Efficient Optimal Multi-Location Robot Rendezvous

Ken Alton and Ian M. Mitchell
Department of Computer Science
University of British Columbia
UBC CS TR-2007-17
October 5, 2007

Abstract

We present an efficient algorithm to solve the problem of optimal multi-location robot rendezvous. The rendezvous problem considered can be structured as a tree, with each node representing a meeting of robots, and the algorithm computes optimal meeting locations and connecting robot trajectories. The tree structure is exploited by using dynamic programming to compute solutions in two passes through the tree: an upwards pass computing the cost of all potential solutions, and a downwards pass computing optimal trajectories and meeting locations. The correctness and efficiency of the algorithm are analyzed theoretically, while a discrete robotic clinic problem and a continuous robot arm problem demonstrate the algorithm's practicality.

This is an extended version of a paper submitted to ICRA 2008 [1].

I. Introduction

Path planning is a central area of study in robotics, but most current algorithms find an efficient path for only a single robot at a time. Coordinated path planning for multiple robots has received increased attention recently, but is often difficult because the complexity of many techniques increases considerably with each additional robot. For example, using dynamic programming to find optimal collision-free paths is reasonable for 2 robots [2], but it becomes intractable with the addition of more robots because the planning is done in the high dimensional cross product of the robot state spaces, and the complexity of such algorithms grows exponentially with dimension.

We focus here on a particular type of coordinated robot path planning problem and, in so doing, we are able to find a very efficient algorithmic solution. More specifically, we examine optimal coordinated multi-robot multi-location rendezvous, an extension of the frugal feeding problem considered in [3]. Given a hierarchical structure that describes which robots are to meet and which robots are to continue on to future meetings, our dynamic programming algorithm can compute the optimal meeting locations and the optimal paths between meetings with complexity linear in the number of meetings. To simplify the scenarios, we assume central and complete knowledge of the map(s) and robots states, and we ignore collisions between the robots. Despite this simplification, we believe that core aspects of our algorithm for solving the robot rendezvous problem can be utilized in real world robot applications. To demonstate the practical potential of our algorithm, we use it to solve two example problems: a

discrete problem involving a meeting hierarchy in a robotic medical clinic and a continuous problem involving robotic arms cooperating to deliver some cargo from a source to a destination.

The main contribution of this paper is the presentation of a dynamic programming principle (DPP) and related algorithm that can efficiently find optimal solutions to a broad category of hierarchical multi-location robot rendezvous problems. An important property of the algorithm is that no calculation is done in a state space with dimension greater than that of the individual robots. We define the type of problems considered and our notation in Section III. Mathematical analysis in Section IV shows that the hierarchical problem structure implies that a DPP can be used to efficiently compute costs of potential solutions. The algorithm presented in Section V utilizes the DPP to find optimal meeting locations and connecting trajectories with a complexity linear in the size of the meeting hierarchy. Finally, in Section VI we use the algorithm to solve both a discrete robotic clinic and a continuous robot-arm rendezvous problem. Although the continuous problem is not analyzed with the same rigour as the discrete problem, we offer the example as a proof of concept but leave the theory for future work.

II. Related Work

Research in robot path planning is considered a central endeavor in the development of mobile robotics [4]. Approaches to robot path planning are diverse and include potential function methods, sampling-based methods, trajectory planning methods and combinations of these [5], [6]. Most related to this work are the dynamic programming algorithms for solving shortest path problems on grids [7], [8]; however, most path planning research, including that described in the above references, focuses on single robots.

In this paper, we investigate a dynamic programming method for the multi-location rendezvous of multiple robots. This problem is distinct from the generalized Fermat-Torricelli problem that finds a unique point minimizing the sum of distances from a given set of points [9]. The hierarchical facility location problem [10] involves finding the location of facilities of several levels to serve customers efficiently. This problem is more difficult than the one we consider because typically an optimal number of facilities and the tree structure linking facilities must be found in addition to optimal locations for those facilities. For the robot meeting problem, we assume that this tree structure already exists and develop an efficient algorithm to optimize the location of meetings (i.e. facilities).

This work was motivated by [3], in which the authors describe a robot refueling problem where the goal is to find the optimal rendezvous locations for a fuel-tanker robot to meet individually with each of a collection of worker robots. Our work extends the restricted locations case of that paper. We generalize the problem to include any hierarchy of robot meetings, and show that efficiency can be gained by assuming that the potential meeting locations are nodes in a spatial graph on which commuting costs are only defined between neighboring nodes. Finally, we demonstrate that computation on a grid can be used to approximate a continuous problem with a potentially complex cost function.

III. Problem Description

A robot meeting involves one or more robots colocating at a state within a discrete space. Except for the final meeting, one robot continues on from each meeting to the next meeting.[1] Imagine a meeting tree, where meetings begin at the leaves and progress through the tree culminating in one final meeting at the root. For this problem we fix the meeting structure as well as which robots must attend any particular meeting, but we allow the meeting locations to vary. In other words, we are not concerned with the combinatorial aspects of designing a sequence of meetings, which may be required for a general hierarchical facility location problem [10]. We wish to minimize, over all possible meeting locations, the cost of a given meeting tree. The cost of a meeting tree is measured by summing the cost of all robot commutes between meetings as well as the cost of each meeting. The avoidance of collisions between the robots during the commutes is not considered.

