1989

Preemptive Ensemble Motion Planning on a Tree

Gred N. Frederickson
Purdue University, gnf@cs.purdue.edu

D. J. Guan

Report Number:
89-865

Frederickson, Gred N. and Guan, D. J., "Preemptive Ensemble Motion Planning on a Tree" (1989).
Department of Computer Science Technical Reports. Paper 736.
<https://docs.lib.purdue.edu/cstech/736>

This document has been made available through Purdue e-Pubs, a service of the Purdue University Libraries.
Please contact epubs@purdue.edu for additional information.

PREEMPTIVE ENSEMBLE MOTION PLANNING
ON A TREE

Greg N. Frederickson
D. J. Guan

CSD-TR 865
February 1989
(Revised November 1991)

Preemptive Ensemble Motion Planning on a Tree

Greg N. Frederickson *

D. J. Guan †

Department of Computer Science
Purdue University
West Lafayette, Indiana 47907

November 13, 1991

Abstract

Consider the problem of finding a minimum cost tour to transport a set of objects between the vertices of a tree by a vehicle that travels along the edges of the tree. The vehicle can carry only one object at a time, and it starts and finishes at the same vertex of the tree. It is shown that if objects can be dropped at intermediate vertices along its route and picked up later, then the problem can be solved in polynomial time. Two efficient algorithms are presented for this problem. The first algorithm runs in $O(k + qn)$ time, where n is the number of vertices in the tree, k is the number of objects to be moved, and $q \leq \min\{k, n\}$ is the number of nontrivial connected components in a related directed graph. The second algorithm runs in $O(k + n \log n)$ time.

Keywords. Motion planning, vehicle routing, graph algorithms, directed minimum spanning tree, preemption.

*The research of this author was supported in part by the National Science Foundation under grants CCR-86202271 and CCR9001241 and by the Office of Naval Research under contract N00014-86-K-0689. A portion of this research was done while this author was on sabbatical leave at the International Computer Science Institute, Berkeley, CA.

†The research of this author was supported in part by the National Science Foundation under grant CCR-86202271 and by Purdue University under a David Ross Fellowship.

1 Introduction

Consider an undirected weighted graph with objects located at various vertices. Associated with each object is a destination vertex, to which that object is to be moved by a vehicle that traverses the edges of the graph. A fundamental problem in motion planning is to determine a tour of minimum cost for the vehicle to transport all objects from their initial positions to their destinations. In the case of general graphs, the problem is NP-hard, even if the vehicle can transport only one object at a time [11]. However, for special applications such as those that arise in robotics, it is reasonable to consider more restricted classes of graphs. In this paper and a companion paper [10] we consider problems where the graphs are trees, with a vehicle that can transport only one object at a time. In this paper we focus preemptive object movement. By this we mean that objects can be dropped, and picked up and transported at some later time in the transportation. A *drop* is an unloading of an object at a vertex that is not its destination.

We show that the problem can be solved in polynomial time, and present two efficient algorithms for it. Let n be the number of vertices in the tree and k the number of objects to be transported. Our first algorithm runs in $O(k + qn)$ time, where $q \leq \min\{k, n\}$ is the number of nontrivial strongly connected components in a related directed graph. Our second algorithm runs in $O(k + n \log n)$ time, which is better whenever k is $o(qn)$ and q is $\omega(\log n)$. These results contrast with our results in the case in which objects cannot be dropped at the intermediate vertices. In [10] we show that the nonpreemptive version of our problem is NP-hard, and give polynomial-time approximation algorithms. Note that viewed from the context of discrete job scheduling problems, it is not so surprising that the preemptive version of the problem

is polynomial while the nonpreemptive version is NP-hard. See for example the work on the problem of scheduling independent tasks on identical processors [20] [17] [13].

Our results compare with those of others as follows. For the case in which the graph is a general graph, Frederickson, Hecht and Kim have shown that the problem, which they termed the stacker-crane problem, is NP-hard [11]. For the cases in which the graph is either a simple path or a simple cycle and preemption is allowed, Atallah and Kosaraju have shown that the problem can be solved in $O(k+n)$ time [1]. For the cases in which the graph is either a simple path or a simple cycle and preemption is not allowed, Atallah and Kosaraju have presented algorithms that find an optimal solution in $O(k + n \log \beta(n, q))$ and $O(k + n \log n)$ time for path and cycle, respectively [1]. Frederickson has improved the latter time bound to $O(k + n \log \beta(n, q))$ [8].

We note that our problem appears to be a special case of exercise 7 in section 5.4.8 of [19], in which a bus moves in a tree-shaped network. However neither Knuth nor Karp, to whom the problem is attributed, knows of an efficient solution to this problem [16, 18]. In fact, we are able to show that, in the case that the bus has capacity greater than one, the problem is NP-complete even if preemption is allowed [9].

We make a number of observations about the structure of an optimal tour for the problem. In a manner similar to that in [1], we show that an optimal tour of the original problem can be obtained by solving the balanced version of the problem. While the structure of our approach is similar to that in [1], many additional ideas are needed to generate efficient algorithms when the graph is a tree. We introduce the notion of canonical tour, and show that every balanced problem has an optimal tour that is also canonical. This leads to the reduction of our problem to the prob-

lem of finding a directed minimum spanning tree of a certain directed graph. Our second algorithm uses a hierarchical decomposition of the tree to construct a directed graph with fewer arcs, which thus allows the directed minimum spanning tree to be computed faster.

The rest of the paper is organized as follows. In Section 2, we introduce notation and definitions, and discuss the transformation of the problem into a balanced version. In Section 3, we characterize a canonical solution and present our $O(k + qn)$ time algorithm for the problem. In Section 4, we present our second algorithm for the problem, which runs in $O(k + n \log n)$ time.

2 Generating a Balanced Problem

In this section we define the problem, along with the notion of moves, drops and a transportation. The structure of our approach is similar to that in [1]. Some of the definitions are repeated from [10] for the reader's convenience. In a manner similar to that in [1], we define a balanced version of a problem, and show that an optimal transportation for the original problem can be obtained by solving the balanced version of the problem. Standard terminology of graph theory, such as a directed graph and an Euler tour, is used in our paper, and can be found in Bondy and Murty [3].

An instance P of the motion planning problem on trees consists of a tree $T = (V, E)$, a non-negative cost $c(e)$ on each edge $e \in E$, a starting vertex $s \in V$, a set of objects O , and an *initial vertex* x_j and a *destination vertex* y_j for each object $j \in O$. Each object $j \in O$ is initially located at its initial vertex x_j and has to be moved to its destination vertex y_j by a vehicle that traverses the edges of the tree. The vehicle can carry only one object at a time, and the tour must start and finish at vertex s .

We observe that, for every instance P , there is an optimal transportation such that each object visits the vertices on the path from x_j to y_j exactly once and visits no other vertices. If this is not the case, then there is a cycle traversed by some object. We can replace the cycle traversed by that object by a non-carrying move. This modification does not increase the cost of the transportation, and repeatedly doing this yields a transportation with the desired property.

A *move* is designated by (x, y, c) , where x and y are vertices in V and c is an object $j \in O$ or 0. The vehicle moves along the unique path from x to y in the tree T , and carries an object c in the move if $c \neq 0$, and no object otherwise. Thus, a move

with $c \neq 0$ is called a *carrying move*, and a move with $c = 0$ is called a *non-carrying move*.

Let Q be a sequence of moves, (v_i, v_{i+1}, c_i) , $0 \leq i \leq r$. It is clear that two consecutive moves (u, v, c) and (v, w, c) can be expressed as (u, w, c) . Although in some cases we may want to decompose a move into a sequence of moves, we assume in general that $v_{i+1} \neq v_i$ and $c_{i+1} \neq c_i$ for $0 \leq i \leq r$. For each object $j \in O$, let Q_j be a sequence of moves obtained from Q by deleting every move (v_i, v_{i+1}, c_i) in Q with $c_i \neq j$. An object j is *transported* from x_j to y_j by Q if Q_j is a sequence of moves (u_i, u_{i+1}, j) , $0 \leq i \leq t$, with $u_0 = x_j$ and $u_{t+1} = y_j$. If $t > 0$ then the object j is *dropped* by Q at vertices $u_1, u_2, \dots, u_t$; otherwise it is not dropped by Q .

A *transportation* Q for P is a sequence of moves (v_i, v_{i+1}, c_i) , $0 \leq i \leq r$, such that $v_0 = v_r = s$, $v_{i+1} \neq v_i$, and every object $j \in O$ is transported from its initial position x_j to its destination y_j . The cost $c(Q)$ of a transportation Q is defined to be the sum of the costs of the edges the vehicle traverses. The motion planning problem is to find a transportation with minimum cost for an input instance P .

Figure 1: Insert Figure 1 approximately here.

An example of the motion planning problem in trees is given in Figure 1. There are eight vertices in T and four objects in O . The edges of the tree T are drawn in straight lines. An object j that has to be moved from x_j to y_j is drawn in curved arc from x_j to y_j with label j . The starting vertex is 0. The cost of each edge is 1, as indicated by its label.

We assume that every vertex of degree one or two in T is either s or x_j or y_j for some $j \in O$. A vertex of degree one and the edge incident at it can be deleted

from T if it is not s nor x_j or y_j for some $j \in O$. It is easy to see that a vertex of degree two and its adjacent edges can be replaced by a single edge with a cost the sum of the two edges deleted if it is not s nor x_j or y_j for some $j \in O$. Thus, the number of objects k is $\Omega(n)$.

Because every vertex of degree one is either s or x_j or y_j for some $j \in O$, every edge of T must be traversed by a valid transportation at least once. Furthermore, the number of times an edge is traversed in one direction must be equal to the number of times that edge is traversed in the other direction, since the vehicle starts and finishes at s .

Given an optimal transportation Q for a problem P , define a directed graph $D'(Q)$ on the vertex set V such that there is an arc from x to y labeled c if and only if there is a move (x, y, c) in Q . That is, each arc of $D'(Q)$ represents a move of Q . We shall call an arc that represents a carrying move a *carrying arc*, and an arc that represents a non-carrying move a *non-carrying arc*. It is easy to see that the graph $D'(Q)$ is Eulerian since Q is a transportation that starts and finishes at s . On the other hand, given an instance P , define a directed graph D_0 with vertex set V such that there is an arc from x_j to y_j labeled j if and only if there is an object $j \in O$ initially located at x_j that has to be moved to y_j . If this graph is Eulerian, then any Euler tour starting from s can easily be translated into an optimal transportation for P . Since each arc (x, y) in D_0 , as well as in $D'(Q)$, represents a move, we assign a cost $d(x, y)$ to it, equal to the sum of the costs of the edges from x to y in T . In a fashion similar to that in [1], the problem is reduced to a special type of graph augmentation problem, that of finding a minimum-cost set of noncarrying moves to add to D_0 to make it Eulerian.

