Approximation Algorithms for the Multi-Vehicle Scheduling Problem

Binay Bhattacharya* and Yuzhuang Hu

School of Computing Science, Simon Fraser University, Burnaby, Canada, V5A 1S6
{binay,yhu1}@cs.sfu.ca

Abstract. In this paper we investigate approximation algorithms for the multi-vehicle scheduling problem (MVSP). In MVSP we are given a graph $G = (V, E)$, where each vertex u of V is associated with a job $j(u)$, and each edge e has a non-negative weight $w(e)$. There are m identical vehicles available to service the jobs. Each job $j(u)$ has its own release time $r(u)$ and handling time $h(u)$. A job $j(u)$ can only be serviced by one vehicle after its release time $r(u)$, and the handling time $h(u)$ represents the time needed to finish processing $j(u)$. The objective is to find a schedule in which the maximum completion time of the jobs, i.e. the makespan, is minimized. In this paper we present a 3-approximation algorithm for MVSP on trees, and a $(5 - \frac{2}{m})$-approximation algorithm for MVSP on general graphs.

1 Introduction

In the vehicle routing problem (VRP) [5], we are given m vehicles and a complete graph G where each edge is associated with non-negative edge cost satisfying the triangle inequality. The objective is to find a separate tour for each vehicle such that the total cost of the tour is minimized. In the typical setting of VRP, a depot node o is also given, and in each tour the vehicle should start from and end at the depot.

VRP is NP-hard as it is a generalization of the famous traveling salesman problem (TSP) [13]. VRP in fact represents one of the most challenging tasks in the area of optimization, and has been extensively investigated by many researchers. Various tools, such as integer programming, local search, and clustering, have been used to solve this class of problems [17]. As a well-recognized approach of dealing with NP-hard problems, good approximation algorithms for VRP are of great theoretical and practical importance. In this paper, we study approximation algorithms for a variant of VRP. More specifically, we investigate approximation algorithms for the multi-vehicle scheduling problem (MVSP) [9].

1.1 Multi-Vehicle Scheduling Problem (MVSP)

In MVSP we are given a graph $G = (V, E)$, where each vertex u of V is associated with a job $j(u)$, and each edge e has a non-negative weight $w(e)$. There are m

* Research was partially supported by MITACS and NSERC.

O. Cheong, K.-Y. Chwa, and K. Park (Eds.): ISAAC 2010, Part II, LNCS 6507, pp. 192–205, 2010.

identical vehicles available to service the jobs. Each job $j(u)$ has its own release time $r(u)$ and handling time $h(u)$. A job $j(u)$ can only be serviced by one vehicle after its release time $r(u)$, and the handling time $h(u)$ represents the time needed to finish processing $j(u)$. This implies that when the vehicle arrives earlier, the vehicle may have to wait until the time $r(u)$ to service $j(u)$. The vehicle may also choose to move to other vertices and come back to u later to service $j(u)$. The objective of MVSP is to find a schedule in which the maximum completion time of the jobs, i.e., the makespan, is minimized.

MVSP is a variation of the classical VRP with time windows (VRPTW). In VRPTW each vertex u in the graph is associated with a full time window $[r(u), d(u)]$, where $r(u)$ and $d(u)$ denote the release time and the deadline for the job $j(u)$ respectively. Time windows are called *soft*, if they are considered non-biding, i.e., the vehicle is allowed to arrive earlier or later at a customer node. However, when the time window constraint is violated, a separate penalty cost of earliness or lateness will incur. Time windows are called *hard*, if the customer nodes must be serviced within the specified time intervals. A survey for VRPTW can be found in [17]. For approximation algorithms, Yehuda et al. gave an $O(\log n)$-approximation algorithm for VRPTW on a line in a geometric space [3]; Blum et al. presented an $O(\log^2 n)$-approximation algorithm for VRPTW on general graphs [2].

MVSP can be viewed as a relaxation of VRPTW, as in MVSP only release times are associated with the customer nodes. The current research status on MVSP is as follows. A substantial amount of research work [4] [7] [14] has been devoted to a special case of MVSP, called SVSP, where only one vehicle is available to service the customers. Nagamochi et al. [9] introduced MVSP in the general sense and gave a 2-approximation algorithm for MVSP on paths. For MVSP on trees, Nagamochi et al. [15] [16] gave a $(5-2/(m+1))$-approximation algorithm. There are also two polynomial time approximation schemes (PTAS) [1] [8] for MVSP on trees under certain restrictions. For MVSP on general graphs, a $(9+\epsilon)$-approximation can be derived from Even et al. [6].

1.2 Our Results and Solution Techniques

We summarize our results as follows.

1. For MVSP on trees, we design a 3-approximation algorithm.
2. For MVSP on general graphs, we present a $(5-\frac{2}{m})$-approximation algorithm. This algorithm is based on the 3-approximation algorithm we propose for MVSP on trees.

