The Preemptive Swapping Problem on a Tree

Shoshana Anily
Faculty of Management, Tel-Aviv University, Tel-Aviv 69978, Israel

Michel Gendreau
CIRRELT, Université de Montréal, C.P. 6128, succursale Centre-ville, Montreal, Quebec, Canada H3C 3J7

Gilbert Laporte
GERAD, CIRRELT, and Canada Research Chair in Distribution Management, HEC Montréal, 3000, chemin de la Côte-Sainte-Catherine Montreal, Quebec, Canada H3T 2A7

This article considers the swapping problem on a tree. In this problem at most one object of some type is available at each vertex, and each vertex also requests at most one object of a given type. The total demand and the total supply of each object type are identical. The problem is to determine a minimum cost routing plan starting and ending at a prespecified vertex which is the depot, for a single vehicle of unit capacity and *m* object types, so that all vertex requests are satisfied. We consider the preemptive mode in which objects may be temporarily dropped along the way. It is shown that this problem is NP-hard. A heuristic with a worst-case performance ratio of 1.5 is developed. Finally, it is shown that the case where $m = 1$ is polynomially solvable. © 2011 Wiley Periodicals, Inc. NETWORKS, Vol. 58(2), 83–94 2011

Keywords: swapping problem; stacker crane problem; Transshipment

1. INTRODUCTION

The swapping problem (SP), introduced by [2], is defined as follows. Let $G = (V, E)$ be a connected undirected graph where $V = \{1, \ldots, n\}$ is the vertex set, vertex 1 represents a "depot," and $E \subseteq \{[v, w] : v, w \in V, v < w\}$ is the edge set. Each edge $e \in E$ has a non-negative cost or length $c_e \geq 0$. Let also $c(v, w)$ be the length of a shortest path between vertices v and w. Let $O = \{0, \ldots, m\}$ be a set of "object types." Object types $1, \ldots, m$ are real objects whereas an object of type 0, also called the "null object," is a dummy object introduced to simplify the presentation. With each vertex v is associated with a pair (a_v, b_v), where $a_v, b_v \in O$ and possibly $a_v = b_v$; a_v represents an object type supplied by v, while b_v represents an object type required by v. If v has no supply or no demand, this is represented by the null object $a_v = 0$ or $b_v = 0$. A vertex with $a_v = b_v = 0$ is called a "transshipment vertex." It is assumed that the graph is "balanced," that is, the total demand of each object type is equal to its total supply. In the SP, vehicles of finite capacity are used to swap objects between vertices in such a way that each vertex receives its required object. Vehicles start and end their trip empty at the depot and must perform all swapping operations while minimizing the total distance traveled.

In this definition of the SP, it is implicitly assumed that at most one object is supplied or required by any vertex. This assumption is not restrictive as long as the number of objects supplied or required by each vertex is uniformly bounded, as the vertices can be replicated to accommodate the case of multiple objects. As in [2], we assume that there is a single vehicle of unit capacity. In such problems, the objects can be "preemptive," "non-preemptive," or "mixed." In the preemptive case, all objects can be dropped at intermediate vertices while in the non-preemptive case, all objects remain in the vehicle between their origin and their destination. The mixed case allows some of the objects to be preemptive while the others are non-preemptive, as was done in [1, 2]. Here we assume that the preemptive mode applies.

Applications of the SP arise in the optimization of robot arm movements [4] and in printed circuit board assembly [5], as well as in the operations of automated guided vehicles and of material handling devices used in manufacturing systems (Katoh and Yano, 2006). The SP generalizes the classical *stacker crane problem* (SCP) (Frederickson et al., 1978) in which objects must be swapped between *specified* origin-destination pairs without preemption. The SCP is a special

Received January 2009; accepted January 2011
Correspondence to: G. Laporte; e-mail: gilbert.laporte@cirrelt.ca
Contract grant sponsor: Israel Institute of Business; Contract grant number: 38816-10
Contract grant sponsor: Canadian Natural Sciences and Engineering Research council; Contract grant number: 39682-10
DOI 10.1002/net.20451
Published online 2 August 2011 in Wiley Online Library (wileyonlinelibrary.com).

TABLE 1. Complexity results for the SCP and the SP.

	Stacker crane problem			Swapping problem		
Graph structure	Non-preemptive	Preemptive	Mixed	Non-Preemptive	Preemptive	Mixed
General	NP-hard[a]	NP-hard[b]	NP-hard	NP-hard[c]	NP-hard[c]	NP-hard[c]
Tree	NP-hard[d]	Polynomial[e]	NP-hard	NP-hard[d]	NP-hard[f]	NP-hard
Line	Polynomial[g]	Polynomial[h]	Polynomial[i]	Polynomial[i]	Polynomial[i]	Polynomial[i]
Circle	Polynomial[j]	Polynomial[j]	?	?	?	?

[a]Frederickson et al. [15] provide a heuristic with a worst-case performance ratio of 9/5.
[b]By reduction from the traveling salesman problem.
[c]Anily and Hassin [2] provide a heuristic with a worst-case performance ratio of 5/2. For $m = 2$, Chalasani and Motwani [9] improve this ratio to 2.
[d]Frederickson and Guan [11] provide several heuristics with bounded worst-case performance ratios.
[e]Frederickson and Guan [10].
[f]This article, Section 3. A heuristic with a worst-case performance ratio of 3/2 is provided in Section 4. The special case where $m = 1$ is polynomial, see Section 5 of this article.
[g]Atallah and Kosaraju [4], Ball and Magazine [5].
[h]Atallah and Kosaraju [4].
[i]Anily et al. [1].
[j]Atallah and Kosaraju [4].

case of the SP in which each object type is supplied and required by only one vertex. As the SCP is NP-hard, this is also the case for the SP. A branch-and-cut algorithm [7] and heuristics [8] have been developed for the SP on general graphs. Given the NP-hardness of the problem, it makes sense to investigate its complexity on simpler structures, such as trees. Katoh and Yano [15] mention applications of the SP and of the SCP on tree structures in the context of automated store and retrieval systems. Tree structures are also encountered in some river and highway networks [6] and in some pit mine railways [16]. Known complexity results on the SCP and the SP for various graph structures are summarized in Table 1.

The purpose of this article is to investigate the preemptive SP on a tree graph. In Section 2, we introduce some notation and preliminary results. We then show in Section 3 that the problem is NP-hard. In Section 4, we develop a polynomial time heuristic with a worst-case performance ratio of 1.5. Finally, in Section 5, we show that the single-type case ($m = 1$) can be solved in polynomial time. The complexity of the problem for any fixed value of $m > 1$ is an open question.

2. NOTATION AND PRELIMINARY RESULTS

The SP considered here is defined on a tree $T = (V, E)$. Everywhere in this article, except for the proof of Theorem 2, we assume that T is rooted at the depot. We denote by z^* the cost of an optimal SP solution, and by $T_v = (V_v, E_v)$ the subtree rooted at v. If v is a leaf of T, then $T_v = (\{v\}, \emptyset)$. As in [10, 11], we assume without loss of generality that all transhipment vertices, except the root, have a degree at least 3. Whereas the SP is defined on an undirected graph, its solution is better represented by a directed graph. A solution is a sequence of arcs, each associated with a certain object type. We will therefore consider the set of $2|E|$ arcs, denoted by A, connecting two adjacent vertices of T. For each edge $e = [u, v] \in E$, both the arc (u, v) directed from u to v, and the arc (v, u) directed from v to u, are in A. In addition, we define a set of $(m + 1)|A|$ "loaded arcs," denoted by $\bar{A}$, which associates an object type to each arc in A. Thus, $\bar{A} = \{(u, v)^i : (u, v) \in A, i \in O\}$. A "service path" of object $i \in O$ is a sequence of loaded arcs $(u_\ell, u_{\ell+1})^i \in \bar{A}$, $\ell = 1, \ldots, L$, which is traversed while the vehicle is loaded by object i, where the initial vertex on the path supplies object i, i.e., $a_{u_1} = i$, and the ending vertex on the path demands object i, that is, $b_{u_{L+1}} = i$. A "service cycle of the null object" is a sequence of loaded arcs $(u_\ell, u_{\ell+1})^0 \in \bar{A}$, for $\ell = 1, \ldots, L$, which is traversed while the vehicle is empty and $u_1 = u_{L+1}$.

A feasible solution to the SP consists of a set of service paths, as well as service cycles of the null object. The loaded arcs of a service path of object $i \neq 0$ do not necessarily occur consecutively in the solution because of the preemption option, but their order is preserved. In particular, the first loaded arc on the service path departs from a supply vertex of object i and the last loaded arc enters a demand vertex of object i. Suppose that V contains n_i vertices whose supply is i but whose demand is different from i. If a vehicle never unloads an object at a vertex to immediately load the same object, then any feasible solution contains exactly n_i service paths of object i. Each vertex v for which $i = a_v \neq b_v$ is a starting point for such a service path, and each vertex v for which $i = b_v \neq a_v$ is the end point of such a service path. The set of loaded arcs carrying the null object in a feasible solution forms service paths and service cycles of the null object. However, the loaded arcs of the null service paths are not necessarily traversed consecutively in a feasible solution, or in the order defined by the service path, as any vertex can be assumed to hold or require the null object.

