k-delivery traveling salesman problem on tree networks*

Binay Bhattacharya[1] and Yuzhuang Hu[2]

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

2 **School of Computing Science, Simon Fraser University, Burnaby, Canada, V5A 1S6**
huyuzhuang@gmail.com

Abstract

In this paper we study the k-delivery traveling salesman problem (TSP) on trees, a variant of the non-preemptive capacitated vehicle routing problem with pickups and deliveries. We are given n pickup locations and n delivery locations on trees, with exactly one item at each pickup location. The k-delivery TSP is to find a minimum length tour by a vehicle of finite capacity k to pick up and deliver exactly one item to each delivery location. We show that an optimal solution for the k-delivery TSP on paths can be found that allows succinct representations of the routes. By exploring the symmetry inherent in the k-delivery TSP, we design a $\frac{5}{3}$-approximation algorithm for the k-delivery TSP on trees of arbitrary heights. The ratio can be improved to $(\frac{3}{2} - \frac{1}{2k})$ for the problem on trees of height 2. The developed algorithms are based on the following observation: under certain conditions, it makes sense for a non-empty vehicle to turn around and pick up additional loads.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases k-delivery traveling salesman problem approximation algorithms

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2012.325

1 Introduction

In the capacitated vehicle routing problem with pickups and deliveries (CVRPPD), we are given an undirected complete graph $G=(V, E)$, with edge costs satisfying the triangle inequality, and each vertex is associated with a pickup or a delivery load. A vehicle, or a fleet of vehicles with finite capacity k, traverses the edges of G and serves all the pickup and delivery nodes. During the traversal, the vehicle should never exceed its capacity. The objective is to find a minimum length tour, which starts at the depot and traverses all the nodes of G while satisfying the pickup/delivery demands of the nodes. CVRPPD can capture many real-world transportation and distribution problems. It is a generalization of the traveling salesman problem, and is therefore NP-hard. The problem can further be generalized based on whether preemption of loads of vehicles is allowed. The preemptive version allows the vehicle to temporarily unload the items at an intermediate point of the network which will later be picked up for the delivery. In this note we consider the case where the preemption of loads is not allowed. It is not difficult to see that any solution to

* This work was partially supported by MITACS and NSERC.

32nd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2012).
Editors: D. D'Souza, J. Radhakrishnan, and K. Telikepalli; pp. 325–336
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

CVRPPD without preemption is an upper bound on the optimal solution where preemption is allowed. A survey of these problems can be found in [5, 6].

In this paper, we focus on a variant of CVRP with pickups and deliveries, called the k-delivery traveling salesman problem.

1.1 k-Delivery traveling salesman problem (k-delivery TSP)

Here the vertex set $V = V_p \cup V_d$ is partitioned into a set V_p of *pickup vertices* and a set V_d of *delivery vertices*. Each vertex of V_p provides an item, and each vertex of V_d requires one item. A vehicle with capacity k starts from a depot and gathers items from the pickup vertices, and delivers them to the delivery vertices before returning to the depot. All the items are identical, therefore an item picked up from a vertex in V_p can be delivered to any vertex in V_d. We assume here $|V_p| = |V_d|$, and the capacity $k < |V_p|$. The objective of the k-delivery TSP is to determine a minimum cost tour of the nodes of G subject to the vehicle capacity constraint.

The k-delivery TSP is also called the capacitated pickup and delivery TSP (CPDTSP) in [8]. Lim et al. [8] showed that the problem can be solved optimally in time $O(n^2/min(k, n))$ on paths, where n is the number of vertices of G. The authors proved in the same paper that the k-delivery TSP is NP-hard in the strong sense even for trees with height 2, using a reduction from the 3-partition problem. A 2-approximation algorithm for the k-delivery TSP on trees is later given in [7]. This algorithm follows a rule that the vehicle would continue to pick up (or deliver) items if possible. The best known approximation ratio of 5 for the k-delivery TSP on general graphs is due to Charikar et al. in [4]. The first constant factor approximation algorithm for the problem was proposed by Chalasani et al. in [2]. A restricted version of the problem was considered in [1].

1.2 Our results

Our results described in this paper are summarized as follows:

1. For the k-delivery TSP on paths, a linear time algorithm is proposed to find the optimal solution. This improves the running time $O(n^2/k)$ of the algorithm in [8].
2. For the k-delivery TSP on trees, we improve the approximation ratio to $\frac{5}{3}$. The best known approximation ratio 2 is due to Lim et al. in [7]. This ratio can be improved to $\frac{3}{2} - \frac{1}{2k}$ for trees with height 2.

All the developed algorithms use a *come-back* rule. Under this rule the vehicle would not be allowed to cross a particular edge e if some certain condition is met. According to the come-back rule, the vehicle postpones delivering its load immediately and picks up more items instead for a better solution. As a consequence, on its way to pick up more loads, the vehicle may traverse several edges without delivering any items, even when it has a non-empty load. This is somewhat contrary to our intuition. However, in this paper we show that this strategy is effective for the k-delivery TSP on paths and trees.

2 Lower bounds

We describe a widely used lower bound, called *the flow bound*, for tree networks for the considered problems. Some notations used to describe the lower bound are similar to the one used in [3] and [8].