A. Formulation

We make some formal definitions regarding the space in which the robots move and the cost of movement within that space. Let there be a finite discrete state space $\mathcal{X}$. We use the words *state* and *location* interchangeably. If it is possible to commute from state $x \in \mathcal{X}$ to state $y \in \mathcal{X}$ say that y is a neighbor of x and write $y \in \mathcal{N}(x)$, where $\mathcal{N}(x)$ is the set of neighbors of x. For convenience we define a set $\mathcal{N}^{-1}(x) = \{y \mid x \in \mathcal{N}(y)\}$, the set of nodes for which x is a neighbor. Let $d : \mathcal{X}^2 \to \mathbb{R}^+$ be a positive commuting cost function, where $d(x, y)$ gives the cost of commuting from state $x \in \mathcal{X}$ to $y \in \mathcal{X}$ and $d(x, y) = \infty$ if $y \notin \mathcal{N}(x)$. Note that the above formulation is equivalent to a directed graph $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \mathcal{X}$, an edge $e = (x, y) \in \mathcal{E}$ if $y \in \mathcal{N}(x)$, and the weight of e is $d(x, y)$. Let $z = (z_1, z_2, \ldots z_L)$, be a trajectory through $\mathcal{X}$ with each $z_l \in \mathcal{X}$ and such that $z_{l+1} \in \mathcal{N}(z_l)$, for $1 \leq l \leq L(z) - 1$. We may use L in place of $L(z)$ where the trajectory z is obvious. For example, we may refer to the last element in z as simply z_L. Define $\lambda_d(z)$ to be the total cost of trajectory z:

$$\lambda_d(z) = \sum_{l=1}^{L(z)-1} d(z_l, z_{l+1}), \tag{1}$$

where $L(z)$ is the number of nodes in trajectory z. Define $\delta_d(x, y)$ to be the minimum total cost of any trajectory from x to y under the commuting cost function d:

$$\delta_d(x, y) = \min_{z \in \mathcal{Z}(x,y)} \lambda_d(z),$$

where $\mathcal{Z}(x, y)$ is the set of all trajectories that begin at x and end at y. Note that $\delta_d(x, x) = 0$ and $\delta_d(x, y) > 0$ for $x \neq y$.

A meeting tree structure is used to define the problem, as well as compute and store potential solutions. Let Υ be a *meeting tree* with ρ being the *root meeting node*. Each leaf node corresponds to a starting state of a single robot, whereas each non-leaf node in the tree Υ corresponds to a single meeting of one or more robots. A meeting

[1] In fact, two or more robots may continue on from a meeting so long as they travel together in a group. Since the group moves as a single entity to the next meeting, the essential properties of the problem remain the same.

of one robot involves requiring a robot to incur a meeting cost at some potentially unknown meeting location but not to meet with any other robots before continuing on to its next meeting. From any meeting a single robot will continue on to the next meeting, incurring a commuting cost, unless it is the root meeting, in which case the tree of meetings is completed.

Each *meeting node* η contains the number of children K and an array of child nodes κ. To define a meeting tree problem, each node η also contains a node problem definition, including a positive *meeting cost function* $c : \mathcal{X} \to \mathbb{R}^+$ and a positive *commuting cost function* $d : \mathcal{X}^2 \to \mathbb{R}^+$. The d function for the root node ρ is of no consequence to the problem and is considered undefined. Let $c(x) = +\infty$ indicate that a meeting cannot take place at x and $d(x, y) = +\infty$ indicate that a robot cannot commute directly from x to y. To define a meeting tree solution, each node η must contain a *meeting location* $p \in \mathcal{X}$. We may also wish to determine a *meeting-to-meeting trajectory* z for each node η with the exception of the root ρ. This trajectory is the path the robot follows from the meeting corresponding to η to the parent meeting. Finally, for each node η we define a *meeting value function* $v : \mathcal{X} \to \mathbb{R}$ and a *meeting-plus-commute value function* $w : \mathcal{X} \to \mathbb{R}$, which are explained further in Section IV and are used by Algorithm 1 to compute the optimal meeting locations.

symbol	type	description
$\eta.K$	$\mathbb{Z}^+$	number of children
$\eta.\kappa$	meeting node array	children
$\eta.c$	function : $\mathcal{X} \to \mathbb{R}$	meeting cost function
$\eta.d$	function : $\mathcal{X}^2 \to \mathbb{R}$	commuting cost function
$\eta.p^{(*)}$	$\mathcal{X}$	meeting location
$\eta.z^{(*)}$	a trajectory in $\mathcal{X}$	meeting-to-meeting trajectory
$\eta.v^{(*)}$	function : $\mathcal{X} \to \mathbb{R}$	meeting value function
$\eta.w^{(*)}$	function : $\mathcal{X} \to \mathbb{R}$	meeting-plus-commute value function