One type of non-carrying moves added are the *balancing moves*. They are added so that every edge is traversed at least once and the number of times an edge is traversed in one direction is equal to the number of times that edge is traversed in the other direction. In the remainder of this section we shall show how to compute a set of balancing moves. Suppose that a set of balancing moves B is given. For each balancing move $(x, y, 0) \in B$, add a *balancing arc* $\langle x, y \rangle$ with label 0 to D_0 , and let the resulting graph be D . It is easy to see that the in-degree is equal to the out-degree for every vertex in the graph D , and each connected component of D is thus strongly connected. We shall call the graph D the *balanced graph*. Note that the augmentation by the balancing arcs may not be sufficient to get a transportation.

A strongly connected component of D is called a *trivial component* if it contains only one vertex and this vertex is not s . Otherwise, it is called *non-trivial component*. Note that a non-trivial component that contains x_j or y_j for some $j \in O$ must contain more than one vertex. Since each non-trivial component is Eulerian, no additional non-carrying moves between two vertices in the same non-trivial component are needed. All additional non-carrying moves will be used to connect non-trivial components. We call these non-carrying moves the *linking moves*. We shall show how to find a set of linking moves with minimum cost in the following sections.

There are, in general, many sets of balancing moves of minimum cost that satisfy the above conditions. In [10], we have shown how to construct a set of $O(k+n)$ balancing moves B with minimum cost, and such that the graph D will have minimum number of non-trivial components. The method is briefly described as follows. First, compute the number of balancing moves required at each edge so that, after these moves are added, every edge is traversed at least once and the number of times an

edge is traversed in one direction is equal to the number of times that edge is traversed in the other direction. Second, generate one balancing move on each edge in each direction. Third, generate the remaining balancing moves by merging moves of the form $(u, v_1, o_0), (v_1, v_2, o_1) \dots (v_i, w, o_i)$ into one move (u, w, o) so that there are at most $O(k + n)$ balancing moves.

Figure 2: Insert Figure 2 approximately here.

Figure 2 shows the balanced problem corresponding to the problem shown in Figure 1. Although balancing moves are non-carrying moves, we assign for convenience a unique label for each balancing move generated. The moves added are $(3, 1, 5)$, $(1, 3, 6)$, $(3, 7, 7)$, $(6, 3, 8)$, $(4, 1, 9)$, $(1, 0, 10)$, $(0, 2, 11)$, $(2, 5, 12)$, $(4, 0, 13)$, and $(0, 5, 14)$. Note that balancing moves $(3, 1, 5)$ and $(1, 3, 6)$ are added so that the edge $(1, 3)$ is traversed by the vehicle at least once.

In [10], we prove that for every instance P , there is an optimal transportation Q that contains all the moves in the balancing moves B generated by our algorithm. We also show that these balancing moves can be computed in linear time.

Lemma 1 [10] *Given an instance P for the motion planning problem on trees, the balanced graph D for P can be computed in $O(k + n)$ time.*

A motion planning problem is *balanced* if none of the moves in the balanced graph D are balancing moves. Given an instance P , we first construct a set of balancing moves B by the algorithm in [10]. For each balancing move from x to y in B , add an object $o_{x,y}$ to O with initial vertex x and destination vertex y . Let the resulting set of objects be O' . The new instance P' with the objects O' is called the balanced version of the original problem P . We show that adding the balancing

moves will not increase the cost of the transportation of the original problem. Part of the proof is similar to that of lemma 2 in [10], but we must also show that the capability to perform drops does not create a difficulty.

Lemma 2 *The costs of optimal transportations for P and its balanced version P' are equal.*

Proof Given a transportation Q' for P' , we can obtain a transportation Q for P from Q' by replacing each move (x, y, c) such that $c \notin O$ with a non-carrying move $(x, y, 0)$.

On the other hand, let Q be an optimal transportation for P with a minimum number of drops. First, construct a graph $D'(Q)$ with vertex set V such that there is an arc $\langle x, y \rangle$ labeled c if and only if there is a move (x, y, c) in Q . Since Q is a transportation, graph $D'(Q)$ is Eulerian. Second, replace every arc $\langle x, y \rangle$ in $D'(Q)$ with label 0 by a sequence of arcs $\langle v_i, v_{i+1} \rangle$ labeled 0, $0 \leq i \leq r - 1$, where $x = v_0, v_1, \dots, v_r = y$ are the list of vertices on the path from x to y in T . The modified $D'(Q)$ will remain Eulerian. Third, for each balancing move from x to y , replace a set of moves $(u_i, u_{i+1}, 0)$, $0 \leq i \leq t$ by a move $(x, y, o_{x,y})$, where $x = u_0, u_1, \dots, u_{t+1} = y$ are the list of vertices on the path from x to y in T . These moves must exist by the definition of balancing moves. Note that $D'(Q)$ will remain Eulerian under this operation. We claim that any Euler tour of $D'(Q)$ starting from s , is a transportation of P' .

The proof of the claim is as follows. It is easy to see that the graph $D'(Q)$ is Eulerian. Therefore, if no objects are dropped by Q , then any Euler tour of $D'(Q)$ starting from s is a transportation of P' . Consider any object j that is dropped at a vertex, say v . Then there must be two arcs with label j , say $\langle u, v \rangle$ and $\langle v, w \rangle$,

incident at v . We claim that any directed path from s to v in $D'(Q)$ must contain the arc $\langle u, v \rangle$. Otherwise, we can replace the two arcs $\langle u, v \rangle$ and $\langle v, w \rangle$ by one arc $\langle u, w \rangle$. Since there is a path from s to v that does not contain the arc $\langle u, v \rangle$, the graph $D'(Q)$ would still be connected. But, this modification eliminates the drop of object j at vertex v , which is a contradiction of the assumption that Q is a transportation with minimum number of drops. Thus in any Euler tour of $D'(Q)$, arc $\langle u, v \rangle$ must be traversed before arc $\langle v, w \rangle$. Therefore, any Euler tour of $D'(Q)$ starting from s is a transportation of P' .

Since the split and merge of moves will not change the total cost of the transportation, the cost of Q' is equal to the cost of Q . ■

The constructive proof of the above lemma gives a method to translate a transportation for P' into a transportation for P . That is, a transportation for the original problem P can be obtained from the transportation of the balanced problem P' by replacing each move (x, y, c) with $c \notin O$ by $(x, y, 0)$. In the following sections, we discuss how to compute a transportation for the balanced version of the problem.

3 Generating Canonical Transportations

In this section, we introduce the notion of a canonical transportation and show how it leads to an efficient algorithm for preemptive motion planning problem in a tree. We show how to reduce our problem to the problem of finding a minimum directed spanning tree in a directed graph [2, 4, 5, 6, 12, 22]. This then leads to an $O(k + qn)$ time algorithm.

Given an instance P of the motion planning problem in trees, our algorithm first constructs a balanced graph D as described in Section 2. Recall that if the balanced graph D is Eulerian and s is not an isolated vertex in D , then any Euler tour of D starting with vertex s is an optimal transportation with no objects dropped. We thus concentrate in this section on how to connect the nontrivial components of D with a minimum cost set of linking moves in the case that D is not Eulerian. Hence we shall assume in this section that a problem P is balanced.

3.1 Bridges and Canonical Transportations

In this subsection we identify a certain type of transportation, and show that there always exists a transportation of this type that is optimal. We first identify sets of vertices that are related to each strongly connected component of D . We then characterize how any given strongly connected component relates to other strongly connected components. We then define our special type of transportation, which we call canonical. Finally we show that there is always some canonical transportation that is an optimal transportation.

Let D_i be a nontrivial strongly connected component of D . We first identify sets of vertices that relate to each strongly connected component D_i . Let $j \in O$ be

an object with initial vertex x_j and destination vertex y_j . Note that x_j and y_j must be in the same nontrivial component of D . Thus, an object j is an object in D_i if x_j and y_j are both vertices in D_i . Define $IP(D_i)$ to be the set of vertices in D_i each of which is either the initial vertex for some object in D_i or the start vertex. (We choose the designator IP to stand for “initial position”.) Also define $V_T(D_i)$ to be the set of vertices each of which will be visited by some object in D_i . Note that every vertex v must be in $V_T(D_i)$ for some component D_i whenever the problem is balanced.

Figure 3: Insert Figure 3 approximately here.

Consider the example shown in Figure 3. Each straight line represents an edge of T . The cost of $(0,1)$ is 9 and all the other edges have cost 1. Each curved arc (x_j, y_j) with label j , $1 \leq j \leq 7$, represents a carrying arc of D . The starting vertex is 0. Note that the example is a balanced problem. The balanced graph D has four nontrivial components, namely, $D_1 = \{0\}$, $D_2 = \{1,7\}$, $D_3 = \{5,8\}$ and $D_4 = \{4,6,9\}$. $IP(D_1) = \{0\}$, $IP(D_2) = \{1,7\}$, $IP(D_3) = \{5,8\}$ and $IP(D_4) = \{4,6,9\}$. Note that $IP(D_i)$ is the same as the vertex sets in the component D_i if the problem is balanced. $V_T(D_1) = \{0\}$, $V_T(D_2) = \{1,0,2,3,5,7\}$, $V_T(D_3) = \{5,8\}$ and $V_T(D_4) = \{4,2,3,6,9\}$.

We next study how a given strongly connected component D_i relates to other strongly connected components. Let D_i and D_j be two nontrivial components of D . Since the vehicle can drop objects at intermediate vertices, a path from some vertex $u \in V_T(D_i)$ to some vertex $v \in IP(D_j)$ can be used to link component D_j to component D_i . We call such a path a *bridge* from D_i to D_j . The following example explains how a bridge can be used to connect two components in a transportation.

Assume that the path from u to v is a bridge from D_i to D_j . Let x_o and y_o be the initial and final positions of some object o of D_i such that u is on the path from x_o to y_o . Starting at any vertex in $IP(D_i)$, all objects in D_i and D_j can be transported with at most one drop. The vehicle first transports objects in D_i until it is carrying the object o at vertex u . It then drops object o at u , goes to vertex v and transports all objects in D_j . After finishing the objects in D_j , the vehicle must be at vertex v . It can then go back to vertex u and pick up object o and finish the rest of the moves of the objects in D_i .

Let each bridge be identified by: (1) the components D_i and D_j that it connects, (2) the origin u and the terminus v of the path, and (3) an object o in D_i such that u is a vertex in the path from x_o to y_o in T . We use $b_{i,j}$ to denote such a bridge from D_i to D_j . Note that, if $u \neq v$, then noncarrying moves $(u, v, 0)$ and $(v, u, 0)$ are the linking moves that are used to connect the components D_i and D_j . If $u = v$, then no linking moves are needed. In either case, if u is not a destination of any object in D_i , then the object o that is associated with the bridge $b_{i,j}$ is dropped at vertex u .

A component D_j is *reachable* from the starting vertex s , with respect to a set of bridges B , if either D_j contains the vertex s or there is a component D_i which is reachable from s with respect to B and there is a bridge $b_{i,j}$ in B from D_i to D_j . If we can find a set B of bridges that make all components reachable from the starting vertex s , then we can compute a transportation by the following procedure.