Our results are obtained by introducing an appropriate relaxation problem for MVSP. We apply dynamic programming to give a 3-approximation for VRP on trees. The main idea of this algorithm, is to use dynamic programming to indirectly decompose the original problem P into a set of disjoint subproblems. Approximating these subproblems separately gives us the solution for P with the desired approximation ratio.

Dynamic programming is one of the most fundamental and powerful tools for designing efficient algorithms. However, its application in approximation algorithms for VRP, is relatively new. The existing way of using dynamic programming for VRP, e.g., in [2] [9] and [10], works as follows. First a set of disjoint NP-hard subproblems is defined for the original problem P, and an approximation algorithm A with ratio α is designed for these subproblems. Typically these subproblems have good properties, and the algorithm A can approximate them well. In this method, dynamic programming is used as a master algorithm to locate a set of subproblems with cost bounded by $\beta \cdot OPT(P)$, where $OPT(P)$ denotes the optimal solution cost of P. During this process all the configurations of the subproblems are tried, and the algorithm A is applied to each of the configurations. The one with the smallest cost is chosen to be the final solution. It is easy to see that this solution is an $(\alpha\beta)$-approximation for P.

This approach relies on the fact that all the configurations of the subproblems of P can be examined in polynomial time by dynamic programming. Therefore it is only applicable when the underlying graph is a path [9], or when some ordering can be found for the underlying graph (as in the $O(\log^2 n)$-approximation algorithm for VRPTW [2], where the dynamic programming proceeds based on an ordering of the vertices). However, for VRP on trees, we do not know how to deploy this method, since for a vertex u, the number of possible configurations of the subproblems containing u is exponential in the number of children of u.

We apply dynamic programming to obtain a 3-approximation algorithm for MVSP on trees in the following way. Our algorithm consists of two steps. In the first step, we use dynamic programming to decompose the original problem P into a set of disjoint subproblems. However, as it is not possible to try all the configurations of the subproblems by directly obtaining solutions for P, we instead find a relaxation problem P' and locate a set S of disjoint subproblems. In the second step we work on approximation algorithms for the subproblems in S. We show that a good approximation for P can be obtained by approximating these subproblems well.

The two steps in our algorithm are highly connected. For MVSP on trees, the problem P' solved in the first step in fact comes from the approximation algorithm used in the second step. We locate P' in the following way. Define an edge e to be a gap if and only if in the optimal solution no vehicle passes through e to service customers. Define a gapless subproblem to be a subproblem whose underlying subgraph contains no gaps. For a gapless subproblem P_1, we design an approximation algorithm which produces a solution with cost function $f(P_1)$. Let $S(P)$ denote the set of gapless subproblems of P. We treat $\max_{P_1 \in S(P)} f(P_1)$ as a bound, and define P' to be the problem of locating a set of subproblems with the smallest possible bound $\max_{P_1 \in S(P)} f(P_1)$. More details of this algorithm can be found in Section 2 of this paper.

The rest of the paper is organized as follows. In Section 2 we present a 3-approximation algorithm for MVSP on trees. In Section 3 we show that a $(5 - \frac{2}{m})$-approximation is possible for MVSP on general graphs. In Section 4 we conclude the paper and give figure research directions.

2 3-Approximation for MVSP on Trees

In this section we show the 3-approximation algorithm for MVSP on trees. The following notation is used in describing our algorithms. Assume we are given a rooted tree T. For a node u in T, we denote the parent of u by $p(u)$, and the subtree rooted at u (including u) by T_u. Here $p(u)$ is null if u is the root. We use $w(e)$ to denote the weight or cost of edge e.

2.1 Defining the Problem P' for MVSP on Trees

To define the relaxation problem P' for MVSP on trees, we first introduce several lower bounds for a gapless subproblem of an MVSP instance on trees. These lower bounds are also used to design approximation algorithms for MVSP on paths [9]. For a gapless subproblem P_1, we define two lower bounds for the makespan:

$$LB_1(P_1) = \max_{u \in V(P_1)}\{r(u) + h(u)\}, \qquad LB_2(P_1, m') = \frac{W(P_1)+H(P_1)}{m'}$$

where $V(P_1)$ and $E(P_1)$ are the respective vertex set and edge set involved in P_1, m' represents the number of vehicles used to service the jobs in P_1, and $W(P_1)$ and $H(P_1)$ are the total edge weights of $E(P_1)$ and the total handling times of the jobs associated with the vertices in $V(P_1)$ respectively.

A collection of disjoint sets $S_1, S_2, \cdots, S_t$ is called a *partition* of the graph $G = (V, E)$, if $S_i \subseteq V$ $(1 \leq i \leq t)$ and the vertices of S_i $(1 \leq i \leq t)$ induce a connected subgraph in G. We define a *vehicle configuration* to be a partition of the tree where each connected component C_i in the partition is associated with a positive integer m_i and $\sum_i m_i = m$. Given a vehicle configuration VC, we call an edge a *cut edge* under VC, if its two ends are in different subsets of vertices in the partition corresponding to VC.