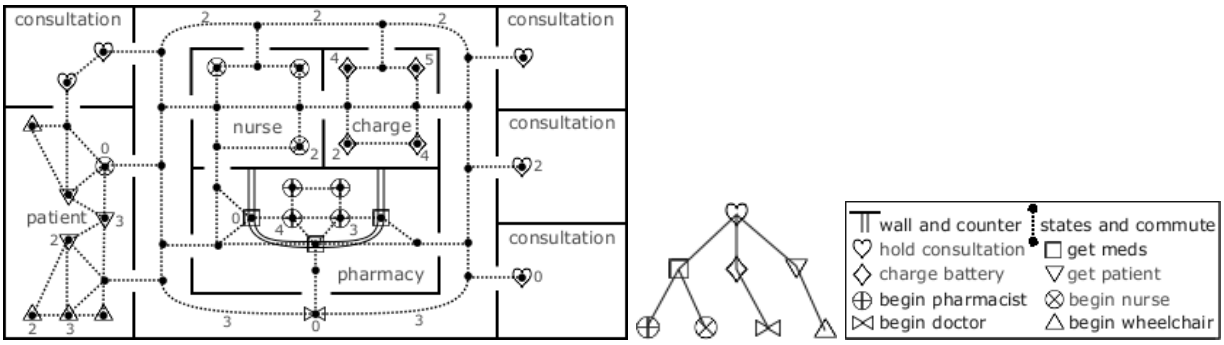
TABLE I

COMPONENTS OF MEETING NODE η. VARIABLES $\eta.p$, $\eta.z$, $\eta.v$, AND $\eta.w$ MAY BE SUPERSCRIPTED USING * TO INDICATE THAT THEY CORRESPOND TO AN OPTIMAL MEETING TREE.

As shown in Table I we use a dot notation to select components of a meeting node η. We define a recursive function f that computes the value (i.e. total cost) of a meeting subtree rooted at node η.

$$f(\eta) = \sum_{k=1}^{\eta.K} \left[f(\eta.\kappa[k]) + \delta_{\eta.\kappa[k].d}(\eta.\kappa[k].p, \eta.p) \right] + \eta.c(\eta.p), \tag{2}$$

where $\eta.\kappa[k]$ is the kth child of η. Note that f takes only a single node η as a parameter, but is really a function of all meeting locations in the subtree rooted at η. This function adds the meeting cost $\eta.c(\eta.p)$ at the current meeting location to the sum over all children of the value $f(\eta.\kappa[k])$ of the meeting subtree rooted at the child and the commuting cost $\delta_{\eta.\kappa[k].d}(\eta.\kappa[k].p, \eta.p)$ from the child meeting location to the current meeting location. We further define a related function g that computes the value of a meeting subtree rooted at node η in addition to the



(a) (b) (c)

Fig. 1. (a) Map of medical clinic. The rooms in the clinic are labeled. A number represents the cost of the meeting or commute to which it is adjacent; all unlabelled meetings and commutes have unit cost. (b) Meeting tree. (b) Legend of symbols used in the map and the meeting tree.

commute from the last meeting location to a state $x \in \mathcal{X}$:

$$g(\eta, x) = f(\eta) + \delta_{\eta.d}(\eta.p, x). \tag{3}$$

Similar to f, g is a function of all meeting locations in the subtree rooted at η. Using function g we can simplify the definition of f:

$$f(\eta) = \sum_{k=1}^{\eta.K} g(\eta.\kappa[k], \eta.p) + \eta.c(\eta.p). \tag{4}$$

We use the dot notation on a tree Υ to select a particular component of all nodes; for example, $\Upsilon.p$ is a tree of meeting locations. Such a tree $\Upsilon.p$ represents a potential solution to the problem definition tree $\Upsilon.(c, d)$. The goal is to find a minimal-cost solution tree $\Upsilon.p^*$ (which may not be unique) given problem tree $\Upsilon.(c, d)$:

$$\Upsilon.p^* \in \arg\min_{\Upsilon.p} f(\rho). \tag{5}$$

B. Example Problem: Patient Consultation

The setting of our example problem is a robotic medical clinic, a map of which is shown in Figure 1(a). The state space $\mathcal{X}$ is represented by the bullets on the map and the possible commutes between states are represented by dotted lines. The goal is to efficiently get a patient to a medical consultation with a robotic doctor and a robotic nurse in a consultation room. The meeting tree used to define this problem is illustrated in Figure 1(b). The symbols used to distinguish each type of meeting are labelled in Figure 1(c) and the states in $\mathcal{X}$ where each type of meeting may occur are indicated by the appropriate meeting symbol on the map. In this example, at most one type of meeting may be held at any one state $x \in \mathcal{X}$, although in general this restriction is not required.