For each bridge $b_{i,j}$ in B , from D_i to D_j with origin u and terminus v , we add linking arcs $\langle u, v \rangle$ and $\langle v, u \rangle$ to D if the bridge is not a single vertex. Recall that each arc in D represents a move and has a label and a cost. The label of an arc $\langle u, v \rangle$ is an object or 0 and the cost is the distance from u to v in the graph T . The two arcs

$\langle u, v \rangle$ and $\langle v, u \rangle$ will have 0 as their labels and distance $d(u, v)$ for their costs. If the vertex u is not a destination of any object in the component D_i then the object o associated with the bridge will be dropped at vertex u . This is done by splitting the arc $\langle x_o, y_o \rangle$ at vertex u . In general, an object o can be associated with more than one bridge. Thus, the arc $\langle x_o, y_o \rangle$ may be split at more than one vertex. The splits are handled all together, rather than one bridge at a time. We shall show how to do this efficiently after we present the algorithm.

If B is of minimum cost over all sets of bridges that make all components reachable from s , then we call the resulting graph D_B the *augmented balanced graph*. Note that any Euler tour of the augmented balanced graph D_B will traverse the arc $\langle x, u \rangle$ before the arc $\langle u, y \rangle$. This is because u is not a terminus of any arc that can be reached without the bridge $b_{i,j}$. It is easy to see that the augmented balanced graph D_B defines a transportation for P with cost $c(D) + 2c(B)$, where $c(D)$ is the total cost of the arcs in the balanced graph D and $c(B)$ is the total cost of the bridges in B .

Consider the example shown in Figure 3. Recall that the balanced graph D has four nontrivial components, namely, $D_1 = \{0\}$, $D_2 = \{1, 7\}$, $D_3 = \{5, 8\}$ and $D_4 = \{4, 6, 9\}$. Let the path from 0 to 4 be the bridge $b_{1,4}$ that connects D_4 from D_1 . Let the path from 3 to 5 be the bridge $b_{4,3}$ that connects D_3 from D_4 . Let the path from 5 to 7 be the bridge $b_{3,2}$ that connects D_2 from D_3 . Note that bridge $b_{3,2}$ is needed in spite of the fact that the moves of D_2 pass through vertex 5, since reaching D_2 without initially going through vertex 5 is very expensive. Let 3 be the object that is associated with $b_{4,3}$. All the other bridges start from a destination vertex, thus, the objects associated with them are not used. Since these three bridges make

every component reachable from s , we can find a transportation $Q = (0, 4, 0)(4, 3, 3)(3, 5, 0)(5, 8, 6)(8, 5, 7)(5, 7, 0)(7, 1, 2)(1, 7, 1)(7, 3, 0)(3, 9, 3)(9, 6, 4)(6, 4, 5)(4, 0, 0)$.

Finally, we consider a special type of transportation. We want to show that for any balanced problem P in which objects can be dropped at the intermediate vertices, there is an optimal transportation Q such that each linking move is either the forward or the backward traversal of a bridge. We shall call such a transportation a *canonical transportation*. This reduces our problem to the problem of finding a minimum cost set of bridges that connect the components so that every component can be reached from the starting point s . This is the problem of finding a minimum directed spanning tree in a directed graph, which can be solved efficiently [12, 22].

We first study some properties of an optimal transportation of a balanced motion planning problem on trees, that will allow us to prove that there always exists a canonical transportation that is an optimal transportation.

Lemma 3 *No optimal transportation of a balanced problem can traverse an edge in the same direction more than once without carrying an object.*

Proof Given a balanced problem P , let Q be an optimal transportation. Without loss of generality, assume that all non-carrying moves of Q are $(x, y, 0)$, where (x, y) is an edge of T . Since every vertex in T must be visited by Q , $IP(D_i) = V_T(D_i) = \{v\}$ for each trivial component $D_i = \{v\}$. Consider the set B of the paths that are traversed by the non-carrying moves in Q . Each path in B is a bridge, since every vertex $v \in IP(D_i)$ for some component D_i . Since Q is a transportation, all vertices in T are reachable from s . Thus B contains a set of non-zero-length bridges for the set of all components. Note that no bridges can appear more than once in B , since it

could not increase the set of components reachable from s with respect to B . Thus, the lemma follows. ■

Given a problem P , let Q be an optimal transportation for P . Let $D'(Q)$ be a directed graph with vertex set V such that there is an arc from x to y labeled c if and only if (x, y, c) is a move in Q . Let $e = (u, v)$ be an edge of T such that e is traversed in Q when the vehicle is not carrying an object. Let $D'_e(Q)$ be a directed graph obtained from $D'(Q)$ by omitting the non-carrying moves on the edge (u, v) . That is, replace the non-carrying move $(x, y, 0)$ that traverses the edge e in the direction from u to v by two moves $(x, u, 0)$ and $(v, y, 0)$. Delete any degenerate moves $(u, u, 0)$ and $(v, v, 0)$ that arise whenever $x = u$ and $y = v$, respectively.

Lemma 4 *Let Q be an optimal transportation for P . Let D'_u and D'_v be the two strongly connected components of $D'_e(Q)$. Then the edge (u, v) is traversed in Q when the vehicle is carrying some object not in the component that contains s .*

Proof Without loss of generality, assume that s is in D'_u . Since every edge of T is traversed by some object, it is sufficient to show that no objects in D'_u can traverse the edge (u, v) . Assume that there were an object o in D'_u that is carried along the edge (u, v) . Then v is a bridge from D'_u to D'_v in $D'_e(Q)$. This would imply that a transportation with smaller cost than Q could be obtained by omitting the non-carrying moves on the edge e . Thus, no objects in D'_u can traverse the edge (u, v) . Therefore, the lemma follows. ■

Lemma 5 *Let Q be an optimal transportation of P , and $(x, y, 0)$ be the first non-carrying move in Q . Let $x = v_0, v_1, \dots, v_t = y$ be the sequence of vertices on the path from x to y in T . Let $\tilde{P}$ be an instance obtained from P by adding a set of required*

moves $(x, y, o_{x,y})$ and $(v_i, v_{i-1}, o_{v_i, v_{i-1}})$, $1 \leq i \leq t$ to P . Then optimal transportations for P and $\bar{P}$ have the same cost.

Proof Since the optimal transportation Q traverses the path from x to y without carrying an object, it must also traverse every edge on the path from y to x without carrying an object. For each edge (v_i, v_{i-1}) on the (y, x) -path, first find a move $(u, w, 0)$ in Q such that (v_i, v_{i-1}) is in the (u, w) -path in T , and then replace it by $(u, v_i, 0)$, $(v_i, v_{i-1}, 0)$ and $(v_{i-1}, w, 0)$. A transportation Q' of P' can be obtained from Q by replacing each non-carrying move $(v_i, v_{i-1}, 0)$ by $(v_i, v_{i-1}, o_{v_i, v_{i-1}})$, $1 \leq i \leq t$. It is easy to see that Q' and Q have the same cost.

On the other hand, let Q' be an optimal transportation for P' . A transportation Q'' for P can be obtained from Q' by replacing each move $(u, v, o_{u,v})$ such that $o_{u,v}$ is in M' but not in M by $(u, v, 0)$. It is easy to see that Q'' and Q' have the same cost. Therefore, the lemma holds. ■

Theorem 1 *Every balanced problem has an optimal transportation that is canonical.*

Proof Let P be a counterexample with a minimum number of nontrivial components in the balanced graph. Let Q be an optimal transportation for P , and $(u, v, 0)$ be the first non-carrying move in Q . There must be non-carrying moves in Q since an optimal transportation without non-carrying moves is, by definition, canonical. Note that the path from u to v must be a bridge between two nontrivial components of D . Let $u = v_0, v_1, \dots, v_t = v$ be the sequence of vertices in the path from u to v in T . Add a set of moves $(u, v, o_{u,v})$ and $(v_i, v_{i-1}, o_{v_i, v_{i-1}})$, $1 \leq i \leq t$, to M and let the resulting instance be $\bar{P}$. By Lemma 5, optimal transportations of P and $\bar{P}$ have the same cost. Since the balanced graph of $\bar{P}$ has fewer nontrivial components than the

balanced graph of P , $\bar{P}$ must have a transportation $\bar{Q}$ that is canonical. A canonical transportation Q' for P with the same cost as $\bar{Q}$ can be obtained from $\bar{Q}$ as follows.

Let $D'(\bar{Q})$ be a directed graph with vertex set V such that there is an arc from x to y labeled o if and only if (x, y, o) is a move of $\bar{Q}$. Delete the carrying arcs labeled with objects in $\bar{P}$ but not in P to give $D''(\bar{Q})$. Let $u = u_0, u_1, \dots, u_r = v$ be the vertices in the path from u to v upon which there are incident arcs. Let D'_i be the component of $D''(\bar{Q})$ that contains the vertex u_i , $1 \leq i \leq r$. Note that D'_0 is the component that contains s .

For each vertex u_i , $1 \leq i \leq r$, do the following. If there are carrying arcs incident to u_i , that is, u_i an initial position for some objects in some component in D'_i , then add the arcs $\langle u_{i-1}, u_i \rangle$ and the $\langle u_i, u_{i-1} \rangle$ to $D''(\bar{Q})$. Otherwise, there must be two linking arcs $\langle u_i, v'_i \rangle$ and $\langle v'_i, u_i \rangle$ incident on u_i . (Recall that u_i is a vertex on the (u, v) -path at which there are arcs incident on it.) First, find an object o in D'_i that visits the vertex u_i . By Lemma 4, such an object must exist. Second, split the carrying arc $\langle x_o, y_o \rangle$ at u_i . Finally, add the arcs $\langle u_{i-1}, v'_i \rangle$ and $\langle v'_i, u_{i-1} \rangle$ to $D''(\bar{Q})$. It is clear that the transformation does not increase the cost of the transportation, and an Euler tour of the resulting $D''(\bar{Q})$ starting with s yields the desired canonical transportation Q' of P . ■

Define a *directed bridging graph* A with vertex set the set of non-trivial components of D . For each ordered pair of distinct vertices D_i and D_j , the weight of arc $\langle D_i, D_j \rangle$ is equal to the sum of the costs on the edges of the minimum cost bridge $b_{i,j}$ from D_i to D_j . Let $c(D)$ be the sum of costs of all arcs in D .

Theorem 2 *A balanced problem P has an optimal transportation with cost $c(D)+2x$ if and only if the directed bridging graph A has a minimum directed spanning tree of*

weight x rooted at the component that contains s .

Proof Without loss of generality, let D_1 be the strongly connected component of D that contains s . We first show that a directed spanning tree S with root D_1 and weight x of A can be translated into a transportation of P with cost $c(D) + 2x$. For each arc $\langle D_i, D_j \rangle$ of S let $b_{i,j}$ be the corresponding bridge from D_i to D_j , and B be the set of these bridges. It is clear that every nontrivial component of D can be reached from s with respect to the bridges in B . Therefore, P has a transportation of cost $c(D) + 2x$.