For a vertex u, we denote $C_{VC}(u)$ to be the set of connected components in T_u under VC. Let $S_{VC}(u)$ be the set of subproblems defined on the corresponding connected components in $C_{VC}(u)$. We define a new bound $B(G, m)$ to be $\min_{VC} \max_{P_2 \in S_{VC}(o)}(B(P_2, m') = LB_1(P_2) + (2W(P_2) + H(P_2))/m')$, where o is the root of the tree and m' is the number of vehicles allocated for P_2 under VC. We similarly define $B(T_u, m')$ where $m' \in (0, m]$ is the number of vehicles used to service the vertices in the subtree T_u. Our relaxation problem P' for MVSP is just to find the set of subproblems corresponding to the bound $B(G, m)$. We first solve P', namely compute $B(G, m)$, in strongly polynomial time by dynamic programming. By doing so we find a set of cut edges and the original problem is correspondingly decomposed to a set S of disjoint subproblems. In the second step we figure out a feasible schedule for the subproblems in S with a makespan of at most $B(G, m)$. Note that the cut edges we located might be different from the gaps that are defined on the optimal MVSP solution.

We first justify the second step of our algorithm in Lemma 1. The proof is constructive.

Lemma 1. *For MVSP on trees the subproblems corresponding to $B(G, m)$ induce a feasible MVSP solution with cost at most $B(G, m)$.*

Proof. Let VC be the vehicle configuration corresponding to $B(G, m)$. Consider a subproblem P_2 defined on a connected component under VC. Assume m' vehicles are involved in P_2 under VC. In the optimal solution for P_2, an edge might be traversed by more than one vehicles. We double the tree edges and obtain a hamiltonian path containing all the jobs (by a depth first traversal of the tree). We traverse the path from the root, and associate the handling times with the jobs when they are visited for the first time. Therefore we transformed P_2 for MVSP on trees to another subproblem P_2' of MVSP on paths with $W(P_2') = 2W(P_2)$, $H(P_2') = H(P_2)$ and $LB_1(P_2') = LB_1(P_2)$. Using the algorithm in [9], we obtain a schedule for P_2' with cost bounded by $LB_1(P_2') + LB_2(P_2', m')$. This schedule is also feasible for P_2 and has a cost of at most $LB_1(P_2) + (2W(P_2) + H(P_2))/m' = B(P_2, m')$. The lemma then holds after applying this analysis to every other connected component under VC. □

Lemma 2. *For MVSP on trees the bound $B(G, m)$ corresponds to a feasible MVSP solution with cost bounded by $3 \cdot OPT(P)$.*

Proof. The gapless subproblems will be encountered when computing $B(G, m)$. The same operation in the proof of Lemma 1 can be applied on the gapless subproblems. Since $B(G, m)$ is the smallest possible, we have that $B(G, m) \leq 3 \cdot OPT(P)$. Due to Lemma 1 there exists a feasible solution with cost at most $B(G, m)$. This completes the proof. □

The above proof also shows how to solve the subproblems once they are located, therefore in the following we only focus on how to locate the appropriate subproblems.

2.2 Locating the Subproblems for MVSP on Trees

The core of our solution is an algorithm to solve the following simple feasibility decision problem D: Given a real number λ, is $B(G, m) \leq \lambda$?

Solving the Decision Problem. In the following we assume that only the leaves of the tree are associated with handling times. The general case can be easily transformed to this setting by creating a pseudo vertex u' for each vertex u in the graph and connecting u and u' by a zero cost edge. We set $h(u')$ to $h(u)$. It is easy to see that the optimal MVSP solution remains the same after this transformation.

We solve the feasibility problem also by dynamic programming. Given a vehicle configuration VC, we denote $P_{VC}(u)$ to be the subproblem defined on the connected component containing u in $C_{VC}(u)$, and $P'_{VC}(u)$ to be the set of subproblems defined on the connected components not containing u in $C_{VC}(u)$. Recall that $C_{VC}(u)$ consists of all the connected components of T_u under VC.

To solve the decision problem, we maintain a variable $obj(u)$ and a table $table(u)$ for each vertex u. The variable $obj(u)$ records the minimum number m' of vehicles needed to guarantee that $B(T_u, m') \leq \lambda$. The variable $table(u)$ has two dimensions. Assume the minimum value of an entry $table(u)[i_1][i_2]$ of $table(u)$

is obtained under the vehicle configuration VC. Then this value represents $2 \cdot W(P_{VC}(u)) + H(P_{VC}(u))$, given that i_1 equals $LB_1(P_{VC}(u))$, and i_2 vehicles have been used to service the jobs in the subproblems of $P'_{VC}(u)$. Here $LB_1(T_u)$ represents the first lower bound LB_1 for all the jobs in T_u. In other words, given the constraints of i_1 and i_2, the value of the entry $table(u)[i_1][i_2]$ equals the minimum of $2 \cdot W(P_{VC}(u)) + H(P_{VC}(u))$, for any vehicle configuration VC in T_u. For each vertex u, we initialize all the table entries of $table(u)$ to $+\infty$. Given an input of λ, the algorithm maintains that an entry $table(u)[i_1][i_2]$ is not $+\infty$ if and only if there exists a vehicle configuration VC under which the B bound of each subproblem in $P'_{VC}(u)$ is less than or equal to λ.