The patient begins in the patient room. One of 4 possible robotic wheelchairs ($\eta = \triangle$) must retrieve the patient and take her to the consultation. The wheelchair may pick up the patient ($\eta = \nabla$) in one of 3 states. One of 4 possible robotic nurses ($\eta = \bigotimes$) must attend the consultation, while picking up medications at the pharmacy on the

way. A robotic pharmacist must fill the medications ($\eta = \bigoplus$) at one of 4 states behind the counter of the pharmacy. It then meets the nurse ($\eta = \square$) at one of 3 states at the counter to hand over the medications. Finally, the doctor must get from its initial state ($\eta = \bowtie$) to the consultation, while charging its batteries ($\eta = \diamondsuit$) at one of 4 charging stations on the way. The consultation ($\rho = \heartsuit$) may occur at one of 5 possible states.

Each meeting location x has a cost $\eta.c(x)$ associated with it. Some meetings locations have a cost specified on the map, while all others have unit cost. For a class of robots with multiple starting locations, such as the wheelchairs and nurses, each robot within that class may have a different initial state and start-up cost but will otherwise be identical, i.e., the costs for future commutes and meetings will not depend on the starting state of the robot. Each commute from state x to y also has a cost $\eta.d(x, y)$ associated with it. Some commutes have a cost specified on the map, while all others have unit cost. For simplicity, the commute costs for all robots are the same, with some exceptions. One exception is that the pharmacist begins inside the rounded pharmacy counter and may only visit states inside or on the counter. Furthermore, no other robot is permitted inside the pharmacy counter—they may only go as far as states on the counter. Another exception is that the wheelchair robot is not permitted in the nurses room or charge room while carrying the patient.

IV. Dynamic Programming

Define the *optimal meeting value function* $\eta.v^*(x)$ as

$$\eta.v^*(x) = \min_{\Upsilon.p,\ \eta.p=x} f(\eta). \tag{6}$$

In other words, $\eta.v^*(x)$ specifies the minimal total cost of the meeting subtree rooted at η subject to the condition that the meeting corresponding to η is located at x. Define the *optimal meeting-plus-commute value function* $\eta.w^*(x)$ as

$$\eta.w^*(x) = \min_{\Upsilon.p} g(\eta, x). \tag{7}$$

In other words, $\eta.w^*(x)$ specifies the minimal total cost of the meeting subtree rooted at η and the commute from the meeting location $\eta.p$ to x. Because of the recursive definitions of f in (4) and g in (3), $\eta.v^*$ and $\eta.w^*$ can be computed independent of any ancestors of η.

We observe three important properties of the meeting tree problem that allow optimal meeting locations $\Upsilon.p^*$ (and optimal meeting-to-meeting trajectories $\Upsilon.z^*$) to be found efficiently. The first observation is that the value functions $\eta.v^*$ and $\eta.w^*$ of any node η can be computed using just the value functions $\eta.\kappa[k].w^*$ of the children of η (see Theorem 1). This property allows $\eta.v^*$ and $\eta.w^*$ to be computed for all nodes η in a single pass through the tree Υ from the leaves towards the root (see Algorithm 2). The second observation is that the value $\eta.w^*(x)$ for any node η and any state $x \in \mathcal{X}$ can be computed using only the value $\eta.v^*(x)$ at the same state $x \in \mathcal{X}$ and the values $\eta.w^*(y)$ at the neighboring states $y \in \mathcal{N}^{-1}(x)$ (see Theorem 2). This property allows Dijkstra's algorithm to be used to compute $\eta.w^*$ in a single pass through the states in $\mathcal{X}$ (see Algorithm 3). The final observation is that the value functions $\Upsilon.v^*$ and $\Upsilon.w^*$ together contain sufficient information to find $\Upsilon.p^*$ and $\Upsilon.z^*$ in a single

pass through the tree Υ from the root towards the leaves (see Theorem 3 and Algorithms 1 and 4). We formalize the first two observations below, and the third in the next section.

The following theorem establishes a dynamic programming principle that for any node η defines the value function $\eta.v^*$ using only the value functions $\eta.\kappa[k].w^*$ of the children of η and the meeting cost function $\eta.c$, and defines the value function $\eta.w^*$ using only the value function $\eta.v^*$ and the commute cost function $\eta.d$.

Theorem 1: The value function $\eta.v^*$ satisfies

$$\eta.v^*(x) = \sum_k \eta.\kappa[k].w^*(x) + \eta.c(x), \tag{8}$$

for all $x \in \mathcal{X}$ for all nodes η in Υ. The value function $\eta.w^*$ satisfies

$$\eta.w^*(x) = \min_{y \in \mathcal{X}} \left[\eta.v^*(y) + \delta_{\eta.d}(y, x)\right], \tag{9}$$

for all $x \in \mathcal{X}$ for all nodes η in Υ.

The proof first shows that (8) holds for the case where η is a leaf. It then shows that (8) holds for the alternative case where η has children. Finally, it shows that (9) holds for all nodes η in Υ.