We next show that there is an optimal transportation of cost y to the motion planning problem that can be translated into a directed spanning tree with root D_1 and weight $(y - c(D))/2$ of A . Let Q be an canonical and optimal transportation of P . By Theorem 1 there is such a transportation. Construct a directed spanning tree S for A as follows. Examine each move of Q , from the first one to the last. Whenever there is a non-carrying move $(u, v, 0)$ in Q and there is no $\langle D_j, D_i \rangle$ arc in S we add an arc $\langle D_i, D_j \rangle$ to S , where D_i is the component for the object of the preceding move and D_j is the component for the next move. Note that we also want to add an arc to S for a degenerate bridge. This can be done by examining two consecutive moves of Q . If an object from D_i is in the first move and an object from D_j is in the second move then there is a degenerate bridge in Q . Add an arc $\langle D_i, D_j \rangle$ to S if the arc $\langle D_j, D_i \rangle$ is not already in S . Since Q must visit all the components of D , S must span all the components. S must be a tree, otherwise we could delete some bridges that correspond to an arc of a cycle in S . The resulting set of arcs would still make all components reachable from s and we could generate a transportation which has less cost than Q , which is a contradiction. By Lemma 3, an optimal transportation must

traverse an edge zero times, or two times, once in each direction, without carrying any object. Therefore, the weight of S is equal to $(y - c(D))/2$. ■

With the above theorem, our problem is reduced to finding a minimum directed spanning tree with root D_1 that contains s in the bridging graph A . In the next subsection, we shall present an efficient algorithm for the problem.

Figure 4: Insert Figure 4 approximately here.

Consider once again the example in Figure 3. In Figure 4 we give the corresponding bridging graph A . There is a node for each of the strongly connected components, D_1, D_2, D_3 and D_4 . Consider the nodes D_1 and D_4 . The minimum-cost bridge from D_1 to D_4 is the path in T from vertex 0 to vertex 4. Thus the cost of arc $\langle D_1, D_4 \rangle$ is 2. Note that the cost function is not symmetric, since the minimum-cost bridge from D_4 to D_1 is the path in T from vertex 2 to vertex 0, of cost 1. Also note that some arcs have cost 0, as does $\langle D_2, D_3 \rangle$, since vertex 5 is both in the set $V_T(D_2)$ and in the set $IP(D_3)$. The other arcs correspond to minimum-cost bridges that are easily identified. The arcs in a directed minimum spanning tree are shown in bold, and have a total cost of 4.

3.2 An Efficient Algorithm

In this section we present the algorithm *with-drops* and show that it can be implemented to run in $O(k + qn)$ time. We first present the algorithm in a high-level description, and then discuss how to implement it efficiently. In particular, we discuss carefully how to compute the directed bridging graph A efficiently. We also compute efficiently how moves should be interrupted in the case that several drops must be made on the same move. Finally, we analyze the time required by the algorithm.

We first present our algorithm. Recall that the directed bridging graph was defined between Theorems 1 and 2 in the preceding subsection.

ALGORITHM *with-drops*

INPUT: an instance P of motion planning problem on trees.

OUTPUT: an optimal transportation Q for P .

METHOD:

1. Find the balanced graph D for the motion planning problem P .
2. Find the directed bridging graph A for D , rooted at the node representing the component that contains the start vertex.
3. Find a minimum directed spanning tree B of the graph A .
4. Find the augmented balanced graph D_B for D with bridges in B .
5. Output a transportation Q by finding an Euler tour of D_B starting from s .

We first show how to construct the bridging graph efficiently. The algorithm processes one nontrivial component of D at a time. Let D_i be a nontrivial component. With each bridge $b_{i,j}$ we associate the following information: the components D_i and D_j , the origin u and the terminus v of the path, and an object o in D_i such that o must visit the vertex u in the transportation. For each vertex u in $V_T(D_i)$, we use $a_i(u)$ to denote such an object. We show how to compute the vertex set $V_T(D_i)$. Note that the value of $a_i(v)$ for every vertex v in $V_T(D_i)$ can also be computed as we compute the vertex set $V_T(D_i)$.

Let s be the root of T . For each object o in D_i , let t be the nearest common ancestor of x_o and y_o . If t is not x_o or y_o , then replace the carrying arc $\langle x_o, y_o \rangle$ by two arcs $\langle x_o, t \rangle$ and $\langle t, y_o \rangle$, both labeled with the object o . Reorient each arc so that every arc is directed from a child toward an ancestor. For each vertex v in T let $into(v)$ be the list of arcs with terminus v , and let $outof(v)$ be a list of arcs with origin v . Each entry in $into(v)$ stores the address of the corresponding entry in $outof(u)$ that represents the arc $\langle u, v \rangle$. Each entry in $outof(u)$ stores the name of the arc that the entry represents. The lists $into(v)$ and $outof(v)$ for all vertices in T can be constructed in $O(k + n)$ time. Given these lists, we then call the recursive procedure *search* with parameter s .

For each vertex v , the procedure *search* determines if v is in $V_T(D_i)$, and computes the value $a_i(v)$ by maintaining a list of arcs $L(v)$ such that v is a vertex of the path from x to y for every arc $\langle x, y \rangle$ in $L(v)$. A vertex v is in $V_T(D_i)$ if and only if $L(v)$ is not empty. Given $L(v)$, the value $a_i(v)$ can also be computed in constant time, e.g., the label of the first arc in the list $L(v)$.

The procedure *search(v)* does the following. If v is a leaf, then let $L(v) = outof(v)$. Otherwise, if v is not a leaf, do the following. First, let $L(v)$ be empty. Second, for each child w of v , call *search(w)* and merge $L(w)$ to $L(v)$. Third, if $L(v)$ is not empty, then add v to $V_T(D_i)$, and let $a_i(v)$ be the original name of the first arc in $L(v)$. Otherwise, let $a_i(v) = 0$. Finally, delete each arc in $into(v)$ from the list $L(v)$. This completes the description of *search*.

Edge costs in the directed bridging graph A are computed as follows. Note that the graph induced by the vertex set $V_T(D_i)$ must be connected. Initialize $d(D_i, D_j) = \infty$ for all $i \neq j$. For each vertex v in $V_T(D_i)$ such that v is also in $IP(D_j)$ for some

$j \neq i$, add $\langle D_i, D_j \rangle$ to A with cost 0. These are the degenerate bridges. The bridges of length greater than zero are computed as follows. For each edge in the graph induced by $V_T(D_i)$, assign cost zero to that edge. Let v be any vertex in $V_T(D_i)$. Determine the shortest distances $d'(v, w)$ from v to every other vertex w , noting the last vertex v_w in $V_T(D_i)$ on a shortest path to w . Consider each vertex w , where $w \notin V_T(D_i)$ and $w \in IP(D_j)$ for some $j \neq i$. If $d(D_i, D_j)$, the distance from D_i to D_j , is greater than $d'(v, w)$ then update $d(D_i, D_j)$ to be $d'(v, w)$, and identify the corresponding path as (v_w, w) .

Lemma 6 *The directed bridging graph A can be computed in $O(k + qn)$ time, where q is the number of nontrivial components in the balanced graph D .*

Proof The tree can be rooted at s in $O(n)$ time. The processing time for each component is as follows. With $O(n)$ preprocessing time, the nearest common ancestor for each pair of vertices can be computed in $O(1)$ time [21, 15]. Thus, arcs in D_i can be processed in $O(k_i)$ time, where k_i is the number of objects in D_i . The set of vertices $V_T(D_i)$ can be computed in $O(k_i + n)$ time. With the doubly linked list for $L(v)$ and the address of each arc with terminus v in the list $into(v)$, the deletion of an arc in $L(v)$ can be done in $O(1)$ time. For each vertex in T , the algorithm uses $O(1)$ time to merge the list $L(w)$ into its parent's list, $O(|into(v)|)$ time in deleting arcs from the list $L(v)$ and $O(1)$ time in generating the value of $a_i(v)$. Since T is a tree, the single-source shortest path problem can be solved in $O(n)$ time. The update of the costs on the arcs of A can be done in $O(n)$ time. Therefore, the algorithm runs in $O(k_i + n)$ time for each component of D . Since there are q nontrivial components, the total computation can be implemented in $O(k + qn)$ steps. ■

Finally, we show that, given a set of bridges B , the augmented balanced graph can be computed in $O(n)$ time. Let u_i , $1 \leq i \leq r$ be the internal vertices on the path from x_o to y_o at which the arc $\langle x_o, y_o \rangle$ should split. Let $d_1(v)$ be the distance from s to v , in terms of the number of edges. We compute the position $p(u_i, x_o, y_o)$ of vertex u_i , $1 \leq i \leq r$, on arc $\langle x_o, y_o \rangle$ as follows. If u_i is an ancestor of x then $p(u_i, x_o, y_o)$ is $d_1(x) - d_1(u)$. Otherwise, $p(u_i, x_o, y_o)$ is $d_1(x) + d_1(u_i) - 2d_1(t)$, where t is the nearest common ancestor of x and y . Perform a lexicographic sort on all the triples $(x_o, y_o, p(u, x_o, y_o))$. This sorted list gives, for each arc $\langle x_o, y_o \rangle$, the order for vertices at which the arc $\langle x_o, y_o \rangle$ is to split. Let u'_i , $0 \leq i \leq r$ be such a sequence for $\langle x_o, y_o \rangle$. The arc $\langle x_o, y_o \rangle$ is then replaced by a set of arcs $\langle u'_i, u'_{i+1} \rangle$ $0 \leq i \leq r$, where $u'_0 = x_o$ and $u'_{r+1} = y_o$. These split arcs will have the same label as the original arc, but their costs, which represent distances in T , will be changed to the corresponding distances that the arcs represent. Since there are q nontrivial components in D , there are at most $q - 1$ bridges. Therefore, the augmented graph D_B can be computed in $O(n)$ time.

Theorem 3 *Given an instance P , let k be the number of objects to be moved and n be the number of vertices in T . The algorithm with-drops can be implemented to compute an optimal transportation for P in $O(k + qn)$ time.*

Proof The correctness of the algorithm is based on Theorem 2. The balanced graph D can be computed in $O(k+n)$ time. The directed bridging graph A can be computed in $O(k + qn)$ time, and has q vertices. The minimum directed spanning tree of A can be computed in $O(q^2)$ time [22, 12]. The augmented balanced graph can be constructed in $O(n)$ time. Since there are only $O(k + n)$ arcs in D , the generation

of the transportation Q can be computed in $O(k + n)$ time. Since $q \leq \min\{k, n\}$, *with-drops* can be implemented to run in $O(k + qn)$ time. ■

4 A Multi-Level Approach

In this section, we present another algorithm for our problem. It generates a variation of the bridging graph, called a multi-level bridging graph. This graph is based on a hierarchical decomposition of the tree that produces in general more nodes but fewer directed edges. This allows the directed minimum spanning tree algorithm to run faster in the case that the number of connected components is large as a function of the number of vertices. Our algorithm then runs in $O(k + n \log n)$ time. Thus, it is more efficient asymptotically than the algorithm in the preceding section whenever k is $o(qn)$ and q is $\omega(\log n)$.