We show the pseudo code of generating the tables in Figure 1.

For a leaf u, since there must be a vehicle to service u, and there are no other jobs in T_u, the entry $table(u)[r(u) + h(u)][0]$ is set to $h(u)$. This is for the case where the edge $(p(u), u)$ is not a cut edge. The edge cost of $(p(u), u)$ will be considered later when merging $table(u)$ to $table(p(u))$. When $(p(u), u)$ is a cut edge, again as u needs to be serviced, we set $obj(u)$ to 1.

When u is a non-leaf node, the dynamic programming proceeds as follows. We assume that all the tables and variables of its children have already been computed. Let the children of u be $c_1, c_2, \cdots, c_t$ (in an arbitrary order). The algorithm scans this list of children from left to right, and incorporates the tables of the children into $table(u)$ one at a time. The variable $obj(u)$ is updated after all the children of u are considered and $table(u)$ becomes available. The computation of $obj(u)$ can be expressed as

$$obj(u) = \min_{i_1, i_2}(i_2 + \lceil table(u)[i_1][i_2]/(\lambda - i_1)\rceil).$$

where $1 \leq i_1 \leq LB_1(G)$, $0 \leq i_2 \leq m$.

For an entry $table(u)[i_1][i_2]$, assume its value is obtained under the vehicle configuration VC. Then $table(u)[i_1][i_2]$ represents $2 \cdot W(P_{VC}(u)) + H(P_{VC}(u))$. Recall that i_2 records the number of vehicles used in the subproblems of $P'_{VC}(u)$. Therefore the total number of vehicles used in T_u corresponding to the entry $table(u)[i_1][i_2]$ is $i_2 + \lceil table(u)[i_1][i_2]/(\lambda - i_1)\rceil$. As we assume $e = (p(u), u)$ is a cut edge, and i_1 represents $LB_1(P_{VC}(u))$, we can then get the minimum for $obj(u)$ after trying all possible values of i_1 and i_2.

The updating of $table(u)$ for a vertex u is crucial for solving the decision problem D. Let v be a child of u. Assume $table(v)$ is already available before we start to incorporate $table(v)$ into $table(u)$. Let $table_v(u)$ be the resulting new table for u. Every entry of $table_v(u)$ is initialized to $+\infty$, and $table(u)$ will be set to $table_v(u)$ after processing v. Note that we need to work on this new table, because additional vehicles may be needed to service all the vertices in T_v. Consider the following cases:

Case 1: (u, v) is a cut edge. In this case $table_v(u)$ is updated as

$$table_v(u)[i_1][i_2 + obj(v)] = table(u)[i_1][i_2], \text{ given that } i_2 + obj(v) \leq m.$$

Here $1 \leq i_1 \leq LB_1(G)$, $0 \leq i_2 \leq m$. The entry $table_v(u)[i_1][i_2 + obj(v)]$ will be updated if it is greater than $table(u)[i_1][i_2]$.

Algorithm MVSP-decision(T_u, m, λ)
Input: an MVSP instance P defined on a tree T_u, an integer m and a real number λ.
Output: a table showing a bound $B(T_u, m)$ (at least λ).

```
1  if u is a leaf then
2     obj(u) = 1
3     table(u)[r(u) + h(u)][0] = h(u)
4     return table(u)
5  endif
6
7  for each child v of u do
8     table(v) = MVSP-decision(T_v, m, λ);
9     tmp ← a new table with each entry set to +∞
10    for i_1 = 0 to LB_1(G) do
11       for i_2 = 0 to m do
12          Comment: when e = (u, v) is a cut edge
13          if table(u)[i_1][i_2] < tmp[i_1][i_2 + obj(v)] and obj(v) ≤ m − i_2 then
14             tmp[i_1][i_2 + obj(v)] = table(u)[i_1][i_2]
15          endif
16          Comment: when e = (u, v) is not a cut edge
17          for j_1 = 0 to LB_1(G) do
18             for j_2 = 0 to m − i_2 do
19                t_1 = max{i_1, j_1}
20                t_2 = table(u)[i_1][i_2] + table(v)[j_1][j_2] + 2w((u, v))
21                if t_2 < tmp[t_1][i_2 + j_2]) then
22                   tmp[t_1][i_2 + j_2] = t_2
23                endif
24    table(u) ← tmp
25
26 Comment: consider the case when e = (p(u), u) is a cut edge
27 for i_1 = 0 to LB_1(G) do
28    for i_2 = 0 to m do
29       t_1 = ⌈table(u)[i_1][i_2]/(λ − i_1)⌉
30       if t_1 + i_2 < obj(u) then
31          obj(u) = t_1 + i_2
32       endif
33
34 return table(u)
```