It is assumed in this paper that the depot node is located at the root of the tree network. The vehicle starts and ends its tour at the depot node. The flow bound for the k-delivery TSP on trees is described below. Denote a subtree rooted at vertex u by T_u and the parent of u by $p(u)$. For each vertex v of G, we associate a label $a(v)$ with it, which is set to 1 if it is a pickup vertex, and -1 if it is a delivery vertex. For trees, we define $g(v) = \sum_{t \in T_v} a(t)$ to be the net number of items of all the vertices in T_v. Then any k-delivery TSP tour must traverse edge $e = (p(v), v)$ at least $2max\{\lceil \frac{|g(v)|}{k} \rceil, 1\}$ times.

Note here that the flow bound described above is a preemptive lower bound. For all the problems we solved, we bound our solutions to the optimum of the *preemptive* version of these problems.

3 Linear time algorithm for the k-delivery TSP on paths

As implied by the flow bound, an optimal solution of the problem on paths may contain $O(n^2/k)$ edges. The algorithm in [8] takes $O(n^2/k)$ time since it constructs the final optimal solution by explicitly enumerating its edge sequences. However, in this paper we show that our optimal schedule has a succinct linear size representation and the optimal schedule can be computed in linear time.

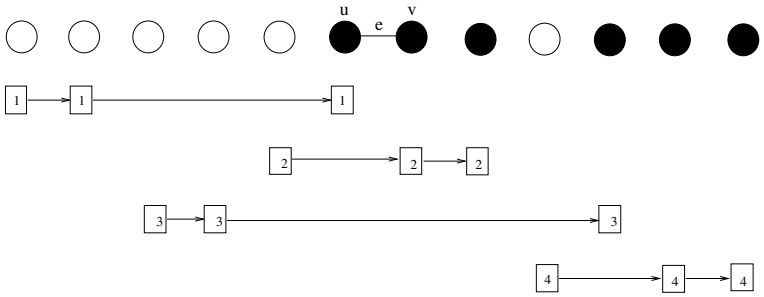

Figure 1 A succinct representation of an optimal solution for the k-delivery TSP on a path where k=2. A white circle represents a pickup vertex and a black circle represents a delivery vertex.

In the succinct representation (see Figure 1), the solution comprises several subroutes, $r_1, r_2, \cdots, r_t$. Each vertex is marked to be picked up or delivered by one such subroute. For each subroute r_i, where $1 \leq i \leq t$, the vehicle travels along the path from the leftmost vertex to the rightmost vertex, which are marked by this subtour. In the meantime, the vehicle services all the vertices marked by the subtour. After reaching the rightmost vertex, the vehicle comes back to the leftmost vertex marked by the next subtour r_{i+1}, and the above process continues. It is clear that such a representation needs only $O(n)$ space.

Let $P =< v_1, v_2, ..., v_n >$ be the path network where the depot is located at v_1. Note that v_1 could be a pickup or a delivery vertex. Define $g(v_i) = \sum_{t=1}^{i} a(v_t)$ to be the net number of items among vertices to the left of v (including v). Using the values $g(v)$ of the nodes v of P, we partition P into path segments P_i with endpoints q_i and q'_i, $i = 1, 2, ..$, where $g(q'_i) = 0$, and $g(w)$ is nonzero for all the remaining vertices w of the segment P_i. The vehicle traverses P_i from the left if $g(q_i) > 0$, otherwise P_i is traversed from the right. Note that any feasible schedule will traverse every edge of the path at least twice. Therefore, without any loss of generality we assume that, except the last vertex, $g(w) > 0$ for all w of P. The main contribution of our algorithm is a strategy called *come-back-path*. The procedure scans the path from left to right and maintains a route number and two stacks. The route number

represents the number of the current subtour, and is used to mark each vertex for a succinct representation. A stack called *vehicle-stack* is used to simulate the behaviour of the vehicle; pushing a vertex to vehicle-stack has the same effect as loading one item from the vertex into the vehicle, and popping a vertex from vehicle-stack means the vehicle delivers one item to the vertex. During the processing, a pickup vertex is put into vehicle-stack if the current vehicle load is less than k, otherwise, it is stored in another stack called *repository-stack*.

The algorithm can be briefly stated as follows:

Procedure come-back-path {
Scan the vertices from left to right.
Step (a) (The vertex is a pickup vertex.) The vertex is pushed into vehicle-stack if the size of the stack is less than k. Otherwise it is pushed into repository-stack.
Step (b) (The vertex is a delivery vertex.) An item from vehicle-stack is popped and delivered to the delivery vertex.
- If the next vertex to be processed is also a delivery vertex, check if the total size of the two stacks is a multiple of k.
- If the condition is true, the action is not to proceed to the next delivery vertex. Instead a new subroute is created, and the vehicle goes back to pick up more loads. This is accomplished by transferring loads from repository-stack to fill up vehicle-stack. The size of vehicle-stack is now k.