As in [1], we construct the directed "multitype multigraph B," which we call the "basic graph." Multitype refers to the fact that an object type is associated with each arc, referred to as a loaded arc; multigraph means that several copies of the same loaded arc (origin, destination, object type) can

exist. More specifically, the loaded arcs of the basic graph can be partitioned according to their object type, that is, $B = (V, \bar{A}^0 \cup \cdots \cup \bar{A}^m)$. We construct B as follows: First initialize the sets of loaded arcs $\bar{A}^0 := \bar{A}^1 := \ldots =: \bar{A}^m = \emptyset$. Consider in turn every vertex $v \in V$, the subtree T_v and the predecessor (father) vertex $p(v)$ of v. For every $i \in O$, compute $\Delta_v^i = |\{w \in V_v : a_w = i\}| - |\{w \in V_v : b_w = i\}|$. The quantity Δ_v^i represents the net supply of object i in T_v. If Δ_v^i is positive, then Δ_v^i copies of the loaded arc $(v, p(v))^i$ are added to $\bar{A}^i$. If Δ_v^i is negative, then $|\Delta_v^i|$ copies of loaded arc $(p(v), v)^i$ are added to $\bar{A}^i$. $|\Delta_v^i|$, for the case $\Delta_v^i > 0$ (resp. $\Delta_v^i < 0$), represents the minimum number of times the loaded arc $(v, p(v))^i$ (resp. $(p(v), v)^i$) will be traversed in an optimal SP solution. This construction process ensures the existence of a service path between the origin of every object and at least one of its destinations. The total length of the loaded arcs of B is denoted by $z(B)$, and thus $z(B) \leq z^*$. Constructing B can be achieved in $O(n)$ time, starting from the leaves of T and gradually moving toward the root. The basic graph is directed and the in-degree of each vertex is equal to its out-degree, as for every subtree T_v we must have $\sum_{i \in O} \Delta_v^i = 0$. The loaded arcs of the basic graph B form a union of service paths, where each service path of object $i \in O$ starts at a vertex which is the origin of i and ends at a vertex that demands i. The graph B is not necessarily connected, but when it is connected, it is also strongly connected as the in-degree of each vertex is equal to its out-degree. If the basic graph B is not connected then it can be partitioned into strongly connected components. We note that strong connectivity is determined by the arcs without considering the load of the arcs. Therefore, even if B is strongly connected, it may not be possible to obtain a feasible SP solution by using its loaded arcs. Consider for example the tree depicted in Figure 1, where the label of v is (a_v, b_v), and the corresponding basic graph where the label of each loaded arc is the object type carried on this arc. No SP solution using the loaded arcs of B exists given that the vehicle must start and end its trip empty at vertex 1.

A "feasible path" in the basic graph B consists of a sequence $(u_\ell, u_{\ell+1})^{i_\ell} \in \bar{A}$, $\ell = 1, \ldots L$, $i_\ell \in O$ of loaded arcs of B, which a unit capacity vehicle can follow assuming that the vehicle starts empty at the first vertex u_1. This means that the necessary object on each loaded arc is available when the vehicle reaches the tail of the loaded arc.

We also note that a vertex v with $a_v = b_v \in O \backslash \{0\}$ cannot be replaced by a transshipment vertex, as the supply at vertex v is not necessarily used for covering the demand of the vertex. To see this, consider the example in Figure 1 where $(a_1, b_1) = (1, 1)$. This change does not affect the basic graph B, but a feasible SP solution now exists. However, without loss of generality, the leaves of the tree can be assumed to supply a different object type from the one they demand.

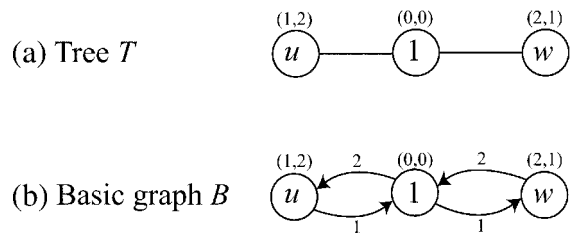


FIG. 1. Instance for which B is connected but does not contain a feasible SP solution.

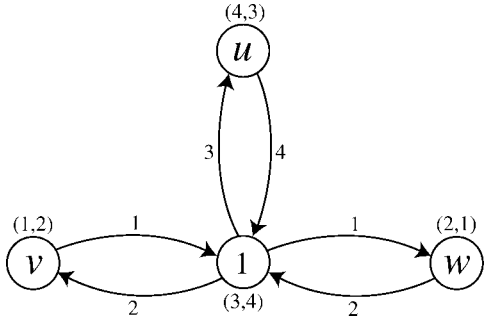


FIG. 2. Strongly connected graph in which not all vertices are mutually reachable.

The example of Figure 1 demonstrates that a stronger property than strong connectivity is required to guarantee a feasible SP solution. To this end we recall the property of "reachability" introduced in [1]. Vertex w is said to be reachable from vertex u if there exists a feasible path from u to w in the basic graph B, taking into account the object type carried on each arc. For example, in Figure 1, vertices 1 and w are reachable from u, vertices 1 and u are reachable from w but neither u nor w is reachable from 1. Indeed, starting from vertex 1, it is impossible to supply vertices u and w without traversing an arc more than once. Thus reachability is not symmetric but is transitive. Two vertices are "mutually reachable" if they are reachable from one another. It may sometimes be necessary to drop an object at an intermediate point along a path to allow reachability. For example, in the graph depicted in Figure 2, vertex u is reachable from vertex v assuming object 1 can be dropped at vertex 1. However, v is not reachable from u as object 2 cannot be made available at vertex 1.

As mutual reachability is symmetric and transitive and each vertex is self-reachable, the mutual reachability relation induces a partition of each of the strongly connected components of V into equivalence classes called "fully connected components." The vertices of the fully connected components are not necessarily contiguous. For example, in the SP depicted in Figure 1, there are two fully connected components: the first one consists of vertices u and w, which are not contiguous vertices, and the second one consists of vertex 1. However, if instead of being a transshipment point vertex 1 had $a_1 = b_1 = 1$, then the basic graph B would consist of a single fully connected component. We say that a basic graph is "fully reachable" if it consists of a single fully connected component, that is, all vertices are reachable from all vertices. The loaded arcs of a fully connected component form a cycle which is a feasible path. Each vertex on the cycle that can be an initial and terminal vertex for the path is in

the fully connected component. If there exist vertices on the cycle that cannot serve as an initial and terminal vertex for this cycle, then they belong to other fully connected components. In other words, the loaded arcs of a fully connected component form a directed cycle that visits all the vertices of the component. Focusing on the vertices of the component, each vertex on the cycle supplies the null object or the object that is needed by the next vertex on the cycle. It follows that any fully connected component of B is balanced, that is, in each component the total demand for object $i \in O$ equals the total supply of object i. If there exists a vertex on the cycle that does not belong to this component, then it belongs to another fully connected component which is reachable (but not mutually reachable) by the former component. The following lemma follows from the above discussion.

Lemma 1. *Any fully connected component of B is balanced.*

This lemma implies that a fully connected component which is a singleton, must necessarily consist of a vertex v with $a_v = b_v$. However, not every vertex v with $a_v = b_v$ defines a singleton in B.

Theorem 1. *An SP solution exists on the basic graph B if and only if this graph is fully reachable.*

Proof. If B is not fully reachable, then no SP solution exists since either the depot cannot be reached from at least one vertex, or at least one vertex cannot be reached from the depot. If B is fully reachable, then it is Eulerian because it is strongly connected and the in-degree of each vertex is equal to its out-degree. An SP solution can therefore be identified by suitably modifying [14] "end-pairing" algorithm for the "Chinese Postman Problem" on an undirected Eulerian graph.

Modified End-Pairing Algorithm

Step 1. Starting at an arbitrary vertex $v \in V$, follow a service path of object of type a_v in the basic graph B emanating from v, until the object reaches its first destination at u. Mark all the loaded arcs of B along the service path. Iteratively apply this process starting from u until v is eventually reached by using only unmarked loaded arcs of B, thus defining a first circuit. Go to Step 3.

Step 2. Construct a second circuit of unmarked arcs of B starting from a vertex w of the first circuit whose supply has not yet been delivered (such a vertex necessarily exists since B is fully reachable and not all vertices have yet been served). Mark all the loaded arcs of the second circuit. Merge the two circuits $(v, \ldots, w_1, w, w_2, \ldots, v)$ and $(w, w_3, \ldots, w_4, w)$ into a single circuit $(v, \ldots, w_1, w, w_3, \ldots, w_4, w, w_2, \ldots, v)$, which now becomes the first circuit. When reaching w for the first time on the merged circuit, drop the object loaded on the vehicle and replace it by the object of type a_w. When reaching w for the second time the object dropped at w is loaded on the empty vehicle.

Step 3. If the circuit contains all the loaded arcs of B, or equivalently, all arcs of B are marked, stop. Otherwise, go to Step 2.