Proof: Let η be a leaf in Υ and let $x \in \mathcal{X}$. The following equation begins with the definition of $\eta.v^*$ from (6):

$$\begin{aligned}
\eta.v^*(x) &= \min_{\Upsilon.p,\ \eta.p = x} f(\eta) \\
&= \min_{\Upsilon.p,\ \eta.p = x} \eta.c(\eta.p) \\
&= \eta.c(x).
\end{aligned}$$

The second equality uses the definition of function f from (4), considering η has no children. The final equality is due to the constraint $\eta.p = x$ in the minimization. Since the sum in (8) disappears when η has no children, (8) is satisfied.

Let η be a node in Υ with one or more children. The following equation begins with the definition of $\eta.v^*$ from (6):

$$\begin{aligned}
\eta.v^*(x) &= \min_{\Upsilon.p,\ \eta.p = x} f(\eta) \\
&= \min_{\Upsilon.p,\ \eta.p = x} \left\{ \sum_k g(\eta.\kappa[k], \eta.p) + \eta.c(\eta.p) \right\} \\
&= \min_{\Upsilon.p} \sum_k g(\eta.\kappa[k], x) + \eta.c(x) \\
&= \sum_k \min_{\Upsilon.p} g(\eta.\kappa[k], x) + \eta.c(x) \\
&= \sum_k \eta.\kappa[k].w^*(x) + \eta.c(x).
\end{aligned}$$

The second equality uses the definition of function f from (4). For the third equality, the constraint $\eta.p = x$ is removed by replacing $\eta.p$ with x in the expression. Also $\eta.c(x)$ is removed from the minimization since it does not depend on $\Upsilon.p$. For the fourth equality, the $\min$ operator is brought inside the sum because the $\sum$ operator

is monotone in each addend and $g(\eta.\kappa[k], x)$ for the kth child depends only on the meeting positions p of nodes within the the subtree rooted at $\eta.\kappa[k]$. The final equality follows from the definition of $\eta.w^*$ from (7) and results in the RHS of (8).

Therefore, since we have considered both the cases where η is a leaf and where η has children, (8) holds for all nodes η in Υ for all $x \in \mathcal{X}$.

Now let η be any node in Υ. The following equation begins with the definition of $\eta.w^*$ from (7):

$$\begin{aligned}
\eta.w^*(x) &= \min_{\Upsilon.p} g(\eta, x) \\
&= \min_{y \in \mathcal{X}} \left\{ \min_{\Upsilon.p,\ \eta.p=y} g(\eta, x) \right\} \\
&= \min_{y \in \mathcal{X}} \left\{ \min_{\Upsilon.p,\ \eta.p=y} [f(\eta) + \delta_{\eta.d}(\eta.p, x)] \right\} \\
&= \min_{y \in \mathcal{X}} \left\{ \min_{\Upsilon.p,\ \eta.p=y} f(\eta) + \delta_{\eta.d}(y, x) \right\} \\
&= \min_{y \in \mathcal{X}} \left[\eta.v^*(y) + \delta_{\eta.d}(y, x) \right].
\end{aligned}$$

The second equality holds because minimizing over $\Upsilon.p$ with fixed $\eta.p = y$ and then minimizing over all $\eta.p = y$ has the same result as miniming over $\Upsilon.p$ all at once. The third equality is by (3). For the fourth equality we use the constraint $\eta.p = y$ to replace $\eta.p$ with y in $\delta_{\eta.d}(\eta.p, x)$ and then remove it from the inner minimization since it is independent of the minimizer. The final equality follows from the definition of $\eta.v^*$ in (6) and results in the RHS of (9). Therefore, since we have not assumed a particular $x \in \mathcal{X}$, (9) holds for all nodes η in Υ for all $x \in \mathcal{X}$. ■

Corollary 1: The value functions $\eta.v^*$ and $\eta.w^*$ satisfy

$$\eta.w^*(x) \leq \eta.v^*(x), \tag{10}$$

for all $x \in \mathcal{X}$ for all nodes η in Υ. Furthermore, for

$$\check{x} \in \arg\min_{x \in \mathcal{X}} \eta.w(x), \tag{11}$$

we have $\eta.w(\check{x}) = \eta.v(\check{x})$.

Proof: Let $x \in \mathcal{X}$ and η be a node in Υ. By (9), we have

$$\begin{aligned}
\eta.w^*(x) &= \min_{y \in \mathcal{X}} \left[\eta.v^*(y) + \delta_{\eta.d}(y, x) \right] \\
&\leq \eta.v^*(x) + \delta_{\eta.d}(x, x) = \eta.v^*(x)
\end{aligned} \tag{12}$$

Let $\check{x}$ be as defined in (11). Assume, for the moment, that $\eta.w(\check{x}) < \eta.v(\check{x})$. Then by (9) there exists some state $\tilde{y} \in \mathcal{X}$ such that $\tilde{y} \neq \check{x}$ and

$$\eta.w(\check{x}) = \eta.v(\tilde{y}) + \delta_{\eta.d}(\tilde{y}, \check{x})$$