}

We define $b_1(t)$ and $b_2(t)$ to be the number of items in vehicle-stack and repository-stack respectively at time step t. By our assumption, the vehicle is moving from left to right. Let the vehicle reach a delivery vertex u at time step t'. The come-back-path strategy applies when the next vertex v is also a delivery vertex. The triggering condition for the come-back-path strategy here is whether $(b_1(t') + b_2(t')) \bmod k = 0$. In other words, the vehicle is allowed to cross edge $e = (u, v)$ from delivery vertex u to delivery vertex v only if $(b_1(t') + b_2(t')) \bmod k \neq 0$. Otherwise, the current subtour is terminated and the vehicle comes back to pick up the topmost $(b_2(t') \bmod k)$ items in repository-stack. To ease the analysis, we assume that the vehicle picks up these items on its way back from u (when the vehicle is moving leftwards). This is achieved by increasing the route number by 1 and transferring the topmost $b_2(t') \bmod k$ items of repository-stack to vehicle-stack.

The come-back-path strategy guarantees that the solution produced by this procedure satisfies the flow bound for the k-delivery TSP. This is stated in the following lemma.

▶ **Lemma 1.** *For an edge e of the path segment, let the subtours found by procedure come-back-path passing through e be $r_1, r_2, \cdots, r_m$. The vehicle crosses e from left to right in each of the routes $r_3, \cdots, r_m$, with exactly k items. Moreover, the vehicle carries the rest (more than k) items in r_1 and r_2.*

Proof. Omitted. ◀

The following theorem is implied by the above lemma.

▶ **Theorem 2.** *For an edge $e = (u, v)$ of the path segment, the tour determined by procedure come-back-path crosses e exactly $2\lceil \frac{|g(u)|}{k} \rceil$ times which is the* flow bound *of e.*

4 A $\frac{5}{3}$-approximation algorithm for the k-delivery TSP on trees

Wang et al. [8] showed that the problem is NP-complete in the strong sense even on trees of height 2. In this section we present a $\frac{5}{3}$-approximation algorithm called *half-load* for the

k-delivery TSP on trees of arbitrary heights. The half-load algorithm is also based on the come-back rule and it achieves a $(\frac{3}{2} - \frac{1}{2k})$-approximation for the k-delivery TSP on trees of height 2. Our initial focus is on height 2 tree networks. The methodology will then be generalized to accommodate arbitrary height tree networks.

Let $T = (V, E)$ denote the tree network containing n vertices. Let V_p and V_d be the set of pickup and delivery vertices of T. We are assuming that the tree network is rooted and the depot location is at the root. We call a subtree T_u positive (negative) if it contains more (less) vertices from V_p than from V_d. An edge $e = (p(u), u)$ or the vertex u is positive (negative) if subtree T_u is positive (negative). Here $p(u)$ is the parent node of the non-root node u. For an edge $e = (p(u), u)$, we say some vertices/items are picked up from (delivered to) e or u, if we pick up (deliver) these vertices/items from (to) the subtree T_u; we say e has a pickup (delivery) load of $g(e)$ if exactly $g(e)$ (which is equal to $|g(u)|$) net number of items need to be picked (delivered) through e. We say the vehicle visits (crosses, traverses) e if the vehicle moves from $p(u)$ to u. Finally we write $LB_e = \lceil \frac{g(e)}{k} \rceil$. Any optimal solution to the k-delivery TSP will traverse edge e at least $2LB_e$ times.

4.1 Exploring the symmetry of the k-delivery TSP

Our improvements for the k-delivery TSP on trees explore the symmetry inherent in the problem. In the k-delivery TSP, the underlying graph has only two types of vertices, it's not difficult to see that if we flip the type of each vertex, to get a new graph, say $\overline{G}$, then the k-delivery TSP has the following property.

▶ **Lemma 3.** *Any feasible solution of the k-delivery TSP on $\overline{G}$ can be converted to a feasible solution of G with the same edge cost, by reversing its edge directions of the routes.*

This property allows us to design approximation algorithms for the k-delivery TSP on trees in the following way. We partition edge set E of the graph into two sets $D(G)$ and $E - D(G)$ based on some definition D. Assume $D(\overline{G}) = E - D(G)$. We have

▶ **Lemma 4.** *Let A be an algorithm that produces tours where each edge in $D(G)$ and $E - D(G)$ is traversed at most α and β times LB_e respectively. Then there exists a solution generated by A with cost no more than $\sum_e \frac{\alpha+\beta}{2} LB_e$.*

Proof. Let τ_1 be a tour after applying algorithm A on G. Flip the type of each vertex and obtain a second tour τ_2 by running the same algorithm on $\overline{G}$. Since $D(G) = E - D(\overline{G})$, each edge e would be crossed at most $(\alpha + \beta) \cdot LB_e$ times in the two tours. The tour with the smaller cost is the desired solution. ◀

Assume algorithm A stated in Lemma 4 is particularly "good" to $D(G)$, in the sense that $\alpha < \beta$. Then Lemma 4 shows that the approximation ratio of A can be further reduced to $\frac{\alpha+\beta}{2}$. In the proposed half-load algorithm, $D(G)$ is defined to be the set of positive edges of G. For the k-delivery TSP on trees of height 2, this algorithm achieves $\alpha = 1$ and $\beta = 2$.