The construction process of B ensures that this graph contains a necessary and sufficient set of arcs to transport all objects, and the end-pairing algorithm together with the marking procedure ensure that all arcs of B are traversed exactly once. ∎

If B is not fully reachable, then it must be augmented into a fully reachable graph by adding new loaded arcs to it. As will be seen in Section 3, the problem of determining a least cost augmentation of B into a fully reachable graph is NP-hard.

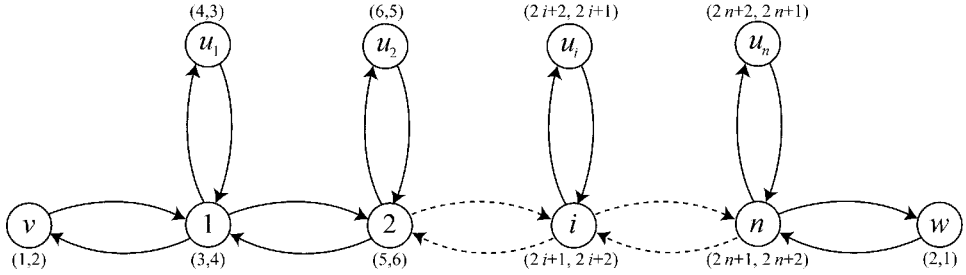
Finally, a natural question is the extent to which preemption can shorten the solution relative to the non-preemptive case.

Theorem 2. *Preemption can shorten the solution by at most 50%, and this bound is tight.*

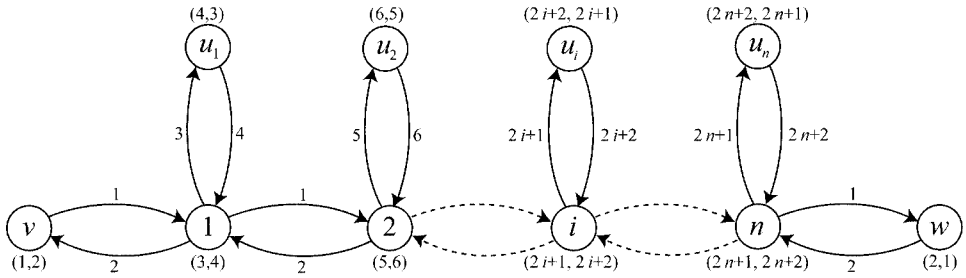
Proof. We first show that the ratio between the optimal non-preemptive solution and the optimal preemptive solution can never be larger than 2. Consider an optimal preemptive solution on a tree T. Let $\tilde{T}$ be a subtree of T spanning the depot and the vertices that were used for preemption by the preemptive solution. The additional cost involved in transforming the preemptive solution into a non-preemptive one is the cost of enabling accessibility to each of the vertices that were used for preemption while the vehicle is free. The transformation cost is bounded above by the cost of adding two arcs of the null object (one in each direction) along each of the edges of $\tilde{T}$. Because the preemptive solution covers each edge of $\tilde{T}$ at least twice (once in each direction), we conclude that the transformation increases the cost by at most 100%.

To show that this bound is tight, consider the example depicted in Figure 3 and representing a comb. We assume that the vertices on the base of the comb are equidistant, that is, $c(v, 1) = c(w, n) = c(i, i + 1) = c$ for $i = 1, \ldots, n - 1$.

A preemptive solution on the arcs of the basic graph B exists. One such solution is obtained when the vehicle loads object 1 at the depot, drops it at vertex 1 to swap objects 3 and 4, reloads object 1 at vertex 1 to ship it to vertex 2, drops it there to swap objects 5 and 6, etc. until it swaps objects $2n + 1$ and $2n + 2$ between vertex n and vertex u_n, and then it reloads object 1 at vertex n, and carries it to vertex w. There it loads object 2 and ships it back to the depot. Next we propose an optimal non-preemptive solution. In addition to the arcs of the basic graph, an empty trip from the depot to vertex n, is added, along which the vehicle stops at each vertex i for $i = 1, \ldots, n$ to swap the objects between vertex i and vertex u_i. After swapping all these objects, the vehicle returns empty to the depot to swap objects 1 and 2 between vertices v and w. Thus, we add to B the arcs $(v, 1)^0$, $(1, v)^0$ and for



a) Tree T



b) Basic graph B

FIG. 3. Example showing that the preemption bound is tight.

$i = 1, \ldots, n-1$ the arcs $(i, i+1)^0$, and $(i+1, i)^0$. Assuming that the distance between vertex i and vertex u_i is $1/c$, and that $c = n$, the ratio between the optimal non-preemptive solution and the optimal preemptive solution can be made as large as $\lim_{n\to\infty} = [(4n+2)n+2]/[(2n+2)n+2] = 2$. ∎

See [3] for more information about the advantage of using preemption in general directed or undirected graphs under various restrictions.

3. THE PREEMPTIVE SWAPPING PROBLEM ON A TREE IS NP-HARD

We now state our main theorem.

Theorem 3. *The preemptive SP on a tree is NP-hard.*

Proof. The proof is rather long and technical and is therefore presented in Appendix 1. Essentially, we show that the problem is at least as hard as the Steiner Tree Problem (STP) on a bipartite graph, which is defined as follows (see [13], pages 208–209). Consider a bipartite graph $\tilde{G} = (\tilde{V}, \tilde{E})$ with bipartition $\{R, S\}$ of $\tilde{V}$, an integer weight $\tilde{c}_e$ for each edge $e \in \tilde{E}$, and a positive integer number β. The problem is to determine whether there exists a subtree of $\tilde{G}$ that spans at least the vertices of R such that the total weight of the edges in the subtree does not exceed β. Frederickson and Guan [11] use a similar version of the STP where all weights are equal to prove that the non-preemptive SCP on a tree is NP-hard. ∎

4. A 1.5-APPROXIMATION ALGORITHM FOR THE PREEMPTIVE SP ON A TREE

In this section, we develop a 1.5-approximation algorithm for the preemptive SP on a tree. We proceed in two steps. We first compute a lower bound on the cost of an optimal solution and we then propose an augmentation procedure that guarantees a feasible solution whose cost never exceeds 1.5 times the optimal cost.

A lower bound on the cost of an optimal solution to the preemptive SP on a tree T can easily be found. Consider the basic graph B associated with the problem. This graph is balanced but not necessarily connected. In the first step, we augment B by identifying the edges of T not covered by any loaded arc of B. For each such edge we add to B two opposite loaded arcs, each associated with the null object. Let B' be the resulting directed multitype, multigraph which is clearly balanced and connected (Fig. 4). As explained in Section 2, this augmentation does not guarantee the existence of a feasible SP solution on B', but $z(B') \leq z^*$. In Fig. 4, B' has two fully connected components $\{1, 2, 3\}$, and $\{4, 5\}$. We will show below that graph B' can be further augmented to obtain a feasible SP solution by adding loaded arcs of the null object, so that the cost of the heuristic solution is at most $1.5z^*$.

Graph B' consists of a number of fully connected components $C_1, \ldots, C_r$. Recall that the depot is located at the root of T, which we assume to be in component C_1. If $r = 1$, then by Theorem 1 the loaded arcs of B' induce a feasible and optimal SP solution. A fully connected component of B' is said to be "unreachable" if none of its vertices is reachable from a

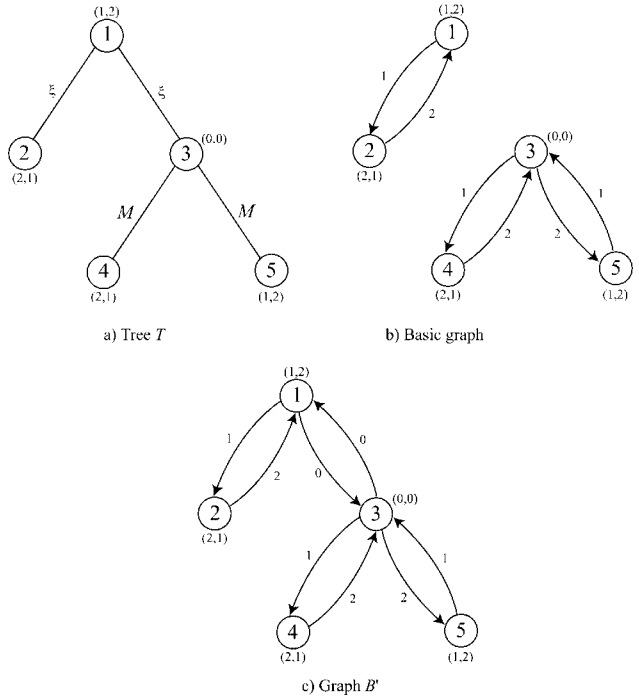


FIG. 4. Example showing that the worst-case bound of the heuristic is tight.

vertex in any other fully connected component by following the service paths induced by B''s loaded arcs. Component C_1, where the depot is located, is considered to be reachable (from the depot). In Figure 4 component $\{4, 5\}$ is unreachable. In the next lemma, we prove that if $r \geq 2$, then there must exist at least one fully connected component of B' that is unreachable.