Fig. 1. Solving the decision problem for MVSP on trees

Note that an entry $table(u)[i_1][i_2]$ has a meaningful value (not $+\infty$), if and only if under the corresponding vehicle configuration VC, $B(P_2, m') \leq \lambda$ holds for every subproblem P_2 in $P'_{VC}(u)$. Here m' denotes the number of vehicles allocated for P_2 under VC. This is implemented by line 13 in the algorithm MVSP-decision in Figure 1. Recall that $obj(u)$ records the minimum number m' of vehicles to guarantee that $B(T_u, m') \leq \lambda$. Therefore if $i_2 + obj(u)$ exceeds m, then we know that VC is not a feasible vehicle configuration and there is no need to complete the rest of the computation for VC.

Case 2: (u, v) is not a cut edge. In this case $table_v(u)$ is updated as

$$table_v(u)[t][i_2 + j_2] = table(u)[i_1][i_2] + table(v)[j_1][j_2] + 2 \cdot w((u, v)).$$

where $1 \leq i_1, j_1 \leq LB_1(G)$, $0 \leq i_2 + j_2 \leq m$, and $t = \max\{i_2, j_2\}$. The entry $table_v(u)[t][i_2 + j_2]$ is updated when $table(u)[i_1][i_2] + table(v)[j_1][j_2] + 2w((u, v))$ is smaller.

Under this case we need to merge the two components C_u and C_v containing u and v respectively under the current settings for the two entries of $table(u)$ and $table(v)$. The new component has LB_1 equal to the maximum of that of C_u and C_v, so we update the first dimension of the new entry to be $t = \max\{i_2, j_2\}$. After the merging, the connected components not containing u and v remain unchanged, therefore we set the second dimension to $i_2 + j_2$. Finally we need to add $2 \cdot w((u, v))$ to this entry, as (u, v) now becomes part of the new connected component containing both u and v.

The above algorithm runs in strongly polynomial time. Recall that $LB_1(P_1) = \max_{u \in V(P_1)} \{r(u) + h(u)\}$, therefore for a vertex u the second dimension of $table(u)$ takes at most $|V|$ distinct values. The time complexity of this algorithm is dominated by Case 2 when updating $table(u)$ for each vertex u. The algorithm runs in time $O(m^2|V|^3)$.

It is not difficult to see that this algorithm can be used to solve a disguised problem D' of D: Given a real number λ, compute the smallest possible bound $B(G, m)$ (at least λ) subject to the constraint that, under the corresponding vehicle configuration VC the bound $B(C, m')$ of each connected component C, not including the root o in $C_{VC}(o)$, is at most λ. Here m' is the number of vehicles used for the customers in C. We will show in the next subsection that a strongly polynomial time algorithm for the optimization problem P' can be obtained by solving D'.

Solving the Optimization Problem. In this subsection we present a polynomial time algorithm called MVSP-optimization (Figure 2) to solve our optimization problem. This algorithm is in the same spirit as that of the parametric searching technique. Developed by Megiddo in [11] and [12], the parametric searching technique works as follows. Let λ^* be the optimal solution of an optimization problem P_1. Assume that for P_1 we have a decision problem $D_1(\lambda)$ which is monotone in λ, in the sense that we can decide whether $\lambda < \lambda^*$, $\lambda = \lambda^*$ or $\lambda > \lambda^*$. Assume that we have an algorithm A for the decision problem $D_1(\lambda)$. To solve the optimization problem P_1, Megiddo's idea is to run A generically. It

seems somewhat strange to run an algorithm when the input is still unknown. The core in Megiddo's method is to maintain an open interval I where λ^* lies throughout the execution of the algorithm A. More specifically, I is initialized to $(-\infty, +\infty)$, and at each step of running A with unknown input, a critical value t_1 is computed and the concrete version of A is executed with parameter t_1. Therefore after the first step the interval I is shrunk to either $(-\infty, t_1]$ or $[t_1, +\infty)$. When the generic version of A is terminated, we either find λ^* or an interval with its lower end equals λ^*.

It is crucial to generate the critical values in the parametric search framework. These critical values in fact discretize the problem P_1 and make it possible to compute P_1 optimally in polynomial time. Intuitively the critical values, or the steps of A represent all the tests which the optimal solution must pass. On the other hand, if a solution passes all such tests, then it is a candidate of the optimal solution. Note that as long as all the critical values are tested, the master optimization algorithm need not necessarily to be the same with A, e.g., in many cases sorting all the critical values suffices for solving the optimization problem.