4.2 Two phases of the half-load algorithm

The half-load algorithm being proposed here for the k-delivery TSP on trees uses a strategy called *come-back-tree*. The algorithm contains two phases, the planning phase and the route generating phase. In the planning phase, the original graph G is transformed to a multi-graph G' as follows. In G' each positive tree edge $e = (p(u), u)$ is split into several pseudo edges with total load equal to $g(e)$. Intuitively a pseudo edge $e = (p(u), u)$ records the number

of items that to be collected from T_u during one visit from $p(u)$ to u. Examples of the transformation are given in Figures 2 and 3.

A pseudo edge is treated as a scheduling unit in the second route generating phase. Each time when the vehicle crosses a positive tree edge $e = (p(u), u)$ to service nodes in $T(u)$, we make sure that the vehicle picks up the full load assigned to a distinct pseudo edge between $p(u)$ and u. Thus under the half-load algorithm the number of times e is crossed, is determined by the number of pseudo edges incident to $p(u)$ and u.

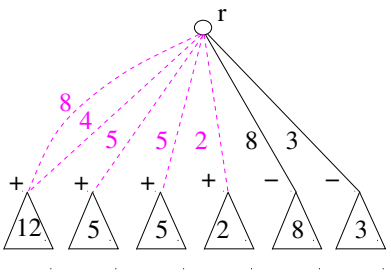

Figure 2 An example of the pseudo edges on a tree of height 2. k=8.

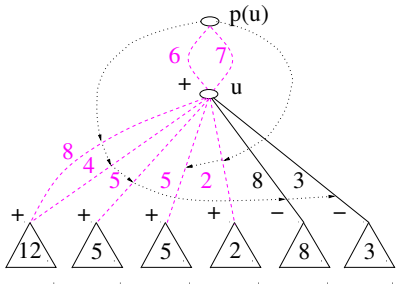

Figure 3 An example of building pseudo edges for a vertex u. k=8.

In the example in Figure 2, the capacity k is set to 8. A pseudo edge of the transformed tree is represented by a dashed line, and a subtree is represented by a triangle with a number showing the net number of items inside this subtree. We assume in G' all the pickup and delivery demands are on the leaves. This can be done easily by adding a pseudo vertex u' for each non-leaf vertex u of G, and attaching an edge between u and u' with zero cost. Since the first positive tree edge in Figure 2 has a load of more than k, it is split into two edges with loads 8 and 4 respectively. Other tree edges do not split, since they are either negative or their loads are less than k.

Note that in the planning phase issues related to route generation, such as the route feasibility constraints, are not considered. The final schedule is computed in the second phase by nested recursive calls of several procedures. A brief description of the second phase is as follows. First a procedure called *come-back-tree* that adopts the come-back rule is applied on the first level of the tree to get the servicing sequence of the involved pseudo and negative tree edges. During the process, two recursive procedures, *pickup* and *deliver*, are called to service the pseudo and negative tree edges respectively. Besides some special processing for picking up and delivering items, procedure come-back-tree is invoked in both procedures on the second level of the tree. This process is continued until a leaf node is met. In all the procedures the vehicle capacity constraint and the non-preemptive scheduling constraint are always obeyed. This guarantees the feasibility of the generated routes.

4.3 Planning phase of the half-load algorithm

In this subsection we discuss in more detail the procedure to build pseudo edges. On a tree of height 2 (Figure 2), a positive edge $e = (p(u), u)$ is split into $\lceil \frac{g(u)}{k} \rceil$ pseudo edges. Except the last, all pseudo edges have a load of k items. The last pseudo edge contains the residual $g(u)$ mod k items.

For trees with arbitrary heights, the pseudo edges and their links are built recursively in a bottom-up fashion. The pseudo edges are created from the leaves, and then spread to higher levels of the tree. For a positive vertex u, let L_+ and L_- consist of the pseudo and

negative tree edges from u to its children respectively. The pseudo edges incident to $p(u)$ and u are generated through the following steps.

First, we select a minimal size subset S of edges from L_+ whose load is just enough to service the edges in L_-. The edges in S and L_- are then merged together to form a new pseudo edge e_1. This merging is in the sense that linking the edges in S and L_- together as a group and setting $g(e_1)$ to be the total load of the involved edges. Let S' consist of e_1 and the remaining pseudo edges in L_+.

Second, a procedure called *merge* is applied on S' to produce a new set S'' of pseudo edges. The merge procedure repeatedly merges two arbitrary pseudo edges with loads $\leq \frac{k}{2}$. It is easy to see that each pseudo edge (except possibly one) in S'' should have load more than $\frac{k}{2}$. We then create a new edge e' between $p(u)$ and u for each edge $e \in S''$. The two edges are linked together through a children link of e' to remember where the load of e' is from. In the second phase, when the load of e' is requested, the children and group links can be used to locate all the edges ever participated in generating e (reachable from e by following the links). An example of building pseudo edges is shown in Figure 3.

4.4 Come-back-tree strategy for the k-delivery TSP on trees

The come-back-tree strategy is applied in the scenario when we are given two lists L_+ and L_- and a vertex u where the vehicle with load α is currently at. The negative edges of L_- will be serviced by the items from the positive pseudo edges of L_+ and load α of the vehicle. To ease the explanation, we assume that the tree is of height 2. This procedure also works for trees with arbitrary heights. The explanations for the general tree networks will be given later in this paper.