Lemma 2. *If B' contains at least two fully connected components then at least one of the components is unreachable.*

Proof. Provided in Appendix 2. ∎

The algorithm we propose adds pairs of opposite direction loaded arcs of the null object until it forms a partition of V into a fully connected component which contains the depot, and possibly some singleton fully connected components. More specifically, as long as the multigraph on hand is partitioned into more than one fully connected component which are not singletons, we start a new iteration (a singleton can serve itself). At each iteration we identify the set of all unreachable vertices, which belong to nonsingleton fully connected components in the multigraph at hand, for which all their predecessors (in T) are reachable. Among those unreachable vertices, we find one which is closest to its predecessor in T in terms of the cost function $c : E \to \Re^+$. We update the multigraph at hand by adding to it two opposite-direction loaded arcs of the null object connecting this vertex to its predecessor vertex (in T), and we restart a new iteration. This procedure obviously preserves the balance of the graph and is finite. We present now the "Augmentation Algorithm" for the SP on $T = (V, E)$. The algorithm returns a multidirected graph H in which each loaded arc is associated with one of the objects in O.

Augmentation Algorithm

Step 1. Input: $T = (V, E)$, the cost c_e for each $e \in E$ and the associated graph B'. Output: multitype, directed multigraph H, and its cost $z(H)$. Set $\tilde{B} \leftarrow B'$.

Step 2. Find the partition of $\tilde{B}$ into fully connected components $\{C_1, \ldots, C_r, C_{r+1}, \ldots, C_{r'}\}$, where $C_{r+1}, \ldots, C_{r'}$ are singletons. If $r = 1$, return $H \leftarrow \tilde{B}$ and $z(H)$; stop. Otherwise, let $C'_1, \ldots, C'_K$, $K \geq 1$, be the unreachable components of $\tilde{B}$ that are not singletons.

Step 3. For $\ell = 1, \ldots, K$ define $S_\ell \subset V$ to be the set of vertices of the unreachable component C'_ℓ. Set $S \leftarrow \cup_{\ell=1}^{K} S_\ell$ and $\tilde{S} \leftarrow \emptyset$. For each $v \in S$ do begin: if the predecessor of v in T is not a vertex in S, then set $\tilde{S} \leftarrow \tilde{S} \cup \{v\}$; end.

Step 4. For each vertex $v \in \tilde{S}$ do begin: set $\kappa_v \leftarrow c(p(v), v)$; let $v^* \in \text{argmin}\{\kappa_v : v \in \tilde{S}\}$.

Step 5. Augment the graph $\tilde{B}$ by adding to it the loaded arcs $(v^*, p(v^*))^0$ and $(p(v^*), v^*)^0$. Go to Step 2.

We first prove that the resulting graph H induces a feasible SP solution.

Theorem 4. *There exists a feasible SP solution on H.*

Proof. The graph H is connected and balanced as graph B' had this property and during the augmentation the balance is preserved. In the graph H there are no unreachable components, which are not singletons. A feasible SP solution on graph H can be obtained by the modification of Hierholzer's end-pairing algorithm for the Chinese Postman Problem used in the proof of Theorem 1. ∎

To analyze the worst-case performance ratio of the proposed heuristic we need a further characterization of the intermediate graphs $\tilde{B}$ obtained during the application of the algorithm.

Lemma 3. *Consider a tree $T = (V, E)$ and a corresponding balanced and connected graph $\tilde{B}$ obtained in the application of the Augmentation Algorithm. Suppose that C' is an unreachable component of $\tilde{B}$. Consider a vertex $w \in C'$ and the subtree $T_w = (V_w, E_w)$. If in the subtree $T_w = (V_w, E_w)$, the set of vertices V_w contains a vertex u belonging to an unreachable fully connected component of $\tilde{B}$, say $\tilde{C}$, $\tilde{C} \neq C'$, then $\tilde{C} \subset V_w$.*

Proof. Suppose that $\tilde{C}$ is not a subset of V_w. Then, there exists a vertex $u_1 \in \tilde{C} \cap V_w$, but $\tilde{C}$ has at least one vertex u_2 that is not a member of V_w. By definition of a fully connected component, there must exists in $\tilde{B}$ a service path that starts at u_1 and ends at vertex u_2. By the property of a tree, this

unique path must pass through vertex w. This service path makes vertex w reachable from $\tilde{C}$, violating the assumption that w belongs to an unreachable fully connected component of $\tilde{B}$. ∎

Consider any unreachable component of $\tilde{B}$. In Lemma 4, we prove that the root of the minimal subtree containing this component is not a member of the component.

Lemma 4. *Consider an unreachable component C' of graph $\tilde{B}$ obtained in the Augmentation Algorithm. Let $T_w = (V_w, E_w)$ be the minimal subtree for which $C' \subseteq V_w$. Then $w \notin C'$.*

Proof. Suppose by contradiction that $w \in C'$. The fact that $T_w = (V_w, E_w)$ is the minimal subtree for which $C' \subseteq V_w$ implies that $p(w) \notin C'$. By Lemma 3 all nonreachable components of $\tilde{B}$ having a vertex in V_w are contained in V_w. Suppose that $u \in V_w$, and u belongs to a reachable component of $\tilde{B}$, which we denote by C^*. If $C^* \not\subset V_w$ then there exists a service path in $\tilde{B}$ connecting vertex u to a vertex $v \notin V_w$ through vertex w, contradicting the fact that C' is a nonreachable component. Thus, $C^* \subset V_w$. Therefore, V_w is equal to the union of some fully connected components, which means that the basic graph B is disconnected along the edge $(p(w), w)$ of T. This implies that in the augmentation of B to B' two opposite loaded arcs of the null object connecting the vertices w and $p(w)$ are added. Thus, in graph $\tilde{B}$, the vertices w and $p(w)$ belong to the same fully connected component, contradicting our assumption that $C' \subset V_w$. ∎

In Theorem 5, we prove that the worst-case performance ratio of the heuristic is 1.5.

Theorem 5. *$z(H) \leq 1.5z^*$, and this worst-case bound is tight for our algorithm.*

Proof. The proof is provided in Appendix 3. ∎

5. THE CASE $m = 1$ IS POLYNOMIAL

In this section, we consider the special case where the set of items consists of a single type object and the null object, that is, $O = \{0, 1\}$. We show that there exists an optimal routing of the vehicle that only uses the loaded arcs of the graph B'. We note that for $m = 1$, the preemptive and non-preemptive cases coincide. But as the non-preemptive SP on a tree has never been studied on its own (the NP-hardness of the problem follows from the proof of [11] on the NP-hardness of the non-preemptive SCP on a tree), this result is new. The vertices of the tree are of four types: a vertex is either (i) a supply vertex associated with the pair $(a, b) = (1, 0)$; or (ii) a demand vertex associated with the pair $(a, b) = (0, 1)$; or (iii) a transshipment vertex associated with the pair $(a, b) = (0, 0)$; or (iv) a vertex associated with the pair $(a, b) = (1, 1)$. Clearly, in the case of one object type there is no need to preempt. Thus, there exists an optimal solution in which a vertex v which is associated with $(a_v, b_v) = (1, 1)$ will always serve itself by using its own supply.

Recall from Section 2 that "all" the loaded arcs in the basic graph B, which are directed from vertex u to vertex v, are associated with a an object $i \in \{0, 1\}$, where all the loaded arcs directed from vertex v to vertex u are associated with the other object $1 - i$. As the basic graph B is balanced, the two sets of loaded arcs have the same cardinality. If B is not connected, the graph is augmented by adding two opposite direction loaded arcs of the null object between any two adjacent vertices of T which are not covered by any loaded arc of B. This results in graph B'. We show in Theorem 6 that graph B' consists of a single reachable fully connected component implying that $z^* = z(B')$.

Theorem 6. *For the case $m = 1$, graph B' consists of a single fully connected component.*

Proof. We show that none of the fully connected components of graph B' is unreachable. Suppose by contradiction that there were at least one unreachable fully connected component C. Let the subtree T_w be the minimal subtree that contains C. According to Lemma 4, $w \notin C$. By definition, there must exist a service path of object 1 starting at a vertex $v_1 \in C$, passing through vertex w, and ending at a vertex $v_2 \in C$. Since B is balanced, there must also exist a service path of the null object starting at v_2, passing through w and ending at v_1. This makes v_1 reachable from w, contradicting our assumption that C is unreachable. ∎

APPENDIX 1: PROOF OF THEOREM 3

To prove Theorem 3, we start by describing a general bipartite graph. We show then that the NP-hard problem of finding a minimum cost Steiner tree that spans the vertices on one side of the graph can be reduced in polynomial time to the preemptive SP on a tree, proving that the later is at least as hard as the former. As the proof is long and the reduction is non-trivial, we break the proof into several parts in order to make it simpler for the readers.