Because $\delta_{\eta.d}(\tilde{y}, \check{x}) > 0$ for $\tilde{y} \neq \check{x}$ and by (12) we have

$$\begin{aligned}
\eta.w(\check{x}) &= \eta.v(\tilde{y}) + \delta_{\eta.d}(\tilde{y}, \check{x}) \\
&> \eta.v(\tilde{y}) \geq \eta.w(\tilde{y}),
\end{aligned}$$

contradicting (11). Consequently, the assumption must be false and by (12) we have $\eta.w(\check{x}) = \eta.v(\check{x})$. ■

The following theorem establishes a dynamic programming principle that for any node η and any $x \in \mathcal{X}$ defines the value $\eta.w^*$ at x using only the value $\eta.v^*$ at x; the values $\eta.w^*$ at y, such that $y \in \mathcal{N}^{-1}(x)$; and the commute costs $\eta.d$ from y to x, such that $y \in \mathcal{N}^{-1}(x)$.

Theorem 2: The value function $\eta.w^*$ satisfies

$$\eta.w^*(x) = \min\left\{\eta.v^*(x), \min_{y \in \mathcal{N}^{-1}(x)} \left[\eta.w^*(y) + \eta.d(y,x)\right]\right\} \tag{13}$$

for all $x \in \mathcal{X}$ for all nodes η in Υ.

Proof: Let η be a node in Υ and let $x \in \mathcal{X}$. Let $y^* \in \mathcal{X}$ be such that

$$y^* \in \arg\min_{y \in \mathcal{X}} \left[\eta.v^*(y) + \delta_{\eta.d}(y,x)\right]. \tag{14}$$

If $y^* = x$, then we have by (9) and (14)

$$\eta.w^*(x) = \eta.v^*(x). \tag{15}$$

Otherwise, $y^* \neq x$. Assume, for the moment, there exists some $\hat{y} \in \mathcal{N}^{-1}(x)$ such that

$$\eta.w^*(\hat{y}) + \eta.d(\hat{y},x) < \eta.v^*(y^*) + \delta_{\eta.d}(y^*,x),$$

Then by (9) there exists some state $\tilde{y} \in \mathcal{X}$ such that

$$\begin{aligned}\eta.w^*(\hat{y}) + \eta.d(\hat{y},x) &= \eta.v^*(\tilde{y}) + \delta_{\eta.d}(\tilde{y},\hat{y}) + \eta.d(\hat{y},x)\\ &< \eta.v^*(y^*) + \delta_{\eta.d}(y^*,x),\end{aligned}$$

contradicting (14). Thus, our assumption was incorrect and for all $y \in \mathcal{N}^{-1}(x)$

$$\eta.w^*(y) + \eta.d(y,x) \geq \eta.v^*(y^*) + \delta_{\eta.d}(y^*,x). \tag{16}$$

Let $z^* \in \mathcal{Z}(y^*,x)$ be a minimal-cost trajectory from y^* to x; i.e.,

$$\delta_{\eta.d}(y^*,x) = \lambda_{\eta.d}(z^*). \tag{17}$$

Let $\bar{y} = z^*_{L-1}$ and note $\bar{y} \in \mathcal{N}^{-1}(x)$. Let $\bar{z} \in \mathcal{Z}(y^*,\bar{y})$ be such that

$$\bar{z} = (z^*_1, z^*_2, \ldots, z^*_{L-1}).$$

We have

$$\begin{aligned}\eta.w^*(\bar{y}) + \eta.d(\bar{y},x) &\leq \eta.v^*(y^*) + \delta_{\eta.d}(y^*,\bar{y}) + \eta.d(\bar{y},x)\\ &= \eta.v^*(y^*) + \lambda_{\eta.d}(\bar{z}) + \eta.d(\bar{y},x)\\ &= \eta.v^*(y^*) + \lambda_{\eta.d}(z^*)\\ &= \eta.v^*(y^*) + \delta_{\eta.d}(y^*,x)\end{aligned} \tag{18}$$

The inequality is by (9). The first equality is because in order that z^* be a minimal-cost trajectory from y^* to x, $\bar{z}$ must be a minimal-cost trajectory from y^* to $\bar{y}$. The second equality results from z^* being the same as $\bar{z}$ with the commute from $\bar{y}$ to x appended. The final equality is by (17).

Because (16) holds for all $y \in \mathcal{N}^{-1}(x)$ and (18) holds for $\bar{y} \in N^{-1}(x)$ we know that

$$\eta.v^*(y^*) + \delta_{\eta.d}(y^*, x) = \min_{y \in \mathcal{N}^{-1}(x)} \left[\eta.w^*(y) + \eta.d(y, x)\right], \tag{19}$$

for the case when $y^* \neq x$.