We maintain two pointers z_+ and z_- pointing to the heads of lists L_+ and L_- respectively. Recall that under the come-back rule, the vehicle may not be allowed to cross an edge if some condition occurs. The triggering condition is raised when the load of the next pseudo edge pointed to by z_+ fits the remaining capacity of the vehicle. In this case the vehicle will come back from delivering any new load and instead pick up the entire load from the pseudo edge pointed to by z_+. When the triggering condition is not raised, the vehicle begins to deliver all or part of its current load to the subtree pointed to by z_-. When the load of a pseudo edge is fully consumed, this edge is removed from L_+. Similarly when the subtree pointed to by z_- is fully served, this edge is removed from L_-. The above process continues until L_+ or L_- becomes empty. The procedure then returns the route constructed and the remaining edges in L_+ or L_-.

We explain the algorithm in [7] (which we call the full-load algorithm) as follows. For a vertex u, let $L_+ = \{c_1^+, c_2^+, \cdots, c_i^+\}$ and $L_- = \{c_{i+1}^-, \cdots, c_l^-\}$ be two lists consisting of the positive and negative children of u respectively. We also assume that in L_+ and L_- the vertices and the subtrees rooted at these vertices are sorted arbitrarily. In the full-load algorithm, the vehicle would pick up k items at a time *consecutively* from the subtrees $T_{c_1^+}, \cdots, T_{c_i^+}$, and deliver the k items consecutively to the subtrees $T_{c_{i+1}^-}, \cdots, T_{c_l^-}$. This also means that the vehicle services the subtrees in the sorted order, and when the vehicle starts to pick up (deliver), the vehicle would continue to load (consume) items if possible.

An example is given in Figure 4 to show the effectiveness of our come-back-tree strategy. In this example, $k = 8$, the input to the algorithm is a list L_+ which contains edges $e_1, e_2, \cdots, e_5$ with loads 6, 6, 6, 6, 2 respectively, and a list L_- which contains edges e'_1, e'_2, e'_3, e'_4 with loads 5, 7, 7, 7 respectively. It is not difficult to verify that only e'_3 will be traversed twice if the come-back-tree strategy is followed. Therefore $e_1, \cdots, e_5, e'_1, \cdots, e'_4$ will be traversed 1, 1, 1, 1, 1, 1, 1, 2, 1 times respectively. If we flip the types of the vertices and apply the same rule,

then only e_1 needs to be visited twice. In this case $e_1, \cdots, e_5, e'_1, \cdots, e'_4$ will be traversed 2, 1, 1, 1, 1, 1, 1, 1 times respectively. However, if we follow the full-load algorithm, then $e_1, \cdots, e_5, e'_1, \cdots, e'_4$ will be traversed 1, 2, 2, 1, 1, 1, 2, 2, 2 times respectively. It is not difficult to see that the come-back rule outperforms the full-load strategy in this example.

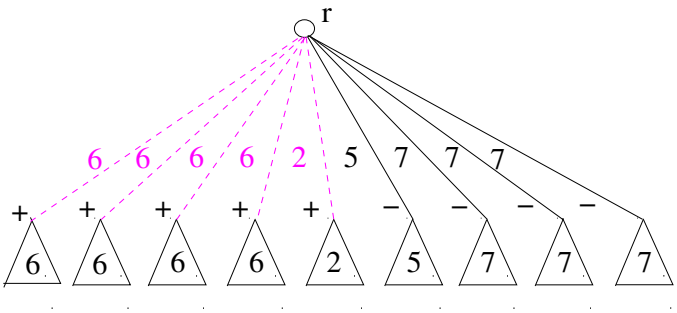

Figure 4 An example of the come-back rule with k=8.

On average, $e_1, \cdots, e_5, e'_1, \cdots, e'_4$ will be visited $\frac{3}{2}$, 1, 1, 1, 1, 1, 1, $\frac{3}{2}$, 1 times respectively in the two tours (one using G and one using $\overline{G}$) after applying the come-back rule. For trees of height 2, picking up (delivering) vertices through a pseudo edge (tree edge) can be done optimally (Figure 2). The following lemma shows that the approximation ratio of the half-load algorithm for the k-delivery TSP on trees of height 2 is $\frac{3}{2}$.

▶ **Lemma 5.** *The half-load algorithm approximates the k-delivery TSP on trees of height 2 within $\frac{3}{2}$ of its optimum.*

Proof. According to the come-back rule, each positive pseudo edge is only visited once, and the vehicle starts to deliver its load, if and only if its load plus the load of the next positive pseudo edge is more than k. Let $r_1, r_2, \cdots, r_t$ be all the routes that pass through edge $e = (p(u), u)$. We show that $t \leq 2LB_e$. It is trivially true if $t = 1$ or 2 since $LB_e \geq 1$, so we assume $t \geq 3$ in what follows. It is clear that, for $q = 1, 2, \cdots$ $(2q + 1 \leq t)$, the half-load algorithm makes the vehicle carry a total of at least $k + 1$ units from T_u in any two consecutive routes, r_{2q-1} and r_{2q}, when crossing e from $p(u)$ to u.

Case 1: t is odd. Excluding r_t, the vehicle dumped at least $(t-1)(k+1)/2 \geq (t-1)k/2+1$ items to T_u. Thus $LB_e = \lceil g(e)/k \rceil \geq \frac{t-1}{2} + 1 = \frac{t+1}{2}$, which implies that $t \leq 2LB_e - 1$.