Part 1. The Steiner Tree Problem in Bipartite Graphs

Given a graph $\tilde{G} = (\tilde{V}, \tilde{E})$ with a bipartition $\{R, S\}$ of $\tilde{V}$. Let $|\tilde{V}| = \vartheta$, $|\tilde{E}| = \varepsilon$, and $|R| = \rho$. We number the vertices of R by $1, \ldots, \rho$, and the vertices of S by $\rho+1, \ldots, \vartheta$. The length of edge $[v, u] \in \tilde{E}$ is denoted by $\tilde{c}(v, u)$. We also let Γ_v be the set of vertices adjacent to vertex v, i.e., the neighborhood of vertex v, and d_v the degree of vertex v, thus $d_v = |\Gamma_v|$, $\Sigma_{v=1}^{\rho} d_v = \varepsilon$ and $\Sigma_{u=\rho+1}^{\vartheta} d_u = \varepsilon$. For any vertex $v \in R$, let $\varphi(v, i)$ be the i^{th} closest vertex to v among the d_v vertices in Γ_v (where ties are broken arbitrarily). Thus, for a given vertex $v \in R$, $\varphi(v, \cdot)$ is a one-to-one mapping of $\{1, \ldots, d_v\}$ into Γ_v such that $\tilde{c}(v, \varphi(v, 1)) \leq \tilde{c}(v, \varphi(v, 2)) \leq \ldots \leq \tilde{c}(v, \varphi(v, d_v))$. The objective in this problem is to determine a minimum Steiner tree on $\tilde{G} = (\tilde{V}, \tilde{E})$ spanning the vertices of R.

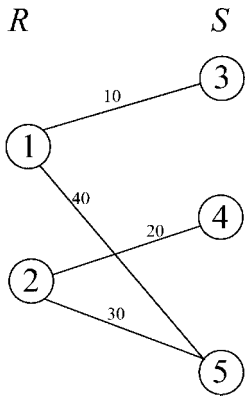


FIG. 5. Graph $\tilde{G} = (\tilde{V}, \tilde{E})$ for the Steiner tree problem on a bipartite graph.

Example: Figure 5 depicts a graph $\tilde{G} = (\tilde{V}, \tilde{E})$ with $\rho = 2$, $\vartheta = 5$, $\varepsilon = 4$, $\tilde{c}(1,3) = 10$, $\tilde{c}(1,5) = 40$, $\tilde{c}(2,4) = 20$, $\tilde{c}(2,5) = 30$, $\varphi(1,1) = 3$, $\varphi(1,2) = 5$, $\varphi(2,1) = 4$, $\varphi(2,2) = 5$, $\Gamma_1 = \{3,5\}$, $\Gamma_2 = \{4,5\}$, $d_1 = 2$, $d_2 = 2$, $\Gamma_3 = \{1\}$, $\Gamma_4 = \{2\}$, $\Gamma_5 = \{1,2\}$, $d_3 = 1$, $d_4 = 1$, $d_5 = 2$.

We next show that if there exists a polynomial algorithm for solving the preemptive SP on a tree then we could have used it to solve the above described STP on a bipartite graph in a polynomial time, contradicting the fact that the later problem is NP-hard. To this end we construct the following reduction.

Part 2. The SP Associated With the Given Steiner Tree Problem

Given a bipartite graph $\tilde{G} = (\tilde{V}, \tilde{E})$, we construct an SP on tree $T = (U, E^*)$ with $|U| = 3\varepsilon + 2\rho + 1$ vertices, and $m = \vartheta - \rho + \varepsilon$ object types, in addition to the null object. The objects' names in the SP are determined as follows: consider graph $\tilde{G} = (\tilde{V}, \tilde{E})$ and the bipartition of $\tilde{V} = R \cup S$. For each vertex $u \in S$ we define $1 + d_u$ different object types, named u^* and u_v for $v \in \Gamma_u$. We let also the function $c^T : E^* \to \Re^+$ represent the length of the edges of T. We next describe the construction of T. To simplify the presentation, we break the description of T into levels, from the root down to its leaves.

Level 1. The tree T is rooted at vertex $\ell(0,0)$ (which does not serve as the depot) and is connected to ρ children named $\ell(1,0), \ldots, \ell(\rho,0)$. The root and its children are transshipment points, that is, $a_{\ell(v,0)} = b_{\ell(v,0)} = 0$ for $v \in \{0, \ldots, \rho\}$. Let the length of the edges connecting the root of T to its children be defined as: $c^T(\ell(0,0), \ell(v,0)) = \tilde{c}(v, \varphi(v,1))$ for $v \in \{1, \ldots, \rho\}$, that is, the distance between the root of T, namely vertex $\ell(0,0)$, and its child $\ell(v,0)$, is equal to the minimum distance in graph $\tilde{G}$ between v and a vertex in Γ_v. Each of the ρ children of the root serves as the root of a subtree $T_{\ell(v,0)}$ for $v \in \{1, \ldots, \rho\}$, described below. If $d_v = 1$ go to Level Intermediate. Otherwise, go to Level 2.

Level 2. We next describe the construction of the second level of T, which is the first level of $T_{\ell(v,0)}$ for $v \in \{1, \ldots, \rho\}$. Vertex $\ell(v,0)$ is a parent of two children, the right-hand side child $r(v,1)$ and the left-hand side child $\ell(v,1)$. The right-hand side child of $\ell(v,0)$, namely vertex $r(v,1)$, is a leaf associated with objects $(a,b) = (\varphi(v,1)^*, \varphi(v,1)_v)$. We let $c^T(\ell(v,0), r(v,1)) = M$ for some $M \in \Re^+$ to be specified later. The left-hand side child of $\ell(v,0)$, namely $\ell(v,1)$, is a transshipment vertex, and we let $c^T(\ell(v,0), \ell(v,1)) = \tilde{c}(v, \varphi(v,2)) - \tilde{c}(v, \varphi(v,1))$, which is non-negative by definition of the function $\varphi(v,\cdot)$. If $d_v = 2$ go to to Level Intermediate. Otherwise, go to Level 3.

Level 3. Similarly to Level 2, we add a new level to the tree T: We let $\ell(v,1)$ to be a parent of the two children $r(v,2)$ and $\ell(v,2)$. The right-hand side child of $\ell(v,1)$, namely vertex $r(v,2)$, is a leaf associated with objects $(a,b) = (\varphi(v,2)^*, \varphi(v,2)_v)$. We let $c^T(\ell(v,1), r(v,2)) = M$. The left-hand side child of $\ell(v,1)$, namely vertex $\ell(v,2)$, is a transshipment vertex, and we let $c^T(\ell(v,1), \ell(v,2)) = \tilde{c}(v, \varphi(v,3)) - \tilde{c}(v, \varphi(v,2))$, which is again non-negative.

This construction process continues up to but not inclusive Level d_v before reaching Level Intermediate.

Level Intermediate. By construction, the total distance on T between the root of the tree, namely vertex $\ell(0,0)$ and vertex $\ell(v, d_v - 1)$, is $\tilde{c}\ (v, \varphi(v, d_v))$, which is the farthest distance in graph $\tilde{G}$ between vertex v and one of the vertices adjacent to it. We let $\ell(v, d_v - 1)$ to be a parent of the two children $r(v, d_v)$ and $\ell(v, d_v)$. The right-hand side child of $\ell(v, d_v - 1)$, namely vertex $r(v, d_v)$, is a leaf associated with objects $(a,b) = (\varphi(v, d_v)^*, \varphi(v, d_v)_v)$, and we let $c^T(\ell(v, d_v - 1), r(v, d_v)) = M$. The left-hand side child of $\ell(v, d_v - 1)$, namely vertex $\ell(v, d_v)$, is associated with objects $(a,b) = (\varphi(v,1)_v, \varphi(v,1)^*)$, and we let $c^T(\ell(v, d_v - 1), \ell(v, d_v)) = M$. Go to Levels Bottom Subtree.

Levels Bottom Subtree. Vertex $\ell(v, d_v)$ has two children. The right-hand side child $r(v, d_v + 1)$ is a leaf associated with objects $(a,b) = (\varphi(v,1)^*, \varphi(v,1)_v)$. The left-hand side child of $\ell(v, d_v)$, namely vertex $\ell(v, d_v + 1)$, is the root of a subtree, which is a path consisting of exactly d_v vertices named $\ell(v, d_v + 1), \ldots, \ell(v, 2d_v)$. The last vertex on the path, that is, vertex $\ell(v, 2d_v)$, is associated with objects $(a,b) = (\varphi(v,1)_v, \varphi(v,1)^*)$. For $i = 1, \ldots, d_v - 1$, vertex $\ell(v, d_v + i)$ is associated with objects $(a,b) = (\varphi(v, i+1)_v, \varphi(v, i+1)^*)$. The length of all edges of the subtree, which is rooted at vertex $\ell(v, d_v)$, can be assumed to be 0.

Example (contd.): Figure 6 depicts the tree $T = (U, E^*)$, corresponding to the example of Figure 5, with $m = \vartheta - \rho + \varepsilon = 5 - 2 + 4 = 7$, $|U| = 3\varepsilon + 2\rho + 1 = 17$, and the object set $O = \{0, 3^*, 3_1, 4^*, 4_2, 5^*, 5_1, 5_2\}$.