The algorithm MVSP-optimization in Figure 2 shares the same spirt with the parametric searching technique, however, the details of MVSP-optimization are different from the methods in [11] and [12]. Instead of running the algorithm for the decision problem generically, we take advantage of the structure of the tree and perform the tests bottom up along the path from the vertex being visited to the root of the tree. Note that the idea of exploring parallelism for serial algorithms as described in [12] does not seem to apply in this context.

Recall that P' is to compute the bound $B(T, m)$ for a tree T and a given integer m. We first define the discrete events, or the critical values, needed by our method. Given a subtree T_u of T and a number $0 < m' \leq m$, we define the critical value at u to be the bound $B(T_u, m')$ under the constraint that m' vehicles are assigned to service the jobs in T_u. Corresponding to this definition, our algorithm runs in the bottom up fashion: the computation of the bound $B(T, m)$ starts from the leaves of the tree, and the optimal value of LB for a subtree T_u will be available after all the vertices in T_u have been processed.

The reason we choose $B(T_u, m')$ $(0 < m' \leq m)$ to be the critical value at a vertex u is as follows. Let $F^*(T)$ be the optimal forest corresponding to the bound $B(T, m)$. Then $B(T, m)$ is determined by a particular connected component C of $F^*(T)$. Assume u is the root of C and T_u is allocated m' vehicles in total in the optimal solution. Then we can get the optimal solution by applying the algorithm MVSP-decision in Figure 1 with the parameter $B(T_u, m')$. Given the values $B(T_u, m')$ for every subtree T_u of T and every integer $0 < m' \leq m$, it is easy to see that we can compute $B(T, m)$ by $m \cdot |V|$ applications of the algorithm MVSP-decision.

For a leaf u of T, it is trivial to compute $B(T_u, m')$ $(0 < m' \leq m)$. In our algorithm, whenever a critical value $B(T_u, m')$ $(u \in T, 0 < m' \leq m)$ is known, we propagate the test on this value along the path from u to the root o of T. In other words, for every vertex u' on the path from u to o, we apply algorithm MVSP-decision on $T_{u'}$ with $m'' \in [1, m]$ vehicles and the feasibility parameter $B(T_u, m')$.

For an arbitrary vertex u, a critical value $B(T_u, m')$ $(u \in T, 0 < m' \leq m)$ is computed if and only if for every vertex u' (other than u) in T_u, the test of $B(T_{u'}, m')$ $(0 < m' \leq m)$ has already been taken on T_u. It is easy to see that the optimal solution can be computed after $m^2|V|^2$ calls to the algorithm MVSP-decision. Therefore the time complexity of the algorithm is $O(m^4|V|^5)$. The pseudo code of the above algorithm is listed in Figure 2.

Algorithm MVSP-optimization(T, m, o)
Input: an MVSP instance P defined on a tree T, an integer m and a root o.
Output: the bound $B(T, m)$.

```
1  if u is a leaf then
2    B(T_u, 1) = r(u) + 2h(u)
3  endif
4
5  for each child v of u do
6    call MVSP-optimization(T_v, m, o)
7
8  for every vertex u' on the path from u to the root o do
9    for every integer m' = 0 to m do
10     for every integer m'' = 0 to m do
11       call MVSP-decision(T_u', m'', B(T_u, m'))
```

Fig. 2. Locating the subproblems for MVSP on trees

Lemma 3. *The algorithm MVSP-optimization computes the bound $B(G, m)$ exactly. The running time of the algorithm is $O(m^4|V|^5)$.* □

Proof. In the algorithm MVSP-optimization, by computing $B(T_u, m')$ $(u \in V, 1 \leq m' \leq m)$, we are in fact assuming that the edge $(p(u), u)$ is a cut edge. The lemma is trivially true when $|V| = 1$. Assume the lemma holds when $|V| < n$ $(n > 1)$. Assume the graph $G = (V, E)$ has n vertices and VC is the vehicle configuration corresponding to $B(G, m)$. Assume $e_1 = (p(u_1), u_1), e_2 = (p(u_2), u_2), \cdots, e_t = (p(u_t), u_t)$ are all the cut edges where $p(u_1), p(u_2), \cdots, p(u_t)$ are in the connected component on which $P_{VC}(o)$ is defined. Assume $m_1, m_2, \cdots, m_t$ vehicles are allocated for the vertices in $T_{u_1}, T_{u_2}, \cdots, T_{u_t}$ respectively under VC. According to our assumption, $B(T_{u_1}, m_1), B(T_{u_2}, m_2), \cdots, B(T_{u_t}, m_t)$ can be computed exactly in polynomial time. In MVSP-optimization the critical value tests with feasibility parameters $B(T_{u_1}, m_1)$, $B(T_{u_2}, m_2)$, $\cdots$, $B(T_{u_t}, m_t)$ will be performed against the root o. The lemma then holds for G due to the definition of the disguised decision problem D' and the fact that $B(G, m)$ is determined by either $P_{VC}(o)$ or a subproblem in $P'_{VC}(o)$.