Case 2: t is even. Excluding r_{t-1} and r_t, the vehicle dumped at least $(t-2)(k+1)/2 \geq (t-2)k/2 + 1/2$ items to T_u. Therefore $LB_e = \lceil g(e)/k \rceil \geq \frac{t-2}{2} + 1 = \frac{t}{2}$, which implies that $t \leq 2LB_e$.

Because of the symmetry, e is visited at most $3 * LB_e$ in the two solutions.

◀

We can improve the result as descirbed below. The proof is omitted.

▶ **Lemma 6.** *There is a $(\frac{3}{2} - \frac{1}{2k})$-approximation algorithm for the k-delivery TSP on trees of height 2.*

4.5 Pickup procedure for the half-load algorithm on trees of arbitrary heights

Assume L_+ and L_- contain the pseudo and tree edges respectively at the first level of the transformed tree. Our solution can be computed by invoking *come-back-tree*(α,L_+, L_-) with

$\alpha = 0$, where the first argument indicates that the vehicle is initially empty. The half-load algorithm would be complete if we have appropriate pickup and deliver procedures.

Both our pickup and deliver procedures run recursively. Given a positive pseudo edge $e = (p(u), u)$, the pickup procedure is applied when the vehicle with a load α tries to cross e to pick up $g(e)$ items from T_u. Let S be the set of pseudo edges involved in generating e, and let the pseudo edges of S induce subtree T_S (reachable from e). The vehicle should service all the delivery vertices of T_S and leave e (from u to $p(u)$) with $\alpha + g(e)$ items. This is achieved by calling the come-back-tree procedure on edges from u to its children. When this call is completed and there are still some pseudo edges unserviced, the pickup procedure will be invoked separately on these edges to collect their loads. Note that because of the non-preemptive nature, the vehicle may enter e and leave e with no items in common. An example of the pickup scheme is shown in Figure 5.

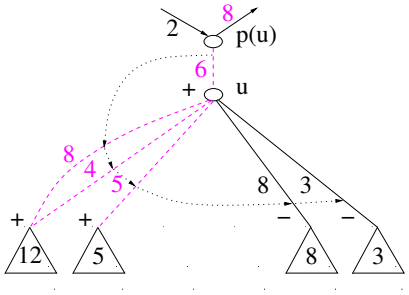

Figure 5 An example of picking up vertices from an edge $e = (p(u), u)$. $k = 8$ and the vehicle has an initial load of 2.

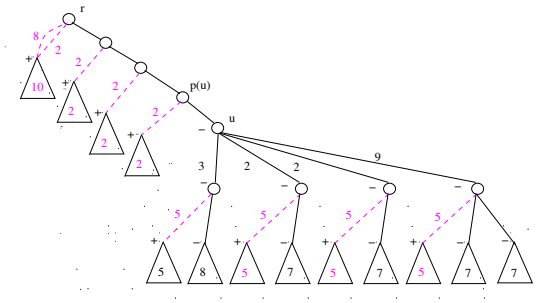

Figure 6 An example of delivering vertices to an edge $e = (p(u), u)$. $k = 8$ and r is the root of the tree.

The example in Figure 5 is a subgraph of the example in Figure 3. We want to show how to pick up 6 vertices from the first positive pseudo edge $e_1 = (p(u), u)$ in Figure 3. Before visiting e_1, the vehicle is assumed to already carry 2 vertices; since $k = 8$, the vehicle should be able to leave $p(u)$ with 8 vertices. We only consider the schedule among the edges from u to its children involved in generating e_1. Let $e_1^+, e_2^+, e_3^+, e_1^-$ and e_2^- be the 5 edges from u to its children as shown in Figure 5, respectively from left to right. In this example, the vehicle visits these edges in the order of e_1^-, e_1^+, e_1^-, e_2^+, e_2^-, e_3^+. The 8 vertices leaving $p(u)$ with the vehicle are from e_2^+ and e_3^+.

The route implied by the planning phase (a bottom-up approach) for T_S assumes zero initial vehicle load. We need to show that the tour is still feasible if the vehicle already has a load. Recall that in the come-back-tree procedure, further picking up of items through a positive pseudo edge e is allowed only if the load of e (i.e. $g(e)$) fits the remaining vehicle capacity. Therefore, when the vehicle decides to pick up $g(e)$ items, it is always guaranteed that the vehicle can service all the items on T_S and come back to $p(u)$ without violating the capacity constraint. This also shows the effectiveness of our planning phase.

4.6 Deliver procedure for the half-load algorithm on trees of arbitrary heights

The deliver procedure is applied when the vehicle with α items tries to service a negative tree edge $e = (p(u), u)$. It needs an additional parameter L'_+ which is a list of positive pseudo edges not in T_u, from which $g(e)$ items are delivered to the vertices of T_u. More specifically,

L'_+ contains the pseudo edges remaining when procedure deliver is invoked on e during a previous execution of procedure come-back-tree.

If u is a leaf node, then we simply deliver one item in the vehicle to service u. If u is not a leaf, we descend one level and calls procedure come-back-tree to service all the edges from u to its children. Different from the case for trees of height 2, some special processing is needed for trees with arbitrary heights.