The proof results can be summarized by

$$\begin{aligned}
\eta.w^*(x) &= \min_{y \in \mathcal{X}} \left[\eta.v^*(y) + \delta_{\eta.d}(y, x)\right] \\
&= \min \left\{ \eta.v^*(x), \min_{y \neq x} \left[\eta.v^*(y) + \delta_{\eta.d}(y, x)\right] \right\} \\
&= \min \left\{ \eta.v^*(x), \min_{y \in \mathcal{N}^{-1}(x)} \left[\eta.w^*(y) + \eta.d(y, x)\right] \right\}
\end{aligned}$$

The first equality is by (9), the second equality follows from (15), and the final equality follows from (19). ■

V. Algorithm

We propose Algorithm 1 to find the set of optimal meeting locations, i.e. $\Upsilon.p^*$ from (5). Assume `FindSolution`(ρ) has just been called. The algorithm first computes the value functions $\Upsilon.v$ and $\Upsilon.w$ for all nodes except ρ by calling the recursive function `FindValue` for each child of ρ. It then computes $\rho.v : \mathcal{X} \to \mathbb{R}$, which specifies the minimal meeting tree cost given the root meeting is located at each $x \in \mathcal{X}$. Next it assigns $\rho.p$ the minimal-cost meeting location. Finally, the algorithm computes the meeting locations $\Upsilon.p$ for all other nodes by calling the recursive function `FindMeetingLocation` for each child of ρ.

Input: ρ

1 **for** $k \leftarrow 1 : \rho.K$ **do** `FindValue`$(\rho.\kappa[k])$

2 $\rho.v \leftarrow \sum_{k=1}^{\rho.K} \rho.\kappa[k].w + \rho.c$

3 $\rho.p \leftarrow \arg\min_{x \in X} \rho.v(x)$

4 $\rho.z \leftarrow [\rho.p]$

5 **for** $k \leftarrow 1 : \rho.K$ **do** `FindMeetingLocation`$(\rho.\kappa[k], \rho.p)$

Algorithm 1: `FindSolution`

Input: η

1 **for** $k \leftarrow 1 : \eta.K$ **do** `FindValue`$(\eta.\kappa[k])$

2 $\eta.v \leftarrow \sum_{k=1}^{\eta.K} \eta.\kappa[k].w + \eta.c$

3 `FindCommuteValue`(η)

Algorithm 2: `FindValue`

Input: η

$\eta.w \leftarrow \eta.v$

$\mathcal{Q} \leftarrow \mathcal{X}$

while $\mathcal{Q} \neq \emptyset$ **do**

  $x^* \leftarrow \arg\min_{x \in \mathcal{Q}} \eta.w(x)$

  $\mathcal{Q} \leftarrow \mathcal{Q} \setminus \{x^*\}$

  foreach $\tilde{x} \in \mathcal{N}(x^*) \cap \mathcal{Q}$ **do**

    $\eta.w(\tilde{x}) \leftarrow \min_{\tilde{y} \in \mathcal{N}^{-1}(\tilde{x})} [\eta.w(\tilde{y}) + \eta.d(\tilde{y}, \tilde{x})]$

  end

end

Algorithm 3: `FindCommuteValue` (Dijkstra's)

Input: η,x

if $\eta.w(x) = \eta.v(x)$ **then**

  $\eta.p \leftarrow x$

  $\eta.z \leftarrow [\eta.p]$

  for $k \leftarrow 1 : \eta.K$ **do** `FindMeetingLocation`$(\eta.\kappa[k], \eta.p)$

else

  $y^* \leftarrow \arg\min_{y \in \mathcal{N}^{-1}(x)} [\eta.w(y) + \eta.d(y, x)]$

  `FindMeetingLocation`(η, y^*)

  $\eta.z \leftarrow [\eta.z, x]$

end

Algorithm 4: `FindMeetingLocation`

A. Correctness

Here we prove that Algorithm 1 results in an optimal set of meeting locations $\Upsilon.p^*$. We first prove lemmas regarding the results of `FindCommuteValue`, `FindValue`, and `FindMeetingLocation`.

Lemma 1: Let η be a meeting node in Υ. Assuming that $\eta.v = \eta.v^*$, a call to `FindCommuteValue`(η) terminates with $\eta.w = \eta.w^*$.

Proof: We introduce an additional state $\hat{x}$, such that $\hat{x} \notin \mathcal{X}$ and define the extended state space $\hat{\mathcal{X}} = \mathcal{X} \cup \{\hat{x}\}$. We then define the extended commuting cost funtion $\eta.\hat{d} : \hat{\mathcal{X}} \times \mathcal{X} \rightarrow \mathbb{R}^+$, similar to d but also including commuting costs from state $\hat{x}$ to all states in $\mathcal{X}$ given by the values $\eta.v$; that is,

$$\eta.\hat{d}(x, y) = \begin{cases} \eta.v(y), & \text{if } x = \hat{x}, \\ \eta.d(x, y), & \text{otherwise.} \end{cases}$$