By Lemmas 1, 2 and 3 we establish

Theorem 1. *There is a 3-approximation for MVSP on trees.* □

3 $(5 - \frac{2}{m})$-Approximation for MVSP on General Graphs

The algorithm MVSP-optimization can be further utilized to get a $(5 - \frac{2}{m})$-approximation for MVSP on general graphs. We simply apply MVSP-optimization on a minimum spanning tree of the underlying graph G. We then obtain a feasible MVSP solution by the strategy mentioned in the proof of Lemma 1. In the following we prove that the cost of this solution is at most $(5 - \frac{2}{m}) \cdot OPT(P)$.

It is well known that minimum spanning trees have the property stated in Lemma 4.

Lemma 4. *Let C be the cycle introduced by adding an edge e to a minimum spanning tree T. Then the cost of e is no less than that of any edge on C.*

Lemma 4 together with Lemma 5 are crucial for our analysis. In the following we define $F^*(P)$ to be the subgraph on which the gapless subproblems are defined for an MVSP problem P.

Lemma 5. *For an MVSP problem P the cost of any edge in $F^*(P)$ is a lower bound on $OPT(P)$.*

Assume we already know $F^*(P)$ and the optimal cost $OPT(P)$. We describe in the following an algorithm MVSP1 based on $F^*(P)$, which is designed for analysis only. In MVSP1, we first remove any edge with cost more than $OPT(P)$ from the original graph G. Let the resulting graph be G'. We then compute minimum spanning trees for each of the connected components of G'. Finally we apply the strategy mentioned in the proof of Lemma 1 to the computed minimum spanning trees. As shown in Lemma 6, the cost of such a solution is at most $(5 - \frac{2}{m}) \cdot OPT(P)$.

Lemma 6. *Given the subgraph $F^*(P)$ for an MVSP problem P, the cost of the solution produced by MVSP1, is at most $(5 - \frac{2}{m}) \cdot OPT(P)$.*

Proof. Let G' be the resulting graph after removing edges with costs more than $OPT(P)$ from the original graph G. Due to Lemma 5, all the edges of $F^*(P)$ still remain in the connected components of G'. Consider an arbitrary connected component CC of G'. Let $CC_1, CC_2, \cdots, CC_t$ be the connected components of $F^*(P)$ in CC. We may assume that each CC_i $(1 \leq i \leq t)$ is a tree, as otherwise some edges can be removed from a connected component of $F^*(P)$ to make it a tree. We construct a new graph $G'' = (V'', E'')$ where each vertex $v_i \in V''$ $(1 \leq i \leq t)$ corresponds to CC_i $(1 \leq i \leq t)$, and the cost of an edge $(v_i, v_j) \in E''$ $(1 \leq i, j \leq t)$ equals the minimum edge cost of any edge between a vertex in CC_i and a vertex in CC_j in G'.

Let $e_1, e_2, \cdots, e_{t-1}$ be the minimum spanning tree edges in G''. Adding these edges to $F^*(P)$ in G' will produce a connected tree T' spanning all the vertices in CC. Let P_0 and P_i $(1 \leq i \leq t)$ be the associated subproblems defined on T' and CC_i respectively. Let m' and m_i $(1 \leq i \leq t)$ be the vehicles used in T' and CC_i respectively. We have the following inequalities.

$$(1)\ t \le m' \qquad (2)\ W(P_i) + H(P_i) \le m_i \cdot OPT(P) \qquad (3)\ w(e_i) \le OPT(P)$$

where $1 \le i \le t$. The first inequality holds, because at least one vehicle should be allocated for each C_i $(1 \le i \le t)$. The second inequality is due to $(W(P_i) + H(P_i))/m_i \le OPT(P)$ $(1 \le i \le t)$. The third inequality is implied by the algorithm MVSP1.

Therefore we have

$$\begin{aligned} & B(T', m') = LB_1(P_0) + (2 \cdot W(P_0) + H(P_0))/m' \\ \le\ & OPT(P) + 2 \cdot \sum_{i=1}^{t} (W(P_i) + H(P_i))/m' + 2 \cdot \sum_{i=1}^{t-1} w(e_i)/m' \\ \le\ & OPT(P) + 2 \cdot \sum_{i=1}^{t} m_i \cdot OPT(P)/m' + 2 \cdot (t-1) \cdot OPT(P)/m' \\ \le\ & (5 - \frac{2}{m}) \cdot OPT(P), \end{aligned}$$

The lemma then holds after applying the same analysis to other connected components of G'. □

Consider another algorithm MVSP2, where we first compute the minimum spanning tree F of G and then remove the edges with costs more than $OPT(P)$ from F. In the following we show that MVSP1 and MVSP2 are essentially equivalent.