We organize this section as follows. First we give a simple transformation for tree T that allows our hierarchical decomposition to be performed efficiently. Then we define our hierarchical decomposition and give an efficient algorithm to find the decomposition. We next specify simple preprocessing of the input that is necessary for generating the multi-level bridging graph. We then describe our algorithm *MULTIL*, which initializes the multi-level bridging graph and calls a recursive procedure *construct* that adds additional nodes and arcs to the multi-level bridging graph. We carefully analyze the size of the graph generated and the time to generate it. We then show how to extract an optimal solution from a directed minimum spanning tree of the multi-level bridging graph. Finally, we prove correctness and claim the time bound for our algorithm.

Assume that our tree is rooted at s . Our algorithm first uses a clustering approach to transform the tree into a binary tree. Given tree $T_0 = (V_0, E_0)$, we shall produce a binary tree $T = (V, E)$. A well-known transformation in graph theory [14, page 132] is used. For each vertex v with $d > 2$ children, $w_1, \dots, w_d$ and parent w_0 ,

replace v with new vertices $v_1, \dots, v_{d-1}$. Add edges $\{(v_i, v_{i+1}) | i = 1, \dots, d-2\}$, each of cost 0, and replace the edges $\{(v, w_i) | i = 1, \dots, d-1\}$ with edges $\{(v_i, w_i) | i = 1, \dots, d-1\}$, of corresponding costs, and replace the edges $\{(w_0, v), (v, w_d)\}$ with edges $\{(w_0, v_1), (v_{d-1}, w_d)\}$, of corresponding costs. The number of vertices and edges will increase by at most $n - 3$.

We consider a multi-level approach that generates a different type of bridging graph, which we call a *multi-level bridging graph*. For t a positive integer, let $A^t = (V^t, E^t)$ be the multi-level bridging graph with t levels. For $t > 1$, A^t has more nodes than the original one, but has fewer arcs whenever q is $\omega(\log n)$ and k is $o(n \log n)$. Our approach relies on partitioning the tree into clusters. Arcs in the multi-level bridging graph induced by the subtrees within the clusters, and are also induced by a tree describing the effect of moves across clusters. Our clusters are somewhat similar to, but a variation of, the clusters generated in [7] for a simply-connected topological partition.

Let z be a positive integer to be specified later. Let $T = (V, E)$ be a rooted binary tree. Let the root and at most two other vertices in T be identified as *required boundary vertices*. Let $E_1, E_2, \dots, E_l$ be a partition of E . Let the *root* s_i of subgraph $(V(E_i), E_i)$ be the (unique) vertex in $V(E_i)$ nearest the root s of T . An *induced boundary vertex* is a vertex that is in $V(E_i)$ and $V(E_j)$, for some i and some $j \neq i$. An *acceptable clustering* of T of parameter z , $z \geq 2$, is a partition $E_1, E_2, \dots, E_l$ of E satisfying the following properties:

1. The subgraph $T_i = (V(E_i), E_i)$ is a tree, for $i = 1, \dots, l$.
2. The number of boundary vertices in $V(E_i) - \{s_i\}$ is at most z , for $i = 1, \dots, l$.

3. There are at most $2z - 2$ edges in E_i , for $i = 1, \dots, l$.
4. There are at most three sets E_i such that there are both fewer than z edges in E_i and fewer than 3 boundary vertices in $V(E_i)$.

Each subgraph T_i is called a *cluster*.

Clusters can be generated as follows. Recursive procedure *cl_search* is called with parameters s and z . Procedure *cl_search*(v, z) partitions the edges in the subtree rooted at v into zero or more clusters, and one set of at most $z - 1$ edges. The clusters with their boundary vertices are output, and the set of remaining edges, with its boundary vertices, are returned to the calling procedure.

When *cl_search* is called with parameters v and z , the following is done. A cluster C and a set BV of boundary vertices are both initialized to the empty set. If v is both a leaf and a required boundary vertex, then v is inserted into BV , and C and BV are returned to the calling procedure. Otherwise, *cl_search* does the following. First, for each child w of v , edge (v, w) is inserted into C , *cl_search*(w, z) is called and returns C' and BV' , and C' and BV' are unioned into C and BV , respectively. Second, if $|C| \geq z$ or $|BV| = 2$ or v is a required boundary vertex, then v is inserted into BV , cluster C is printed, along with boundary vertices BV and root v , and C is reset to be empty and BV to be $\{v\}$. Third, C and BV are returned to the calling procedure.

Let a procedure *FINDCL* be the procedure that initially calls *cl_search* with parameters s and z , and let (C, BV) be returned to *FINDCL*. The set C will be empty, since s is a required boundary vertex.

Lemma 7 *Let T be a rooted binary tree of m edges. Let z be a positive integer,*

$z \geq 2$. The number of clusters in an acceptable clustering of T of parameter z is at most $1 + (2m + 2)/(z + 2)$.

Proof There are at most 3 clusters that contain required boundary vertices, and these may contain as few as one edge. We count the remaining l' clusters as follows. From among these l' clusters, let n_i be the number of clusters with i boundary vertices, for $i = 1, 2, 3$. First note that $n_1 + n_2 + n_3 = l'$. These clusters induce a tree T' , where the nodes correspond to clusters, and there is an edge between two clusters if they share a vertex in T . Thus $n_1 + 2n_2 + 3n_3 = 2(l' - 1)$ follows by noting that total degree in T' is twice the number of edges. From these equations we infer that $n_1 = n_3 + 2$.

Each of the n_3 clusters will contain at least two edges, and each of the $n_1 + n_2$ clusters will contain at least z edges. Thus $3 + 2n_3 + z(n_1 + n_2) \leq m$, which implies that $3 + 2n_3 + z(n_3 + 2) \leq m$. Thus $n_3 \leq ((m - 3 - 2z)/(z + 2))$. It also follows that $n_1 + n_2 \leq (m - 3 - 2n_3)/z$. Thus the total of all clusters is $3 + n_1 + n_2 + n_3$, which will be at most $3 + (m - 3 - 2n_3)/z + n_3$, which is at most $3 + (m - 3)z + (1 - 2/z)(m - 3 - 2z)/(z + 2)$, which equals the claimed bound, as long as $z \geq 2$. ■

Figure 5: Insert Figure 5 approximately here.

Considering the tree T in Figure 3, we give an acceptable clustering of T of parameter $z = 2$ in Figure 5. The clusters formed will have edge sets $E_1 = \{(0, 1)\}$, $E_2 = \{(0, 2), (2, 4)\}$, $E_3 = \{(2, 3), (3, 5)\}$, $E_4 = \{(5, 7), (5, 8)\}$, and $E_5 = \{(3, 6), (6, 9)\}$. Note that each of the five subgraphs identified is a tree. The roots of T_1 , T_2 , T_3 , T_4 and T_5 are 0, 0, 2, 5 and 3, respectively. Tree T_1 has one boundary vertex, vertex 0; tree T_2 has two boundary vertices, 0 and 2; tree T_3 has three boundary vertices, 2,

3 and 5; tree T_4 has one boundary vertex, vertex 5; and tree T_5 has one boundary vertex, vertex 3. Each tree has $z = 2 = 2z - 2$ edges except for tree T_1 , which has one edge.

We next specify the preprocessing of the input for the routine that creates the multi-level bridging graph. The preprocessing involves finding a reduced set of moves of size $O(n)$. The reduced set satisfies the property that for each component D_i , every vertex in $V_T(D_i)$ is covered by a move in the reduced set. For each component D_i do the following. First, choose some vertex v in $IP(D_i)$. Second, find a depth-first search tree of D_i rooted at v . Third, find a spanning tree of D_i rooted at v with arcs directed toward v . This can be done by reversing the direction of all arcs in D_i , finding a depth-first search tree rooted at v , and then reversing the direction of the arcs back to what they were. Fourth, union the arcs of the two trees together. This gives a set of arcs that are strongly connected, and of size proportional to the cardinality of $IP(D_i)$, in time proportional to the number of arcs in D_i . The union of these sets over all components D_i is the reduced set of moves used in generating the multi-level bridging graph. Clearly, the set of reduced moves is of size $O(n)$, and can be found in $O(k)$ time.

We assume that the input to our algorithms to construct bridging graphs is in the following form. First there is a weighted rooted binary tree T , of which the root and at most two other vertices are designated as required boundary vertices. Second is a reduced set M of moves $\langle x, y \rangle$ with each having a label indicating which component D_i it is a member of, along with the original name for the move. Since we will be using a multi-level approach, within the recursion these moves may represent portions of moves in the original problem. Thus in our input two moves with different component

labels may be incident on the same vertex. We avoid problems by assuming that in the recursion no moves have their endpoints treated as initial positions.

We next describe our algorithm *MULTI_L*, which constructs a multi-level bridging graph $A^t = (V^t, E^t)$. Algorithm *MULTI_L* initializes V^t and E^t as follows. For each component D_i , $i = 1, 2, \dots, q$, a node $\bar{i}$ is inserted into V^t . For each vertex v in T , $v = 0, 1, \dots, n-1$, a node v is inserted into V^t . Thus initially there are $q + n$ nodes in V^t . Initialize *label_count* to q . For each component D_i , $i = 1, 2, \dots, q$, for each vertex v in $IP(D_i)$, insert arcs $\langle v, \bar{i} \rangle$ and $\langle \bar{i}, v \rangle$ into E^t with cost 0, *orig_name* set to *null*, and *drop* set to v . For each edge (u, v) in T , with u closer to the start vertex than v , insert $\langle u, v \rangle$ into E^t with cost $c(u, v)$, *orig_name* set to *null*, and *drop* set to u . If the start vertex s is in a component D_i by itself, insert arc $\langle \bar{i}, s \rangle$ into E^t with cost 0, *orig_name* set to *null*, and *drop* set to s . Thus there are initially at most $3n$ arcs in E^t . Algorithm *MULTI_L* next sets up the set M of moves as follows. Each move $\langle x, y \rangle$ in M has a label i for component D_i containing $\langle x, y \rangle$, and *orig_name* set to " $\langle x, y \rangle$ ". Let the moves in M be ordered by label value. *MULTI_L* then calls a recursive procedure *construct*, which identifies additional nodes and arcs of A^t . The procedure is invoked with tree T , reduced set M of moves, and an appropriate number of levels t , which we identify later.

We now discuss our recursive procedure *construct* which is called with parameters T , M and t , where t is a positive integer. Let m be the number of edges in T . We choose a suitable constant $m_0 = 15$ at which to stop the recursion. If $t = 1$ or $m \leq m_0$, then *construct* handles T and M somewhat similarly to the approach in section 3. We shall specify this carefully after seeing how the recursion is handled.