First, before servicing the delivery vertices of T_u, the edges of L'_+ are attached to the end of list L_+ which consists of the positive pseudo edges from u to its children. According to the come-back-tree strategy, when the vehicle gathers the vertices from the last positive pseudo edge of the original L_+, the deliver procedure requires the information of the next positive pseudo edge of L'_+. This information will be used to decide whether the vehicle should come back and pick up some more supplies from outside T_u, to guarantee that every negative tree edge is traversed no more than twice the optimum.

Second, after the stitching of L'_+ and L_+, the merge procedure is applied on the new list before invoking procedure come-back-tree. Thus during the execution of the half-load algorithm, L_+ always contains at most one pseudo edge with load of no more than $\lceil \frac{k}{2} \rceil$. Since the algorithm runs recursively, the positive pseudo edges at higher levels of the tree are merged first. The merging, and also the way of merging in the algorithm are important for the performance guarantee. A formal proof for the correctness of the strategy is shown in Lemma 7.

▶ **Lemma 7.** *For two consecutive visits to a negative tree edge $e = (p(u), u)$ in the direction of $p(u) \to u$, the vehicle carries more than k vertices from outside T_u.*

Proof. In the algorithm, after stitching together L'_+ and L_+, the new list L_+ includes all the available positive pseudo edges (from outside T_u) which can be used to service e. For trees with arbitrary heights, the crossing of e from $p(u)$ to u might be triggered by the come-back rule when it tries to service a negative edge inside T_u. In this case the vehicle would come back and cross e from u to $p(u)$ to pick up more items from outside T_u. We claim that the pickup vertices (not in T_u) passing through e from $p(u)$ to u, are picked up consecutively from the edges of list L'_+ starting from its head, and all except possibly one of the edges in L'_+ contain more than $\lceil \frac{k}{2} \rceil$ vertices.

The first part of the claim holds, because in the deliver procedure, the edges of L'_+ is attached to the end of L_+. Therefore, after all the positive edges in the original L_+ are processed, the pickup items crossing e from $p(u)$ to u, must be picked up consecutively from the edges of list L'_+. The latter part of the claim holds, because the deliver procedure gives merging priority to higher level pseudo edges of the tree. So no matter L'_+ is obtained during the planning phase, or during an earlier call of the pickup or deliver procedure, the merging step has always been deployed on L'_+. This completes the proof.

◀

While the proof follows naturally from the algorithm, in the following we further explain why, during the delivery process, our way of merging is important for the performance guarantee. Assume before servicing a negative edge e' in T_u, the come-back rule is triggered and the vehicle crosses e from u to $p(u)$ to pick up more items, say from a pseudo edge e'' outside T_u. The vehicle collects the items from e'' and returns to T_u to service e'. Since the tree is of arbitrary height, the path P between e' and e'' may contain multiple negative edges. We need to guarantee that for all such visits (except possibly one) on an edge e, the vehicle carries more than $\frac{k}{2}$ distinct items.

Consider the tree in Figure 6. Let the four negative children of u be v_1, v_2, v_3, and v_4 respectively (from left to right). For edge $e = (p(u), u)$, $LB_e = 2$. Assume the pseudo edges in Figure 6 are not merged. Before serving the negative edge in T_{v_1}, the vehicle has a load of 8 and the first two pseudo edges in the list have loads 5 and 2 respectively. Before serving each negative edge in T_{v_2}, T_{v_3} and T_{v_4}, the vehicle has a load of 7 and the first two positive pseudo edges in the list have loads 5 and 2 respectively. It is easy to see that in this example, edge $e = (p(u), u)$ is traversed 2.5 times of LB_e if the come-back rule is deployed but the merging is not applied. If the priority of merging is not given to edges at higher levels of a tree, e.g., if each positive pseudo edge with load 5 is merged with a positive edge with load 2, then e will also be traversed 2.5 times LB_e.

▶ **Lemma 8.** *The half-load algorithm is a 2-approximation for the k-delivery TSP on trees.*

Proof. First, it is not difficult to see that the tour is feasible, since in the tour all the vertices are served and the capacity constraint is always obeyed. Given an edge e of the tree, we prove that the number of traversals the half-load algorithm makes on $e = (p(u), u)$ is no more than $2LB_e$. We have the following two cases:

Case 1: T_u is positive. Let list L_+ contain all the pseudo edges on e. Each pseudo edge e' in L_+ is traversed only once in the algorithm, thus the number of traversals the algorithm makes on e is just the number of pseudo edges in L_+. According to our merging rule in the planning phase, all pseudo edges, except possibly one, must have loads more than $\frac{k}{2}$. Assume the vehicle carries exactly $\lfloor \frac{k}{2} \rfloor + 1$ vertices each time during the first $|L_+| - 1$ visits on e. This is the worst case our algorithm could have on e. Let the last edge in L_+ be e'. When k is odd, the vehicle should carry at least $\lfloor \frac{k}{2} \rfloor + 1$ vertices during the last visit on e. In the following we show that $|L_+| \leq 2LB_e - 1$ when k is even. The case when k is odd can be argued similarly.