Properties of T

We note that by this construction each subtree $T_{\ell(v,0)}$ consists of $3d_v + 2$ vertices, where exactly d_v of them are

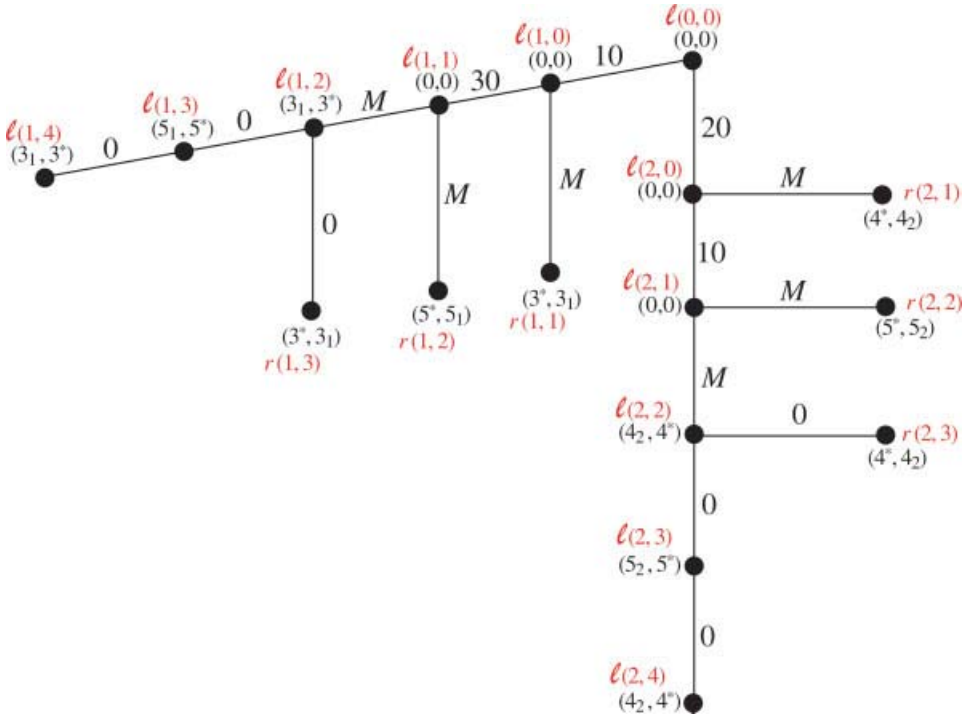


FIG. 6. Swapping problem on a tree $T = (U, E^*)$ corresponding to the Steiner tree problem of Figure 5. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

transshipment points. The subtrees are balanced because the total supply of each object type is equal its total demand. More precisely, each of the objects $\varphi(v,i)^*$ and $\varphi(v,i)_v$ for $i = 2,\ldots,d_v$ is the supply and the demand of exactly one vertex in the subtree. Each of the objects $\varphi(v,1)^*$ and $\varphi(v,1)_v$ is the supply and demand of two vertices in the subtree. We also note that object type $\varphi(v,i)_v$ for $i = 1,\ldots,d_v$ are not used by any subtree $T_{\ell(v',0)}$, $v' = 1,\ldots,\rho$ other than $T_{\ell(v,0)}$. On the other hand, objects $\varphi(v,i)^*$ for $i = 1,\ldots,d_v$ are used by all subtrees $T_{\ell(v',0)}$ with $\varphi(v,i)^* \in \Gamma_{v'}$ and $v' = 1,\ldots,\rho$. In other words, object u^* for $u \in S$ is used by all subtrees $T_{\ell(v',0)}$ for which $v' \in \Gamma_u$.

The Basic Graph Associated With T. We denote the basic graph associated with T by $B(T)$. Note that it consists of exactly ρ fully connected components which are not singletons. Each fully connected component C_v for $v = 1,\ldots,\rho$ consists of all vertices of the subtree $T_{\ell(v,0)}$, which are not transshipment points, namely vertices $r(v,i)$ for $i = 1,\ldots,d_v+1$, and vertices $\ell(v,d_v+i)$ for $i = 0,\ldots,d_v$. Each of the transshipment vertices of $T_{\ell(v,0)}$ serves as a fully connected component which is a singleton. In particular, each singleton in $T_{\ell(v,0)}$ is reachable from the component C_v. The graph $B(T)$ is disconnected as the root of T is not incident to any loaded arc of $B(T)$, that is, vertex $\ell(0,0)$ is not reachable from any of the other fully connected components. Thus graph $B(T)$ must be augmented by adding two opposite direction loaded arcs of the form $(\ell(0,0),\ell(v,0))^{i_v}$ and $(\ell(v,0),\ell(0,0))^{i_v}$ for $v = 1,\ldots,\rho$ and $i_v \in O$. The total cost of these new loaded arcs is $2\Sigma_{v=1}^{\rho}\tilde{c}(v,\varphi(v,1))$. Independently of the location of the depot, any feasible solution for the SP on T must be of a length at least as large as $z(B(T)) + 2\Sigma_{v=1}^{\rho}\tilde{c}(v,\varphi(v,1))$.

Large Loaded Arcs. The loaded arcs of $B(T)$ that were assigned a length M are called "large loaded arcs" in the sequel; they play a key role in the remainder of this proof. We first note that the number of these arcs is $\Sigma_{v=1}^{\rho}4d_v = 4\varepsilon$, as each edge of T of the form $[\ell(v,i-1),r(v,i)]$ for $i = 1,\ldots,d_v$ is associated with two loaded arcs in $B(T)$, and the edge $[\ell(v,d_v-1),\ell(v,d_v)]$ of T is covered by $2d_v$ loaded arcs in $B(T)$. We choose M to be a large number, say $M = z(\tilde{G}) = \sum_{(v,u)\in\tilde{E}}\tilde{c}(v,u)$, to guarantee that the optimal SP solution includes the smallest possible number of copies of large loaded arcs. In $B(T)$ all the large loaded arcs are associated with a real object in $O\backslash\{0\}$. Considering Figure 6, we can see that if the vehicle while empty enters vertex $\ell(v,0)$ for $v = 1,\ldots,\rho$, for the first time, coming from $\ell(0,0)$, then to continue servicing component C_v from there, the vehicle must travel along two opposite-direction large loaded arcs of the null object in addition to the loaded arcs of $B(T)$. The first empty such travel along a large loaded arc is needed to first reach a vertex in C_v from which an object can be loaded. The cost of travel along large loaded arcs of the null object can be avoided only if vertex $\ell(v,0)$ is reached from $\ell(0,0)$ while the vehicle is already loaded by an object that is needed in C_v. As will be shown below, if there exists a Steiner tree on the bipartite graph $\tilde{G} = (\tilde{V},\tilde{E})$ spanning the vertices of R, then any optimal SP solution for T includes only the large loaded arcs that are members of $B(T)$, that is, exactly 4ε large loaded

arcs, which are all travelled while the vehicle is loaded by a real object, meaning that objects in one component are used to cover the demand in other components.

Locating the Depot. To complete the definition of the SP on T, we assume that the depot is located at vertex $r(1,1)$, which, by definition, is not a transshipment vertex (any other choice of the depot's location at a non-transshipment vertex of T would do as well). Any feasible solution to the preemptive SP on T consists of $B(T)$'s loaded arcs, and augmenting loaded arcs that connect the fully connected components of the basic graph $B(T)$ into a single such component.

Solutions with Autonomous Components. In a solution in which a fully connected component C_v, $v \neq 1$, is autonomous, that is, it satisfies its own demand, the augmentation of $B(T)$ must include two-opposite direction paths of the null object, connecting the root, namely $\ell(0,0)$, to the closest vertex in C_v, namely vertex $r(v,1)$, a path whose length is $\tilde{c}(v,\varphi(v,1)) + M$. Thus, the total cost of a solution in which all components are autonomous is $z(B(T)) + 2\Sigma_{v=1}^{\rho}\tilde{c}(v,\varphi(v,1)) + 2(\rho-1)M$. We show below that if there exists a Steiner Tree for $\tilde{G}$, then exploiting the possibility of using objects of type u^*, $u \in \{\rho+1,\ldots,\vartheta\}$, supplied in one fully connected component of T to cover the demand of another component, generates shorter SP solutions, in which the vehicle does not make unnecessary empty trips along large arcs.