If $t > 1$ and $m > m_0$, the following is done. Let $z = \lfloor m^{1-1/t}/2 \rfloor$. Note that

by the conditions on t and m , $z \geq 2$. An acceptable clustering $T_1, \dots, T_l$ of tree T for parameter z is found. If $l = 1$, then recursively invoke procedure *construct* with arguments T , M and $t - 1$. Otherwise, if $l > 1$ then do the following. Define the *compressed tree* $\overline{T}$ as follows. Initialize $\overline{T}$ as a copy of T . Next delete all vertices in T not on a path between two boundary vertices. Then while there is a nonboundary vertex v of degree 2, replace v and its two adjacent edges (u, v) and (v, w) by the edge (u, w) of cost $c(u, v) + c(v, w)$. Tree $\overline{T}$ is the resulting tree. Associated with each edge in $\overline{T}$ is a path in T , which we call a *basic path*. There are at most three basic paths in any one cluster. In the case that there are three, they all share one nonboundary vertex as an endpoint.

We define a set $\overline{M}$ of moves for $\overline{T}$, and sets M_i of moves for tree T_i in the clustering, $i = 1, \dots, l$, as follows. For each move $\langle x, y \rangle$ in M do the following. Suppose that the label of $\langle x, y \rangle$ is i . If there is a cluster T_j such that both x and y are in $V(E_j)$, then insert $\langle x, y \rangle$ into M_j with label i and the same value of *orig_name*. Otherwise do the following. Let u and v be boundary vertices on the path from x to y in T such that the path from u to v in T is of maximum length. If $u \neq x$, then x is in only one cluster T_j , and move $\langle x, u \rangle$ is inserted into M_j with label i and with $\text{orig_name}(x, u) = \text{orig_name}(x, y)$. If $v \neq y$, then y is in only one cluster $T_{j'}$, and move $\langle v, y \rangle$ is inserted into $M_{j'}$ with label i and with $\text{orig_name}(v, y) = \text{orig_name}(x, y)$. If $u \neq v$, insert $\langle u, v \rangle$ into $\overline{M}$, with label i and with $\text{orig_name}(u, v) = \text{orig_name}(x, y)$. This completes the description of how to handle each move $\langle x, y \rangle$. Since M can be examined in order of label value, the moves in $\overline{M}$ and in the sets M_i , for $i = 1, 2, \dots, l$ are generated in order of label value.

Whenever endpoint x or y of a move $\langle x, y \rangle$ is not a boundary vertex, then the

corresponding vertex u or v can be identified in constant time as follows. Assume that preorder and postorder numbers have been computed for T , so that ancestor testing can be done in constant time. Let s_j be the root of T_j , and let s'_j be the root of T'_j . If s_j is an ancestor of s'_j , then choose u as the boundary vertex in T_j that is an ancestor of y and a proper descendant of s_j , and choose v as s'_j . A similar approach applies if s'_j is an ancestor of s_j . If neither s_j nor s'_j is an ancestor of the other, choose u as s_j and v as s'_j .

The construction of M_j , $j = 1, \dots, l$ is completed as follows. Determine which edges in $\overline{T}$ are covered by moves in $\overline{M}$. For each such edge $e = (y_1, y_2)$, do the following. Increment *label_count*, and insert a node with index $\bar{i}$, where $i = \text{label_count}$, into V^t . These nodes can be viewed as transfer nodes: a number of different components may have moves that cover edge e , but in M_j these will all be represented by one move with label i . Let T_j be the cluster containing both y_1 and y_2 . Insert move $\langle y_1, y_2 \rangle$ into M_j with label i and *orig_name* set to null. Note that moves in M_j are still ordered by label value.

The processing of $\overline{T}$ is completed by generating additional arcs for E^t from $\overline{T}$ as follows. For each label i of moves in $\overline{M}$ do the following. Determine the set of edges e in $\overline{T}$ such that there is a move with label i in $\overline{M}$ that covers e . For each such edge e do the following. Choose some move $\langle x, y \rangle \in \overline{M}$ with label i that covers $e = (y_1, y_2)$, and let the node for e in V^t have index i' . Insert arc $\langle \bar{i}, \bar{i}' \rangle$ into E^t with cost 0, label equal to the label of move $\langle x, y \rangle$, and drop set to *null*.

For each cluster T_j , recursively invoke our procedure *construct* with arguments T_j , M_j and $t - 1$. This invocation will add some number of arcs and nodes to A^t . This completes the description of the recursion step of *construct*.

We now discuss how *construct* handles the case when $t = 1$ or $m \leq m_0$. Consider the set of labels for moves in M . For each label i do the following. Determine all vertices covered by moves with this label. For each such vertex u , if there is a move with label i and non-null original name that covers u , let $a_i(u)$ be the original name of such a move. For every vertex u covered by a move with label i , insert into E^t an arc $(\bar{i}, u)$ with cost 0, *orig_name* set to $a_i(u)$, and *drop* set to u . This completes the description of the basis case of *construct*, and with it the description of all of procedure *construct*.

We illustrate algorithm *MULTI-L* and procedure *construct* with an example. We take $t = 2$ and construct a 2-level bridging graph for the tree T and set of moves M shown in Figure 3. The resulting 2-level bridging graph is shown in Figure 6. Algorithm *MULTI-L* initializes V^t with nodes $\bar{1}, \bar{2}, \bar{3}, \bar{4}$, representing components D_1, D_2, D_3, D_4 , and nodes $0, 1, \dots, 9$ representing vertices $0, 1, \dots, 9$ in T . Also, *MULTI-L* initializes E^t with arcs, which we shall designate with quadruples $(\langle \bar{i}, v \rangle, c, o_n, d)$, where $\langle \bar{i}, v \rangle$ is an arc with cost c , original name o_n and drop vertex d . The arcs from nodes representing components to nodes representing initial positions are $(\langle \bar{2}, 1 \rangle, 0, null, 1)$, $(\langle 1, \bar{2} \rangle, 0, null, 1)$, $(\langle \bar{2}, 7 \rangle, 0, null, 7)$, $(\langle 7, \bar{2} \rangle, 0, null, 7)$, $(\langle \bar{3}, 5 \rangle, 0, null, 5)$, $(\langle 5, \bar{3} \rangle, 0, null, 5)$, $(\langle \bar{3}, 8 \rangle, 0, null, 8)$, $(\langle 8, \bar{3} \rangle, 0, null, 8)$, $(\langle \bar{4}, 4 \rangle, 0, null, 4)$, $(\langle 4, \bar{4} \rangle, 0, null, 4)$, $(\langle \bar{4}, 6 \rangle, 0, null, 6)$, $(\langle 6, \bar{4} \rangle, 0, null, 6)$, $(\langle \bar{4}, 9 \rangle, 0, null, 9)$, and $(\langle 9, \bar{4} \rangle, 0, null, 9)$. The arcs corresponding to edges in T are $(\langle 0, 1 \rangle, 9, null, 0)$, $(\langle 0, 2 \rangle, 1, null, 0)$, $(\langle 2, 3 \rangle, 1, null, 2)$, $(\langle 2, 4 \rangle, 1, null, 2)$, $(\langle 3, 5 \rangle, 1, null, 3)$, $(\langle 3, 6 \rangle, 1, null, 3)$, $(\langle 5, 7 \rangle, 1, null, 5)$, $(\langle 5, 8 \rangle, 1, null, 5)$, and $(\langle 6, 9 \rangle, 1, null, 6)$. We shall designate the moves in M by a triple $(\langle u, v \rangle, i, o_n)$, where i is the label, and o_n is the original name of some move. Thus M consists of $(\langle 1, 7 \rangle, 2, \langle 1, 7 \rangle)$, $(\langle 7, 1 \rangle, 2, \langle 7, 1 \rangle)$, $(\langle 5, 8 \rangle, 3, \langle 5, 8 \rangle)$,

$(\langle 8, 5 \rangle, 3, \langle \langle 8, 5 \rangle \rangle)$, $(\langle 4, 9 \rangle, 4, \langle \langle 4, 9 \rangle \rangle)$, $(\langle 6, 4 \rangle, 4, \langle \langle 6, 4 \rangle \rangle)$, and $(\langle 9, 6 \rangle, 4, \langle \langle 9, 6 \rangle \rangle)$.

For the sake of our example, we shall assume that $z = 2$. (Actually, the smallest value of m for which $z = 2$ would be $m = 16$, but considering a tree of this size would unnecessarily clutter the example.) We use the clusters as shown in Figure 5. The boundary vertices, besides the root (vertex 0), will be 2, 3, 5. Thus the compressed tree $\overline{T}$ contains edges $(0, 2)$, $(2, 3)$, and $(3, 5)$.

From M we generate $\overline{M}$ and sets M_i as follows. For $(\langle 1, 7 \rangle, 2, \langle \langle 1, 7 \rangle \rangle)$, we insert $(\langle 1, 0 \rangle, 2, \langle \langle 1, 7 \rangle \rangle)$ into M_1 , $(\langle 5, 7 \rangle, 2, \langle \langle 1, 7 \rangle \rangle)$ into M_4 , and $(\langle 0, 5 \rangle, 2, \langle \langle 1, 7 \rangle \rangle)$ into $\overline{M}$. For $(\langle 7, 1 \rangle, 2, \langle \langle 7, 1 \rangle \rangle)$, we insert $(\langle 7, 5 \rangle, 2, \langle \langle 7, 1 \rangle \rangle)$ into M_4 , $(\langle 0, 1 \rangle, 2, \langle \langle 7, 1 \rangle \rangle)$ into M_1 , and $(\langle 5, 0 \rangle, 2, \langle \langle 7, 1 \rangle \rangle)$ into $\overline{M}$. For $(\langle 5, 8 \rangle, 3, \langle \langle 5, 8 \rangle \rangle)$, we insert $(\langle 5, 8 \rangle, 3, \langle \langle 5, 8 \rangle \rangle)$ into M_4 . For $(\langle 8, 5 \rangle, 3, \langle \langle 8, 5 \rangle \rangle)$, we insert $(\langle 8, 5 \rangle, 3, \langle \langle 8, 5 \rangle \rangle)$ into M_4 . For $(\langle 4, 9 \rangle, 4, \langle \langle 4, 9 \rangle \rangle)$, we insert $(\langle 4, 2 \rangle, 4, \langle \langle 4, 9 \rangle \rangle)$ into M_2 , $(\langle 3, 9 \rangle, 4, \langle \langle 4, 9 \rangle \rangle)$ into M_5 , and $(\langle 2, 3 \rangle, 4, \langle \langle 4, 9 \rangle \rangle)$ into $\overline{M}$. For $(\langle 6, 4 \rangle, 4, \langle \langle 6, 4 \rangle \rangle)$, we insert $(\langle 6, 3 \rangle, 4, \langle \langle 6, 4 \rangle \rangle)$ into M_5 , $(\langle 2, 4 \rangle, 4, \langle \langle 6, 4 \rangle \rangle)$ into M_2 , and $(\langle 3, 2 \rangle, 4, \langle \langle 6, 4 \rangle \rangle)$ into $\overline{M}$. For $(\langle 9, 6 \rangle, 4, \langle \langle 9, 6 \rangle \rangle)$, we insert $(\langle 9, 6 \rangle, 4, \langle \langle 9, 6 \rangle \rangle)$ into M_5 .