Subcase 1: $|L_+|$ is odd. For the first $|L_+| - 1$ visits, the lower bound of the number of traversals on e (part of LB_e) is $\frac{|L_+|-1}{2}$, if excluding $|L_+| - 1$ vertices from these visits. There must be one additional visit for these excluded vertices and the vertices in e'. So in total $LB_e \geq \frac{|L_+|-1}{2} + 1 = \frac{|L_+|+1}{2}$, which is equivalent to $|L_+| \leq 2LB_e - 1$.

Subcase 2: $|L_+|$ is even. For the first $|L_+| - 2$ visits, if excluding $|L_+| - 2$ vertices, the lower bound of the number of traversals on e (part of LB_e) is $\frac{|L_+|-2}{2}$. The vehicle must also carry more than k vertices during the last two visits, so there must also be two additional visits in LB_e. Thus in total $LB_e \geq \frac{|L_+|-2}{2} + 2 = \frac{|L_+|+2}{2}$, which is equivalent to $|L_+| \leq 2LB_e - 2$.

Case 2: T_u is negative. In this case we may have two visits on e, where the vehicle dumps less than $\frac{k}{2}$ pickup vertices on e. However, according to Lemma 7, the total vehicle load is more than k during every two consecutive traversals on e. The proof then follows much in the same way as the proof of Lemma 5. ◀

4.7 A $\frac{5}{3}$-approximation for the k-delivery TSP on trees of arbitrary heights

In this section, we prove that the smallest cost tour from three tours, obtained by applying the full-load algorithm in [7] on G (or $\overline{G}$), and the half-load algorithm on G and $\overline{G}$, is bounded by $\frac{5}{3}$ times of the optimum.

The full-load algorithm [7] has its advantages and disadvantages. The advantage is that after the vehicle gathers exactly k items, it traverses several subsequent edges optimally. The

disadvantage is that, before the vehicle is full, and after delivering these k vertices begins, some edges of the tree might have to be traversed twice of their optimum. It is not difficult to see that the half-load algorithm exchanges its advantages and disadvantages with the full-load algorithm. This explains intuitively why we balance three tours as our final solution. We formally prove the approximation ratio $\frac{5}{3}$ of our final solution in Theorem 9.

▶ **Theorem 9.** *The final solution is a $\frac{5}{3}$-approximation for the k-delivery TSP on trees of general heights.*

Proof. Given an edge $e = (p(u), u)$, let Sol_1 be the solution after applying the full-load algorithm on G (or $\overline{G}$), and Sol_2 and Sol_3 be the solutions after applying the half-load algorithm on G and $\overline{G}$ respectively. As shown in [7], e is only traversed at most $LB_e + 1$ times in Sol_1. Note here that $\overline{G}$ is obtained from G by simply relabeling the pickup vertices as delivery vertices and vice versa.

If e is positive, e is visited at most $2LB_e - 1$ times in Sol_2 according to Lemma 8. Since e is negative in $\overline{G}$, e is crossed at most $2LB_e$ times in Sol_3 (established in the proof of Lemma 5). Because of the symmetry, e is traversed at most $5LB_e$ times in the three tours for any e. ◀

5 Conclusions and future work

In this paper we propose a rule called the come-back rule for a variant of the capacitated vehicle routing problem with pickups and deliveries on trees. We illustrate this rule by an optimal algorithm for the k-delivery TSP on paths. We show this rule can be utilized to improve the approximation ratio for the k-delivery TSP on trees. Observing that the Dial-a-Ride problem [3] is also symmetric as in Lemma 3, in the future we will further investigate the Dial-a-Ride problem on trees and the k-delivery TSP on general graphs.

References

1 T. Asano, N. Katoh and K. Kawashima A New Approximation Algorithm for the Capacitated Vehicle Routing Problem on a Tree. *Journal of Combinatorial Optimization*, Vol 5, Issue 2, Page:213-231, 2001.

2 P. Chalasani, and R. Motwani, Approximating capacitated routing and delivery problems. *SIAM Journal on Computing*, Vol 28, Issue 6, Page:2133-2149, 1999.

3 M. Charikar, B. Raghavachari, The Finite Capacity Dial-a-Ride Problem. *FOCS*, Page:458-467, 1998.

4 M. Charikar, S. Khuller, B. Raghavachari, Algorithms for Capacitated Vehicle Routing. *SIAM Journal on Computing*, Vol 31, Issue 3, Page:665-682, 2002.

5 D. O. Casco, B. L. Golden, and E. A. Wasil, Vehicle Routing with Backhauls: Models, Algorithms and Case Studies. *Vehicle Routing: Methods and Studies*, Eds. Golden and Assad, North Holland, 1988.

6 M.W.P. Savelsbergh and M. Sol, The general pickup and delivery problem. Transportation Science, Vol. 29, Pages:17-29, 1995.

7 A. Lim, F. Wang, Z. Xu, The Capacitated Traveling Salesman Problem with Pickups and Deliveries on a Tree. *ISAAC* LNCS, Vol. 3827, 1061-1070, 2005.

8 F. Wang, A. Lim and Z. Xu, The One-Commodity Pickup and Delivery Traveling Salesman Problem on a Path or a Tree. *Networks* Vol. 48, Issue 1, Page:24-35, 2006.