output: A schedule $\pi_e$ with $C_{\max}(\pi_e) \leq (1 + \varepsilon) \cdot C_{\max}$.

**Step 1:** for $v = 1, 2, \ldots, m$ do

for $i = 1, 2, \ldots, n - v + 1$ do

for $j = i + v - 1, i + v, \ldots, n$ do

$Q(i, j, v, 0) := C_c(G(i, j), r, h, w, v)$

by calling algorithm $A'_c$ for $(G(i, j), r, h, w, v)$

end: /* for */

end: /* for */

end: /* for */

**Step 2:** for $v = 2, 3, \ldots, m$ do

for $\lambda = 1, 2, \ldots, v - 1$ do

for $j = v, v + 1, \ldots, n$ do

$Q(1, j, v, \lambda) := \min \left[ Q(1, j, v, 0), \min_{1 \leq v' \leq v - 1} \min_{1 \leq j' \leq j - 1} \max \{Q(1, j', v - v', \lambda - 1), Q(j' + 1, j, v', 0)\} \right]$

end: /* for */

end: /* for */

end: /* for */

**Step 3:** Compute the configuration of gaps that achieves the $Q(1, n, m, m - 1)$; For each subinstance $G(i, j)$ incurred by the configuration, we compute a schedule $\pi_{(i, j)}$ in Theorem 1;

Let $\pi_e$ be the schedule consisting of these schedules $\pi_{(i, j)}$.

In Step 1, Algorithm $A'_c$ is called $O(mn^2)$ times, and hence it requires $O((1 + 2/\varepsilon)m^2n^2(n + 21^{2/\varepsilon})(8mn^3(1 + 2/\varepsilon)/\varepsilon)^{m(1+2/\varepsilon)})$ time from Theorem 1. The value of $Q(1, j, v, \lambda)$ for each $j, v$ and $\lambda$ at Step 2 can be computed in $O(mn)$ time, and thus this step requires $O(mn^2)$ time.

In Step 3, we can trace the configuration of gaps that achieves the $Q(1, n, m, m - 1)$ in the same time complexity (by storing the indices that attain the minimum in the formula in Step 2). For all subpaths $G(i, j)$ incurred by the configuration, a schedule $\pi_{(i, j)}$
Theorem 1 can be computed in \( O((1 + 2/\varepsilon)m(n + 2^{1+2/\varepsilon})(8mn^3(1 + 2/\varepsilon)/\varepsilon)^{(m(1+2/\varepsilon))}) \) time. By Theorem 1, \( \pi_{(i,j)} \) is a \((1 + \varepsilon)\)-approximation to the subinstance, satisfying \( C_{\text{max}}(\pi_{(i,j)}) \leq O(1, n, m, m - 1) \). Therefore, the schedule which consists of these schedules \( \pi_{(i,j)} \) is a \((1 + \varepsilon)\)-approximation to the original problem \((G(1, n), r, h, w, m)\). This established the next result.

**Theorem 2.** The family of algorithms \( \{A_\varepsilon\} \) is a polynomial time approximation scheme to the VSP-PATH of finding an optimal schedule, whose time complexity is \( O((1 + 2/\varepsilon)m^2n^2(n + 2^{1+2/\varepsilon})(8mn^3(1 + 2/\varepsilon)/\varepsilon)^{(m(1+2/\varepsilon))}) \).

Before closing this section, we remark that our approach to the VSP-PATH can be applied to the VSP-TREE in a tree \( G \). Given release times and handling times on jobs (each of which is located at a vertex in \( G \)) and symmetric weights on edges, the problem asks to find an optimal schedule of \( m \) vehicles to process all the \( n \) jobs. Let \( \ell \) be the number of leaves in \( G \). For a set \( J' \) of jobs with the same release time, we see that an optimal schedule to process \( J' \) is given by visiting all the jobs in \( J' \) along the minimal subtree \( T_{J'} \) containing \( J' \) (hence the completion time is bounded by \( 2\ell \)). Thus such a schedule can be reconstructed in \( O(n) \) time from the first and last jobs to be processed in \( J' \) and the set of leaves of \( T_{J'} \).

To obtain our dynamic programming in Section 3 in the tree case, we need to have a table \( X \) of \( (\ell + 1)Km \)-tuple which consists of the total of traveling and handling times and the set of leaves in a subtree, where \( K (\leq 1 + 2/\varepsilon) \) is the number of distinct rounded release times. An update of each table takes \( O(Kmn(2\ell)^{Km}) \) time. Thus the dynamic programming computes an optimal solution to the problem with integer handling times and edge weights in \( O(Km(n^2 + 2^{K})^Km) \) time. From a similar discussion in Section 4, by scaling handling times and edge weights by \( \delta = (2\ell\varepsilon)/8mn^2 \) (note that each vertex or each edge is visited at most \( n \) times by a vehicle in a schedule), we obtain a \((1 + \varepsilon)\)-approximation algorithm with time complexity \( O((1 + 2/\varepsilon)m^2n^2(n + 2^{1+2/\varepsilon})(8mn^3(1 + 2/\varepsilon)/\varepsilon)^{(m(1+2/\varepsilon))}) \) to the problem of finding an optimal gapless schedule. To find the best configuration of gaps in a tree \( G \), we need to check at most \( \sum_{v=1}^{m} \binom{n-1}{v-1} \binom{m-1}{v-1} = O((nm)^m) \) cases. Therefore, we obtain a \((1 + \varepsilon)\)-approximation algorithm with time complexity \( O((1 + 2/\varepsilon)n^m(m+1)(n^2 + 2^{1+2/\varepsilon})(8mn^3\delta/\varepsilon)^{(m(1+2/\varepsilon))}) \) to the VSP-TREE, which is polynomial if the numbers of vehicles and leaves in \( G \) are fixed.

**6. Concluding remarks**

In this paper, we discussed a scheduling problem of vehicles on a path \( G \) with release and handling times, VSP-PATH. The problem asks to find an optimal schedule of \( m \) vehicles processing \( n \) jobs that minimizes the maximum completion time of all the jobs. The VSP-PATH is NP-hard for any fixed \( m \geq 2 \). In this paper, we showed that the problem with a fixed \( m \) admits a polynomial time approximation scheme, i.e., a family of algorithms \( \{A_\varepsilon\} \). For any \( \varepsilon > 0 \), algorithm \( A_\varepsilon \) delivers a schedule with the maximum completion time at most \((1 + \varepsilon)\) times the optimal in \( O((1 + 2/\varepsilon)m^2n^2(n + 2^{1+2/\varepsilon})(8mn^3(1 + 2/\varepsilon)/\varepsilon)^{(m(1+2/\varepsilon))}) \).
2^{1+2/\varepsilon}(8mn^2(1 + 2/\varepsilon)/\varepsilon)^{(1+2/\varepsilon)}$ time. Our approximation scheme \( \{A,\} \) is based on the approximation of the problem by rounding given release times, and on the fact that any schedule consists of some gapless schedules on subpaths of a given path. We also observed that our algorithm can be extended to the case where \( G \) is a tree, showing the VSP-TREE allows a polynomial time approximation scheme as long as the numbers of vehicles and leaves in \( G \) are constant. It is theoretically significant that the result obtained in this paper specifies an approximation class to which the problem belongs. It is left open whether the VSP-PATH with asymmetric edge weights admits a polynomial time approximation scheme.

**Acknowledgements**

We would like to express our gratitude to the anonymous referees whose constructive suggestions contributed much to improving the written style. This research was partially supported by a Scientific Grant in Aid from the Ministry of Education, Culture, Sports, Science and Technology of Japan.

**References**