Constructing Cheaper Solutions. Consider now a certain fully connected component C_v for $v \in \{2,\ldots,\rho\}$. The only way to serve C_v without paying empty travels along a large loaded arc of C_v, is by entering C_v while carrying an object u^*, which was loaded at another fully connected component visited earlier, say $C_{v'}$ for $v' \in \{1,\ldots,\rho\}\backslash\{v\}$, and the object u^* is needed by a vertex in C_v. This can be achieved only if $u \in \Gamma_v \cap \Gamma_{v'}$. To this end suppose that $\varphi(v',i) = u$ and $\varphi(v,j) = u$. The transfer of object u^* from component $C_{v'}$ to component C_v is done as follows: suppose that the vehicle loads object u^* at vertex $r(v',i)$, when there is still demand for this object in C_v. The vehicle then follows the loaded arc of $B(T)$, $(r(v',i),\ell(v',i-1))^{u^*}$, and along an augmenting path of object u^*, the vehicle continues from $\ell(v',i-1)$ to the root of the tree, and from there to vertex $\ell(v,j-1)$ in component C_v. From $\ell(v,j-1)$ the vehicle follows the path of loaded arcs of object u^* in $B(T)$, to vertex $\ell(v,d_v+j-1)$ in C_v demanding this object. To balance the shipment of the unit of object u^* to C_v, another unit of object u^* must eventually be carried out from C_v to cover the demand in some other component. This solution necessitates, in addition to the loaded arcs of $B(T)$, two augmenting opposite-direction loaded paths associated with object u^* that connect vertex $\ell(v',i-1)$ to vertex $\ell(v,j-1)$. The cost of these two augmenting paths is $2c^T(\ell(v',i-1),\ell(0,0)) + 2c^T(\ell(0,0),\ell(v,j-1)) = 2\tilde{c}(v',\varphi(v',i)) + 2\tilde{c}(v,\varphi(v,j)) = 2\tilde{c}(v',u) + 2\tilde{c}(v,u)$, that is, the cost of this augmentation boils down to twice the cost of two edges $[v',u]$ and $[v,u]$ in the bipartite graph $\tilde{G}$. Comparing the cost of serving C_v by the last proposed solution and the solution in which the vehicle enters C_v while empty, we find that the former is strictly cheaper. (The saving is $2(M - c^T(\ell(v',0),\ell(v',i-1)) - c^T(\ell(v,0),\ell(v,j-1))) = 2(z(\tilde{G}) - (\tilde{c}(v',\varphi(v',i)) - \tilde{c}(v',\varphi(v',1))) - (\tilde{c}(v,\varphi(v,j)) - \tilde{c}(v,\varphi(v,1)))) > 0$.)

Conclusion. If there exists a Steiner tree on the bipartite graph $\tilde{G} = (\tilde{V},\tilde{E})$ spanning the vertices of R, then an optimal SP solution for T does not contain any extra large loaded arcs beyond the 4ε such arcs that are part of $B(T)$. In such a case, all fully connected components C_v, $v \in R$, must be connected to each other by transferring objects of type u^*, $u \in S$ as explained above. Any feasible solution to the SP on T that does not contain more than 4ε large loaded arcs induces a Steiner tree spanning R on the bipartite graph $\tilde{G} = (\tilde{V},\tilde{E})$. The cost of the SP solution is the sum of $z(B(T))$ and twice the cost of the associated Steiner tree on $\tilde{G}(\tilde{V},\tilde{E})$ used to connect the components. Thus, an algorithm that finds the optimal SP solution on T would also find an optimal Steiner tree spanning R in $\tilde{G}$.

Example (contd.): The optimal Steiner tree that spans $R = \{1,2\}$ in the bipartite graph $\tilde{G}$ depicted in Figure 5 consists of edges $(1,5)$ and $(2,5)$ and its length is 70. The associated SP tree $T = (U,E^*)$, depicted in Figure 6, with the depot located in vertex $r(1,1)$, induces a basic graph $B(T)$, which consists of two fully connected components, namely $C_1 = \{\ell(1,2),\ell(1,3),\ell(1,4),r(1,1),r(1,2),r(1,3)\}$ and $C_2 = \{\ell(2,2),\ell(2,3),\ \ell(2,4),\ r(2,1),r(2,2),r(2,3)\}$ (in addition to the components that consist of singletons, which are the transshipment vertices). In the Steiner tree problem, vertex 5 is the only vertex in S which is adjacent to more than one vertex in R. Therefore, the only object that can be used to connect the components C_1 and C_2 of $B(T)$ is object 5^*. The optimal solution for the SP is to load object 3^* at the depot located at vertex $r(1,1)$ in component C_1, carry it to $\ell(1,4)$ to supply its demand, load there object 3_1 and temporarily drop it at $\ell(1,3)$. Load at $\ell(1,3)$ object 5_1 and carry it to $r(1,2)$ to supply its demand. There, at $r(1,2)$, load object 5^* and carry it to the other component, C_2. More specifically, carry it directly to vertex $\ell(2,3)$ in order to supply its demand. Then by following the arcs of the basic graph of T in C_2 we can supply all the demand in that component, and exit C_2 while the vehicle is loaded with object 5^*, which is now shipped to vertex $\ell(1,3)$ in C_1. The vehicle then continues to follow the arcs of $B(T)$ in C_1, which have not yet been used by the solution, to complete the supply of the demand in all the vertices in C_1, and end its trip at the depot. To conclude, the only loaded arcs used by this solution except of the loaded arcs of the basic graph are two copies (one to each direction) of $(\ell(1,1),\ell(1,0))^{5^*}$, $(\ell(1,0),\ell(0,0))^{5^*}$, $(\ell(0,0),\ell(2,0))^{5^*}$, and $(\ell(2,0),\ell(2,1))^{5^*}$. Therefore, the length of the swapping solution is the cost of $B(T)$ plus twice the cost of the optimal Steiner tree of $\tilde{G}$, i.e., it is $z(B(T)) + 2 \times 70$.

APPENDIX 2: PROOF OF LEMMA 2

Suppose by contradiction that none of the $r \geq 2$ fully connected components of B' is unreachable, that is, each component (except possibly for the component that contains the depot) can be reached from some other component. Choose any fully connected component that does not contain the depot. Call it $C_{\alpha(1)}$ and mark it. As $C_{\alpha(1)}$ is reachable, there exists another component $C_{\alpha(2)}$ from which $C_{\alpha(1)}$ can be reached. Clearly $C_{\alpha(2)}$ cannot be reached from $C_{\alpha(1)}$ as otherwise the two components could be merged, and r would be smaller. Mark $C_{\alpha(2)}$. Similarly, $C_{\alpha(2)}$ can be reached from some other component $C_{\alpha(3)}$, $\alpha(3) \neq \alpha(1)$. Mark $C_{\alpha(3)}$. Component $C_{\alpha(3)}$ cannot be reached from neither $C_{\alpha(1)}$ or $C_{\alpha(2)}$, otherwise the two components could be merged, contradicting the partition of B' into r fully connected components. This procedure can be followed for at most r steps. At the end of this part of the procedure, we obtain a sequence of at most r mutually disjoint fully connected components which are all marked, where $C_{\alpha(q-1)}$ is reachable from $C_{\alpha(q)}$ for $q = 2, \ldots, Q$ and $Q \leq r$. Thus, component $C_{\alpha(Q)} = C_1$, i.e., it is the component that contains the depot, as otherwise it would be an unreachable component, contradicting our assumption. If r is greater than the number of marked fully connected components, then this process is repeated starting with a fully connected component $C_{\alpha(Q+1)}$ which is still unmarked. We terminate the process when all the r fully connected components of B' are marked.

At the end of the process, we obtain a directed tree Υ whose r nodes are the fully connected components of B', and whose arcs define the reachability among the components. Assuming that the tree is rooted at $C_1 = C_{\alpha(Q)}$, all the arcs of Υ are directed toward the leaves of Υ, meaning that all the fully connected components are reachable from the depot, but none of them can reach the depot. We will show that this yields a contradiction. Consider the fully connected component $C_{\alpha(Q-1)}$ which is reachable from C_1. The reachability is along an arc of the basic graph B, as otherwise it would be along an arc of the null object added to B while constructing B', and this would make C_1 reachable from $C_{\alpha(Q-1)}$ along the opposite direction arc of the null object in B'. As a result the two fully connected components $C_{\alpha(Q-1)}$ and C_1 would be merged into one, contradicting our assumption about r. This means that the reachability of any fully connected component $C_{\alpha(Q-1)}$, from C_1, is along a loaded arc $(p(v_{Q-1}), v_{Q-1})^{i_1}$ of B, where v_{Q-1} is a vertex in $C_{\alpha(Q-1)}$. As B is balanced, there must also exist an arc $(v_{Q-1}, p(v_{Q-1}))^{i_2}$ in B, such that $i_1 \neq i_2$. If object type i_2 is the supply of at least one vertex in $C_{\alpha(Q-1)}$, or if $i_2 = 0$ then this would make C_1 reachable from $C_{\alpha(Q-1)}$, merging the two components into one, contradicting our assumption about r. Thus, object type i_2 is shipped into $C_{\alpha(Q-1)}$ from some other fully connected component $C_{\alpha(q)}$, where $C_{\alpha(q)} \neq C_1$ and $C_{\alpha(q)} \neq C_{\alpha(Q-1)}$. This means that the two fully connected components $C_{\alpha(q)}$ and $C_{\alpha(Q-1)}$ can be merged, contradicting again our assumption regarding the number r of fully connected components of B'. Therefore, it is impossible that none of the fully connected components of B' is unreachable. ∎

APPENDIX 3: PROOF OF THEOREM 5

While implementing the Augmentation Algorithm, an edge of T is augmented at most once since in future iterations both end vertices of the edge are reachable in $\tilde{B}$ from vertex 1. Let θ be the number of iterations run by the Augmentation Algorithm. To prove that the worst-case performance ratio of the proposed heuristic is 1.5, we will prove the following statements:

1. In each iteration, $j = 1, \ldots, \theta$, the algorithm identifies an edge of T, denoted by $e_j \in E$, that is augmented, and we also identify a set of edges $D_j \subset E$ that are never augmented by the algorithm. In particular, $e_j \notin \cup_{i=1}^{\theta} D_i$.
2. $c_{e_j} \leq z(D_j)$.
3. $e_j \neq e_i$ and $D_j \cap D_i = \emptyset$ for $i \neq j$.
4. $\sum_{i=1}^{\theta} \{2c_{e_i} + 2z(D_i)\} \leq z(B')$.