Figure 6: Insert Figure 6 approximately here.

For the compressed tree $\overline{T}$, we create nodes and arcs as follows. Create and insert into V^t node $\overline{5}$ for edge $(0, 2)$, node $\overline{6}$ for edge $(2, 3)$, and node $\overline{7}$ for edge $(3, 5)$. Also insert into E^t $(\langle \overline{2}, \overline{5} \rangle, 0, \langle \langle 1, 7 \rangle \rangle, null)$, $(\langle \overline{2}, \overline{6} \rangle, 0, \langle \langle 1, 7 \rangle \rangle, null)$, $(\langle \overline{2}, \overline{7} \rangle, 0, \langle \langle 1, 7 \rangle \rangle, null)$, and $(\langle \overline{4}, \overline{6} \rangle, 0, \langle \langle 4, 9 \rangle \rangle, null)$.

Next, *construct* is applied to each cluster T_j . Considering M_1 , *construct* sets $a_2(1) = \langle 1, 7 \rangle$, $a_2(0) = \langle 1, 7 \rangle$, and inserts into E^t $(\langle \overline{2}, 0 \rangle, 0, \langle \langle 1, 7 \rangle \rangle, 0)$, and $(\langle \overline{2}, 1 \rangle, 0, \langle \langle 1, 7 \rangle \rangle, 1)$. Considering M_2 , *construct* sets $a_4(2) = \langle 4, 9 \rangle$, $a_4(4) = \langle 4, 9 \rangle$,

$a_5(0) = \text{null}$, and $a_5(2) = \text{null}$, and inserts into E^t $(\langle \bar{4}, 2 \rangle, 0, \langle 4, 9 \rangle, 2)$, $(\langle \bar{4}, 4 \rangle, 0, \langle 4, 9 \rangle, 4)$, $(\langle \bar{5}, 0 \rangle, 0, \text{null}, 0)$, and $(\langle \bar{5}, 2 \rangle, 0, \text{null}, 2)$. Considering M_3 , *construct* sets $a_6(2) = \text{null}$, $a_6(3) = \text{null}$, $a_7(3) = \text{null}$, and $a_7(5) = \text{null}$, and inserts into E^t $(\langle \bar{6}, 2 \rangle, 0, \text{"null"}, 2)$, $(\langle \bar{6}, 3 \rangle, 0, \text{"null"}, 3)$, $(\langle \bar{7}, 3 \rangle, 0, \text{"null"}, 3)$, and $(\langle \bar{7}, 5 \rangle, 0, \text{"null"}, 5)$. Considering M_4 , *construct* sets $a_2(5) = \langle 1, 7 \rangle$, $a_2(7) = \langle 1, 7 \rangle$, $a_3(5) = \langle 5, 8 \rangle$, and $a_3(8) = \langle 5, 8 \rangle$, and inserts into E^t $(\langle \bar{2}, 5 \rangle, 0, \langle 1, 7 \rangle, 5)$, $(\langle \bar{2}, 7 \rangle, 0, \langle 1, 7 \rangle, 7)$, $(\langle \bar{3}, 5 \rangle, 0, \langle 5, 8 \rangle, 5)$, and $(\langle \bar{3}, 8 \rangle, 0, \langle 5, 8 \rangle, 8)$. Considering M_5 , *construct* sets $a_4(3) = \langle 4, 9 \rangle$, $a_4(6) = \langle 9, 6 \rangle$, and $a_4(9) = \langle 9, 6 \rangle$, and inserts into E^t $(\langle \bar{4}, 3 \rangle, 0, \langle 4, 9 \rangle, 3)$, $(\langle \bar{4}, 6 \rangle, 0, \langle 9, 6 \rangle, 6)$, and $(\langle \bar{4}, 9 \rangle, 0, \langle 9, 6 \rangle, 9)$.

This completes the construction of the multi-level bridging graph A^t for our example. We note that for this particular example A^t is considerably larger than the regular bridging graph A . This is due to our example being relatively small, and the number of components being relatively small.

We next analyze the time requirements of procedure *construct*, and the number of nodes and arcs added by it to A^t .

Lemma 8 *Let T be a weighted rooted binary tree with m edges and at most 3 required boundary vertices. Let M be a set of k moves with $q \leq m+4$ different labels. Let d be the total number of endpoints of moves in M that are not required boundary vertices. Let t be a positive integer. Procedure *construct* uses $O(t(k + m^{1+1/t}))$ time. The number of nodes and arcs introduced into the multi-level directed bridging graph A^t by procedure *construct* are $O(m)$ and $O(t(k + m^{1+1/t}))$, respectively.*

Proof Suppose that $t = 1$ or $m \leq m_0$. Then no nodes are inserted into V^t by *construct*. For each label value i , determining the set of vertices covered by moves with label i , contracting the tree, finding shortest distances from v_i^t , and generating

the arcs reflecting these shortest distances can be performed in $O(k_i + m)$ time, where k_i is the number of moves with label i . Since there are $q \leq m + 4$ different labels, the time used is $O(k + m^2)$. It can easily be seen that at most $(m + 4)(m + 3)$ edges are generated in this case.

Suppose that $t > 1$ and $m > m_0$. By Lemma 7, at most $1 + (2m + 2)/(z + 2) < 1 + (2m + 2)/(m^{1-1/t}/2 + 3/2) = 1 + (4m + 4)/(m^{1-1/t} + 3) < 1 + (4m + 12m^{1/t})/(m^{1-1/t} + 3) = 1 + 4m^{1/t}$ clusters are created. Since all but at most 4 vertices in any cluster are deleted when $\bar{T}$ is generated, $\bar{T}$ has fewer than $4 + 16m^{1/t}$ vertices, and thus fewer than $3 + 16m^{1/t}$ edges. Thus *construct* generates $O(m^{1/t})$ nodes in handling $\bar{T}$. It also generates $O(q\bar{m})$ edges, where $\bar{m}$ is the number of edges in $\bar{T}$. Thus *construct* generates $O(m^{1+1/t})$ arcs, and uses $O(k + m^{1+1/t})$ time in handling $\bar{T}$. The time required to set up $\bar{T}$, $\bar{M}$ and M_j , $j = 1, 2, \dots, l$, and to handle $\bar{T}$ is $O(k + m^{1+1/t})$.

Let m_j be the number of edges in cluster C_j . From the clustering method, it follows that $\sum_{j=1}^l m_j = m$. By choice of parameter z , $m_j \leq m^{1-1/t}$. Let q_j be the number of labels of moves for tree T_j . We bound q_j as follows. There are $m_j + 1$ vertices in T_j , each of which can be an initial position for a different component. In addition, there can be one component for each of at most three basic paths in T_j . Thus there can be a total of at most $m_j + 4$ components in T_j .

We first analyze the number of arcs introduced by *construct* into E^t . Let $R(m, t)$ be the number of arcs introduced by *construct* for a tree with m edges, and with moves of at most $m + 4$ different labels, and parameter t . From the above discussion, $R(m, t)$ is bounded by the recurrence:

$$R(m, t) \leq \begin{cases} cm^2, & \text{for } t = 1 \text{ or } m \leq m_0; \\ cm^{1+1/t} + \sum_{j=1}^l R(m_j, t-1), & \text{for } t > 1 \text{ and } m > m_0, \end{cases}$$

where c is an appropriate constant.

We claim that $R(m, t) \leq ctm^{1+1/t}$. The proof is by induction on t . The basis, with $t = 1$, follows immediately, since *construct* generates $O(m^2)$ arcs. For the induction step, with $t > 1$, assume as the induction hypothesis that the claim is true for $t - 1$. Then we have

$$\begin{aligned} R(m, t) &\leq cm^{1+1/t} + \sum_{j=1}^l R(m_j, t-1) \\ &\leq cm^{1+1/t} + \sum_{j=1}^l c(t-1)m_j^{1+1/(t-1)}, \end{aligned}$$

by the induction hypothesis. The sum is maximized when the values for m_j are as large as possible, i.e., $2z - 2$. Thus

$$\begin{aligned} R(m, t) &\leq cm^{1+1/t} + c(t-1)(2z-2)^{1+1/(t-1)}m/(2z-2) \\ &< cm^{1+1/t} + c(t-1)(2z)^{1/(t-1)}m \\ &\leq cm^{1+1/t} + c(t-1)(m^{1-1/t})^{1/(t-1)}m \\ &= ctm^{1+1/t} \end{aligned}$$

We next bound the number of additional nodes introduced into the multi-level bridging graph by *construct* for a tree with m edges. We count the number of nodes resulting from basic paths. If $t = 1$ or $m \leq m_0$, then no new nodes are introduced. Otherwise, the tree is partitioned into l clusters. If $l = 1$, then no new nodes are created, but the procedure is called recursively. If $l > 1$, then a node is introduced for each basic path, of which there are at most 3 per cluster. Then the procedure is called recursively on each cluster. Consider a decomposition tree of the original tree T , where the root represents tree T , and every other node represents a cluster generated at some point by *construct*. Each node representing a cluster T' has as its children nodes representing the clusters that T' is partitioned into. The number of leaves in

the decomposition tree is less than or equal to m . The total number of children of all nodes with at least 2 children is less than $2m$. Thus the total number of nodes added to the multi-level directed bridging graph because of basic paths will be less than $6m$.

We next analyze the time used by *construct*. Let $T(m, k, k', t)$ be the number of arcs introduced by *construct* for a tree with m edges, k moves, of which k' have both endpoints not being boundary vertices, and with at most $m + 4$ different labels, and parameter t . Note that the number of vertices of degree 1 or 2 in T is at least $(m + 3)/2$. Since these vertices would have been deleted if they were not initial positions or destinations, we must have $k \geq (m + 3)/4$. For $m > m_0 = 15$, $k \geq 5$, and thus there is a constant c such that the time to handle $\bar{T}$ is at most $c(k - 3 + m^{1+1/t})$.

Let k_j be the number of moves in the problem for cluster T_j , and let k'_j be the number of these moves having both endpoints not being boundary vertices. In generating the problems for the clusters T_j , some moves for T may be split. If a move in T already has at least one endpoint that is a boundary vertex, then there can be a corresponding move in at most one cluster T_j . If a move in T already has both endpoints not being boundary vertices, if the move is split, then it is replaced by two moves in the clusters, each of which has at least one endpoint that is a boundary vertex. Also, each cluster can receive at most three new moves, corresponding to basic paths. Thus $\sum_{j=1}^l k_j \leq k + (k' - \sum_{j=1}^l k'_j) + 3l$. It follows that the function $T(m, k, k', t)$ is bounded by the recurrence:

$$\begin{aligned} & T(m, k, k', t) \\ & \leq \begin{cases} c(k + m^2), & \text{for } t = 1 \text{ or } m \leq m_0; \\ c(k - 3 + m^{1+1/t}) + \sum_{j=1}^l T(m_j, k_j, k'_j, t - 1), & \text{for } t > 1 \text{ and } m > m_0, \end{cases} \end{aligned}$$

where c is an appropriate constant.