Lemma 7. *Given $F^*(P)$ for an MVSP problem P, MVSP1 and MVSP2 produce the same solution or two solutions with the same cost.*

Proof. Assume CC is a connected component of G'. We claim that the vertices in CC appear *contiguously* in any minimum spanning tree, say F, of G. This *contiguity* is in the sense that any vertex not in CC is adjacent to at most one vertex in CC by the edges of F.

Assume the vertices in CC are in two disjoint subtrees T_1 and T_2 of F. As CC is a connected component, there is one edge e in CC that connects T_1 and T_2 and has cost at most $OPT(P)$. Adding e to F will create a cycle. As T_1 and T_2 are disjoint in F, on this cycle there must exist two edges e_1 and e_2 of F, each of which connects a vertex of CC to a vertex not in CC. According to MVSP1, both e_1 and e_2 have costs larger than $OPT(P)$. A contradiction to Lemma 4. □

Given a minimum spanning tree F of G, our algorithm MVSP-optimization gives the best partition of the tree to minimize $B(F, m)$. We proved in Lemma 6 and Lemma 7 that there exists a partition of F such that the corresponding LB bound is at most $(5 - \frac{2}{m}) \cdot OPT(P)$. Further due to Lemma 1, we establish

Theorem 2. *There is a $(5 - \frac{2}{m})$-approximation for MVSP on general graphs.* □

4 Conclusions and Future Work

In this paper we obtained a 3-approximation for MVSP on trees, and a $(5 - \frac{2}{m})$-approximation for MVSP on general graphs. We showed that by finding appropriate relaxation problems, dynamic programming can be applied in the design of approximation algorithms for VRP on trees. Our $(5 - \frac{2}{m})$-approximation for MVSP on general graphs is just based on the 3-approximation algorithm for MVSP on trees. In the future we would like to further improve the approximation ratio for MVSP on general graphs, possibly by investigating new ways of extending the use of dynamic programming for VRP to general graphs.

References

1. Augustine, J.E., Seiden, S.: Linear time approximation schemes for vehicle scheduling problems. Theoretical Computer Science 324(2-3), 147–160 (2004)
2. Bansal, N., Blum, A., Chawla, S., Meyerson, A.: Approximation algorithms for deadline-TSP and vehicle routing with time-windows. In: Proceedings of the 36th Annual ACM Symposium on Theory of Computing, pp. 166–174 (2004)
3. Bar-Yehuda, R., Even, G., Shahar, S.: On approximating a geometric prize-collecting traveling salesman problem with time windows. In: Proceedings of the 11th Annual European Symposium on Algorithms, pp. 55–66 (2003)
4. Bhattacharya, B.K., Carmi, P., Hu, Y., Shi, Q.: Single Vehicle Scheduling Problems on Path/Tree/Cycle Networks with Release and Handling Times. In: Proceedings of the 19th International Symposium on Algorithms and Computation, pp. 800–811 (2008)
5. Dantzig, G.B., Ramser, R.H.: The truck dispatching problem. Management Science 6, 80–91 (1959)
6. Even, G., Garg, N., Konemann, J., Ravi, R., Sinha, A.: Min-max tree covers of graphs. Operations Research Letters 32(4), 309–315 (2004)
7. Karuno, Y., Nagamochi, H.: Better approximation ratios for the single-vehicle scheduling problems on line-shaped networks. Networks 39(4), 203–209 (2002)
8. Karuno, Y., 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, pp. 36–47 (2001)
9. Karuno, Y., 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, pp. 218–229 (2001)
10. Korula, N., Chekuri, C.: Approximation algorithms for orienteering with timewindows (September 2007) (manuscript)
11. Megiddo, N.: Combinatorial optimization with rational objective functions. Math. Oper. Res. 4, 414–424 (1979)
12. Megiddo, N.: Applying parallel computation algorithms in the design of serial algorithms. Journal of ACM 30(4), 852–865 (1983)
13. Menger, K.: Reminiscences of the vienna circle and the mathematical colloquium. Dortmund (1994)
14. Nagamochi, H., Mochizuki, K., Ibaraki, T.: Complexity of the Single Vehicle Scheduling Problem on Graphs. Inform. Systems Oper. Res. 35(4), 256–276 (1997)

15. Nagamochi, H., Okada, K.: A faster 2-approximationalgorithm for the minmax p-traveling salesmenproblem on a tree. Discrete Applied Mathematics 140(1-3), 103–114 (2004)
16. Nagamochi, H., Okada, K.: Polynomial time 2-approximation algorithms for the minmax subtree cover problem. In: Ibaraki, T., Katoh, N., Ono, H. (eds.) ISAAC 2003. LNCS, vol. 2906, pp. 138–147. Springer, Heidelberg (2003)
17. Toth, P., Vigo, D. (eds.): The vehicle routing problem. SIAM Monographs on Discrete Mathematics and Applications, vol. 9. SIAM, Phaladelphia (2002)