We now provide the proofs.

1. We will first show that for a given graph $\tilde{B}$ the set $\tilde{S}$ defined by the algorithm contains at least two vertices from each (nonsingleton) unreachable fully connected component C'_ℓ of $\tilde{B}$, for which $\tilde{S} \cap S_\ell \neq \emptyset$. Suppose by contradiction that for the multigraph $\tilde{B}$, the set $\tilde{S}$ contains only one vertex v from S_ℓ. As C'_ℓ is not a singleton, the set S_ℓ contains at least two vertices. According to our assumption, none of the vertices in $S_\ell \backslash \{v\}$ is a vertex in $\tilde{S}$, thus either there exists a vertex in $S_\ell \backslash \{v\}$ which is a descendant of a vertex $w \in \tilde{S} \backslash S_\ell$, or all vertices of $S_\ell \backslash \{v\}$ are descendants of v. We will show that both cases yield a contradiction.

   First suppose that there exists a vertex in $S_\ell \backslash \{v\}$ which is a descendant of a vertex $w \in \tilde{S} \backslash S_\ell$. According to Lemma 3, $S_\ell \subset V_w$, where V_w is the set of vertices of the subtree T_w of T. This is in contradiction with the fact that $v \in S_\ell$ and $v \notin V_w$, as $v, w \in \tilde{S}$. Suppose now that all vertices of $S_\ell \backslash \{v\}$ are descendants of v. Thus, the minimal subtree of T containing C'_ℓ is rooted at v. Recall that C'_ℓ is unreachable, and v is a vertex of C'_ℓ contradicting Lemma 4.

   Let vertex v^* be selected according to Step 4 of the Augmentation Algorithm when applied on $\tilde{B}$ in iteration j. Suppose that $v^* \in C'_\ell$, where C'_ℓ is an unreachable fully connected component of $\tilde{B}$ (which is not a singleton). Thus, the set $\tilde{S}$ associated with $\tilde{B}$ must contain at least one more vertex from C'_ℓ. Let $\tilde{S}_\ell = \tilde{S} \cap S_\ell$. By definition of v^*, $\kappa_{v^*} \leq \kappa_v$ for any $v \in \tilde{S}$. Note that after augmenting $\tilde{B}$ along the edge $[p(v^*), v^*]$ at a cost of $2\kappa_{v^*}$, we have made all vertices in S_ℓ, and in particular all vertices in $\tilde{S}_\ell$, reachable from vertex 1 where the depot is located. This means that the algorithm in future iterations will never augment the graph $\tilde{B}$ along the edges $[p(v), v]$ for $v \in S_\ell$. Let $e_j = [p(v^*), v^*]$ and $D_j = \cup_{v \in S_\ell \backslash \{v^*\}} \{[p(v), v]\} \supseteq \cup_{v \in \tilde{S}_\ell \backslash \{v^*\}} \{[p(v), v]\}$.
2. It follows from the selection procedure of v^* in Step 4 of the Augmentation Algorithm that $c_{e_j} \leq z(\cup_{v \in \tilde{S}_\ell \backslash \{v^*\}} \{[p(v), v]\}) \leq z(D_j)$.

3. This observation follows directly from the fact that once a vertex is made reachable from the root by adding the null object loaded arcs in a certain augmentation iteration, then there is no need to augment along the same edge again. Also, if in some iteration, vertex v^* is made reachable by augmenting along the edge $[p(v^*), v^*]$, then all vertices that belonged to the same unreachable component as v^* become reachable from the root, and thus these vertices will never again be members $\tilde{S}$.
4. By construction, the multitype multigraph B' is connected and balanced, thus each edge of E is covered in B' at least twice. The Augmentation Algorithm augments B' along the edges $\cup_{j=1}^{\theta}\{e_j\}$, and it does not augment it along the edges in $\cup_{i=1}^{\theta} D_i$. The statement follows directly from the facts that these 2θ sets ($\{e_j\}$, and D_j) for $1 \le j \le L$ are disjoint, and $\cup_{j=1}^{\theta}(\{e_j\} \cup D_j) \subset E$.

Recall that $z(B')$ is a lower bound on the optimal SP cost. By the above statements, it follows that the total augmentation cost, satisfying $\sum_{j=1}^{\theta} 2c_{e_j} \le 0.5z(B')$, which proves that the worst-case performance ratio of the algorithm is 1.5, that is, $z(H) \le 1.5z^*$.

To complete the proof, we present an example (Figure 4) where the bound 1.5 is tight. Part (a) of the figure depicts an SP where the depot is located at the root of the tree, namely at vertex 1. The numbers on the edges denote their cost. The basic graph B is presented in part (b) of the figure. This graph is disconnected and is therefore augmented into B' which contains, in addition to the services paths of B, also the two loaded arcs $(1,3)^0$ and $(3,1)^0$, see part (c) of the figure. No feasible SP solution exists on B' as none of the objects 1 or 2 is available at vertex 3 when the vehicle enters there the first time. One can readily check that $z(B') = 4M + 4\xi$.

The fully connected components of B' are C_1 containing the root and vertices 2 and 3, and the unreachable component C_2 containing both vertices 4 and 5. The first iteration of the Augmentation Algorithm generates the set $\tilde{S} = \{4,5\}$, where both vertices 4 and 5 belong to the same unreachable component. In this case $\kappa_4 = \kappa_5 = M$, and therefore the algorithm chooses arbitrarily one of these vertices, say vertex 4 as v^*. Graph B' is augmented by adding to it the two loaded arcs $(3,4)^0$ and $(4,3)^0$. The resulting multitype multigraph is the heuristic solution H that the algorithm generates. According to this solution, the vehicle loads object 1 at vertex 1 and carries it to vertex 2, where it unloads it and loads object 2, which is carried to vertex 1. From there the vehicle travels empty to vertex 4 through vertex 3 and swaps objects 2 and 1 between vertices 4 and 5. Finally the vehicle travels empty from vertex 4 to the root. Clearly, $z(H) = 6M + 4\xi$.

The optimal SP solution to the above example is to start at the depot by loading object 1, and carry it directly to vertex 4, from there the vehicle loads object 2 which is carried to vertex 5. At vertex 5 object 1 is loaded and carried to vertex 2. There it is unloaded and object 2 is loaded to be carried to the root. Thus, $z^* = 4M + 4\xi$, implying that $\lim_{M\to\infty} z(H)/z^* = 1.5$. ∎

Acknowledgments

This research was carried out while the first author visited the Canada Research Chair in Distribution Management and the CIRRELT in Montreal. Thanks are due to the referees for their valuable comments.

REFERENCES

[1] S. Anily, M. Gendreau, and G. Laporte, The swapping problem on a line, SIAM J Comput 29 (1999), 327–335.

[2] S. Anily and R. Hassin, The swapping problem, Networks 22 (1992), 419–433.

[3] E. Arkin, R. Hassin, and L. Klein, Restricted delivery problems on a network, Networks 29 (1997), 205–216.

[4] M.J. Atallah and S.R. Kosaraju, Efficient solutions to some transportation problems with applications to minimizing robot arm travel, SIAM J Comput 17 (1988), 849–869.

[5] M.O. Ball and M.J. Magazine, Sequencing of insertions in printed circuit board assembly, Oper Res 36 (1988), 192–201.

[6] C. Basnet, L.R. Foulds, and J.M. Wilson, Heuristics for vehicle routing on tree-like networks, J Oper Res Soc 50 (1999), 627–635.

[7] C. Bordenave, M. Gendreau, and G. Laporte, A branch-and-cut algorithm for the non-preemptive swapping problem, Naval Res Logistics 56 (2009), 478–486.

[8] C. Bordenave, M. Gendreau, and G. Laporte, Heuristics for the mixed swapping problem, Comput Oper Res 37 (2010), 108–114.

[9] P. Chalasani and R. Motwani, Approximating capacitated routing and delivery problems, SIAM J Comput 28 (1999), 2133–2149.

[10] G.N. Frederickson and D.J. Guan, Preemptive ensemble motion planning on a tree, SIAM J Comput 21 (1992), 1130–1152.

[11] G.N. Frederickson and D.J. Guan, Nonpreemptive ensemble motion planning on a tree, J Algorithms 15 (1993), 29–60.

[12] G.N. Frederickson, M.S. Hecht, and C.E. Kim, Approximation algorithms for some routing problems, SIAM J Comput 7 (1978), 178–193.

[13] M.R. Garey and D.S. Johnson, Computers and intractability: a guide to the theory of NP-completeness, Freeman, New York, 1979.

[14] C. Hierholzer, Über die Möglichkeit einen Linienzug ohne Wiederholung und ohne Unterbrechning zu unifaren, Math Ann VI (1873), 30–32.

[15] N. Katoh and T. Yano, An approximation algorithm for the pickup and delivery vehicle routing problem on trees, Discrete Appl Math 154 (2006), 2335–2349.

[16] M. Labbé, G. Laporte, and H. Mercure, Capacitated vehicle routing on trees, Oper Res 39 (1991), 616–622.