We claim that $T(m, k, k', t) \leq ct(k + k' - 3 + m^{1+1/t})$. The proof is by induction on t . The basis, with $t = 1$, follows immediately. For the induction step, with $t > 1$, assume as the induction hypothesis that the claim is true for $t - 1$. Then we have

$$\begin{aligned}
T(m, k, k', t) &\leq c(k - 3 + m^{1+1/t}) + \sum_{j=1}^l T(m_j, k_j, k'_j, t-1) \\
&\leq c(k - 3 + m^{1+1/t}) + \sum_{j=1}^l c(t-1)(k_j + k'_j - 3 + m^{1+1/(t-1)}),
\end{aligned}$$

by the induction hypothesis. The sum is maximized when the values for m_j are as large as possible, i.e., $2z - 2$. Thus

$$\begin{aligned}
T(m, k, k', t) &\leq c(k - 3 + m^{1+1/t}) + c(t-1)\left(\sum_{j=1}^l k_j + \sum_{j=1}^l k'_j - 3l + (2z - 2)^{1+1/(t-1)}m/(2z - 2)\right) \\
&< c(k - 3 + m^{1+1/t}) + c(t-1)\left((k + k' - \sum_{j=1}^l k'_j + 3l) + \sum_{j=1}^l k'_j - 3l + (2z)^{1/(t-1)}m\right) \\
&\leq c(k - 3 + m^{1+1/t}) + c(t-1)(k + k' + (m^{1-1/t})^{1/(t-1)}m) \\
&\leq ct(k + k' - 3 + m^{1+1/t})
\end{aligned}$$

This concludes the proof. ■

Upon the return of *construct* to algorithm *MULTI-L*, all nodes and a multiset of arcs of the multi-level bridging graph have been identified. A 2-pass radix sort is then performed to sort the arcs lexicographically, and eliminate all but the least expensive of multiple arcs. For each such arc the label and *orig_name* are brought along. The directed minimum spanning tree algorithm of [12] can then be applied to find a directed minimum spanning tree rooted at the node corresponding to the component that contains the start vertex.

The directed minimum spanning tree of A^t can be translated into a minimum cost set of bridges as follows. There will be a bridge in the set from component D_i to component D_j if and only if there is a directed path in the directed minimum spanning tree to node $\bar{j}$ from either node $\bar{i}$ or an initial position in D_i , that contains no intermediate node with index in $\{\bar{1}, \bar{2}, \dots, \bar{q}\}$. The first arc in this directed path carries in its *orig_name* field the name of a move that should be interrupted. (If *orig_name* is *null*, or if the *drop* location is an endpoint of the move, then no move is interrupted, implying that the bridge starts at an initial position.) The first arc on this path that enters a node with index in $\{0, 1, \dots, n-1\}$ will contain in *drop* the name of the vertex at which to drop the object. Once these values are known, the transportation can be constructed as in Section 3.

We return to our running example. A directed minimum spanning tree is shown in bold for our multi-level bridging graph in Figure 6. Since there are paths $P_1 = \bar{1}, 0, 2, 4, \bar{4}$, $P_2 = \bar{4}, \bar{6}, 3, 5, \bar{3}$, and $P_3 = 5, 7, \bar{2}$ in the directed minimum spanning tree, there are bridges from D_1 to D_4 , D_4 to D_3 , and D_3 to D_2 . The first arcs on each of these paths that enter a node with index in $\{0, 1, \dots, n-1\}$ are $\langle \bar{1}, 0 \rangle$, $\langle \bar{6}, 3 \rangle$, and $\langle 5, 7 \rangle$, respectively. These arcs have drop field equal to 0, 3, and 5, respectively. Thus the first bridge is from 0 to 4, the second is from 3 to 5, and the third is from 5 to 7. The first arcs on paths P_1 , P_2 and P_3 are $\langle \bar{1}, 0 \rangle$, $\langle \bar{4}, \bar{6} \rangle$, $\langle \bar{3}, 7 \rangle$, respectively. These arcs have original name field equal to *null*, $\langle 4, 9 \rangle$ and $\langle 5, 8 \rangle$, respectively. It follows that no move is interrupted for the first bridge. Since 3 is not an endpoint of $\langle 4, 9 \rangle$, move $\langle 4, 9 \rangle$ is interrupted at 3 for the second bridge. Since one of the endpoints of $\langle 5, 8 \rangle$ is the drop value, no move is interrupted for the third bridge. We note that we could just as well have chosen for P_1 the path $0, 2, 4, \bar{4}$, since 0 is an initial position

in D_1 . Since the drop field of arc $\langle 0, 2 \rangle$ is 0, and the original name of $\langle 0, 2 \rangle$ is *null*, no change would result.

Theorem 4 *Let T be a weighted rooted binary tree with n vertices, for which there is a set of k moves. Algorithm *MULTIL* finds a minimum-cost preemptive transportation in $O(k + n \log n)$ time.*

Proof We first address correctness. First we claim that the arcs that comprise any bridge are contained in the multi-level bridging graph A^t . Clearly, for every tree edge an arc is introduced, so that the only issue is whether the correct direction is chosen for the arc. But in any traversal of the tree, the first time any given edge is traversed, it will be traversed in the direction away from the start vertex.

Next we claim that for any initial position u of a component D_i , and any vertex v in $V_T(D_i)$, there is a corresponding path in A^t from u to v of cost 0. First note that by introducing arcs from $\bar{i}$ to each initial position in D_i , and vice versa, there is a path of cost 0 in A^t between any pair of initial positions in D_i . Next consider the move $\langle x, y \rangle$ in the reduced set of moves that covers vertex v in $V_T(D_i)$. It can be shown by induction on t that there is a path $\bar{i} = \bar{i}_0, \bar{i}_1, \dots, \bar{i}_p, v$ of cost 0. Thus there is a directed minimum spanning tree of A^t of cost equal to the cost of a minimum-cost set of bridges.

Finally, we verify that the appropriate information is associated with each arc in A^t . Whenever a move is split in our construction of A^t , the original name of the move is retained. Furthermore, in generating an arc $\langle \bar{i}, u \rangle$ to indicate that a move with label i covers vertex u , $drop(\bar{i}, u)$ is set to u . This completes the discussion of correctness.

We next discuss the time used by our algorithm. Let t be a positive integer. By Lemma 8, the time to generate V^t and a multiset containing E^t is $O(t(n + n^{1+1/t}))$, assuming that the reduced set of moves is used, so that k is $O(n)$, and noting that the number of edges is $n - 1$. Multiple arcs can be deleted in $O(tn^{1+1/t})$ time. Since the algorithm of [12] uses $O(m + n \log n)$ time on a graph of n nodes and m arcs, the time to find a directed minimum spanning tree is $O(tn^{1+1/t} + n \log n)$. The time to translate a directed minimum spanning tree of A^t into a set of bridges with drop points specified is proportional to the size of the tree. The time to generate the transportation, given this information is $O(k + n)$. Therefore, the algorithm *MULTIL* can be implemented to run in $O(k + tn^{1+1/t} + n \log n)$ time. Choosing $t = \log n$ yields a running time of $O(k + n \log n)$. ■

Acknowledgment. We would like to thank the referee for his helpful comments.

References

- [1] M. J. Atallah and S. R. Kosaraju. Efficient solutions to some transportation problems with application to minimizing robot arm travel. *SIAM Journal on Computing*, 17(5):849–869, 1988.
- [2] F. Bock. An algorithm to construct a minimum directed spanning tree in a directed network. In Benjamin Avi-Itzhak, editor, *Developments in operations research; Proceedings of the Third Annual Israel Conference on Operations Research*, pages 29–44, Gordon and Breach, New York, 1971.
- [3] J. A. Bondy and U. S. R. Murty. *Graph Theory with Applications*. The MacMillan Press, Ltd, London, England, 1976.
- [4] P. M. Camerini, L. Fratta, and F. Maffioli. A note on finding optimum branchings. *Networks, An International Journal*, 9(4):309–312, 1979.
- [5] Y. J. Chu and T. H. Liu. On the shorest arborescence of a directed graph. *Scientia Sinica*, 14(10):1396–1400, 1965.
- [6] J. Edmonds. Optimum branchings. *Journal of Research of the National Bureau of Standards—B. Mathematics and Mathematical Physics*, 71B(4):233–240, 1976.
- [7] G. N. Frederickson. Data structures for on-line updating of minimum spanning trees, with applications. *SIAM Journal on Computing*, 14(4):781–798, November 1985.

- [8] G. N. Frederickson. *A Note on the Complexity of a Simple Transportation Problem*. Technical Report CSD-TR-809, Department of Computer Science, Purdue University, West Lafayette, Indiana, September 1988. *SIAM Journal on Computing*, accepted for publication.
- [9] G. N. Frederickson and D. J. Guan. Ensemble motion planning with a vehicle of capacity greater than one. manuscript in preparation, 1989.
- [10] G. N. Frederickson and D. J. Guan. *Nonpreemptive Ensemble Motion Planning on a Tree*. Technical Report CSD-TR-864, Department of Computer Science, Purdue University, West Lafayette, Indiana, February 1989.
- [11] G. N. Frederickson, M. S. Hecht, and C. E. Kim. Approximation algorithms for some routing problems. *SIAM Journal on Computing*, 7(2):178–193, May 1978.
- [12] H. N. Gabow, Z. Galil, T. Spencer, and R. E. Tarjan. Efficient algorithms for finding minimum spanning trees in undirected and directed graphs. *Combinatorica*, 6(2):109–122, 1986.
- [13] R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17(2):416–429, 1969.
- [14] Frank Harary. *Graph Theory*. Addison-Wesley Publishing Company, Inc., Reading, Massachusetts, 1969.
- [15] D. Harel and R. E. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM Journal on Computing*, 13(2):338–355, 1984.
- [16] R. M. Karp. private communication, 1988.

- [17] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103, Plenum Press, New York, 1972.
- [18] D. E. Knuth. private communication, 1988.
- [19] D. E. Knuth. *The Art of Computer Programming. Volume III: Sorting and Searching*. Addison-Wesley, Reading, Massachusetts, 1973.
- [20] R. McNaughton. Scheduling with deadlines and loss functions. *Management Science*, 6(1):1–12, October 1959.
- [21] B. Schieber and U. Vishkin. On finding lowest common ancestors: simplification and parallelization. *SIAM Journal on Computing*, 17(6):1253–1262, 1988.
- [22] R. E. Tarjan. Finding optimum branchings. *Networks, An International Journal*, 7(1):25–35, 1977.

Figure 1. An example of the motion planning problem in trees.

Figure 2. A balanced graph (in bold) for the problem in Figure 1.

Figure 3. A second example of the motion planning problem.

Figure 4. The directed bridging graph for Figure 3, with a directed minimum spanning tree in bold.

Figure 5. Clusters of the tree in Figure 3.

Figure 6. The multi-level bridging graph for Figure 3, with a directed minimum spanning tree in bold.