Then, a call to `FindCommuteValue`(η) is the equivalent of running Dijkstra's algorithm [7] with initial state $\hat{x}$, which terminates after finding the minimum cost over all trajectories from $\hat{x}$ to any node $x \in \mathcal{X}$. In other words, a call to `FindCommuteValue`(η) terminates and we have for any $x \in \mathcal{X}$

$$\eta.w(x) = \delta_{\eta.d}(\hat{x}, x) = \min_{y \in \mathcal{X}} \left[\eta.v(y) + \delta_{\eta.d}(y, x)\right].$$

Therefore,

$$\eta.w(x) = \min_{y \in \mathcal{X}} \left[\eta.v^*(y) + \delta_{\eta.d}(y, x)\right] = \eta.w^*(x),$$

due to the assumption $\eta.v = \eta.v^*$ and (9). ■

Lemma 2: Let η be any node except ρ in Υ. A call to `FindValue`(η) terminates with $\tilde{\eta}.v = \tilde{\eta}.v^*$ and $\tilde{\eta}.w = \tilde{\eta}.w^*$for all nodes $\tilde{\eta}$ (including $\tilde{\eta} = \eta$) in the subtree rooted at η.

Proof: The following is an inductive proof showing that after executing `FindValue`(η), when η is a leaf $\eta.v = \eta.v^*$ and $\eta.w = \eta.w^*$; and for any η with children, if $\eta.\kappa[k].w = \eta.\kappa[k].w^*$ for all children, then $\eta.v = \eta.v^*$ and $\eta.w = \eta.w^*$.

Let η be a leaf in Υ. Call `FindValue`(η). Line 1 in Algorithm 2 terminates with no effect since $\eta.K = 0$. Line 2 sets $\eta.v = \eta.c$, so by (8), considering η has no children, $\eta.v = \eta.v^*$. After executing Line 3, by Lemma 1, we have $\eta.w = \eta.w^*$. Thus, a call to `FindValue`(η) terminates with $\tilde{\eta}.v = \tilde{\eta}.v^*$ and $\tilde{\eta}.w = \tilde{\eta}.w^*$.

Let η be any node except ρ in Υ with finite $\eta.K \geq 1$. Assume a call to `FindValue`(η) terminates with

$$\eta.\kappa[k].w = \eta.\kappa[k].w^* \tag{20}$$

for $1 \leq k \leq \eta.K$. Then the **for** loop at Line 1 terminates since $\eta.K$ is finite. Line 2 sets

$$\eta.v(x) \leftarrow \sum_{k=1}^{\eta.K} \eta.\kappa[k].w(x) + \eta.c(x),$$

so by (20) and (8), we have $\eta.v = \eta.v^*$. By Lemma 1, execution of Line 3 terminates with $\eta.w = \eta.w^*$. Thus, a call to `FindValue`(η) terminates with $\tilde{\eta}.v = \tilde{\eta}.v^*$ and $\tilde{\eta}.w = \tilde{\eta}.w^*$.

Now let η be any node except ρ in Υ. By induction on the finite node hierarchy of the subtree rooted at η, a call to `FindValue`(η) terminates with $\tilde{\eta}.v = \tilde{\eta}.v^*$ and $\tilde{\eta}.w = \tilde{\eta}.w^*$. ■

We make some further definitions for use in the remaining lemmas and Theorem 3 in this section. We say that the subtree rooted at η is *proper* and write $\phi(\eta) =$ `True` if for all nodes $\tilde{\eta}$ in the subtree rooted at η, $\tilde{\eta}.\kappa[k].z_L = \tilde{\eta}.p$ for $1 \leq k \leq \tilde{\eta}.K$ and $\tilde{\eta}.p = \tilde{\eta}.z_1$. In other words, $\phi(\eta) =$ `True` if all the meeting-to-meeting trajectories in the subtree rooted at η are connected to one another at the appropriate meeting locations. The function ϕ can be defined recursively as

$$\phi(\eta) := \bigwedge_{k=1}^{\eta.K} \left[\phi(\eta.\kappa[k]) \wedge (\eta.\kappa[k].z_L = \eta.p)\right] \wedge (\eta.p = \eta.z_1). \tag{21}$$

We also define a recursive function h that computes the total cost of all trajectories and meetings in the subtree rooted at node η.

$$h(\eta) = \sum_{k=1}^{\eta.K} h(\eta.\kappa[k]) + \eta.c(\eta.p) + \lambda_{\eta.d}(\eta.z), \tag{22}$$

Note that although h only takes a node η as a parameter, it is really a function of all meeting-to-meeting trajectories and meeting locations in the subtree rooted at η.

The following condition is used in Lemmas 3, 4, and 5, and Theorem 3. It is a precondition for correct execution of `FindMeetingLocation` and is shown in Theorem 3 to be a postcondition of the execution of Line 1 in Algorithm 1.

Condition 1: For all nodes η except ρ in Υ, $\eta.v = \eta.v^*$ and $\eta.w = \eta.w^*$.

The following condition is used in Lemmas 3 and 5. It is a precondition for the correct execution of the **then** block of Algorithm 4. It says roughly that given a node η, for all children $\eta.\kappa[k]$ and all states $x \in \mathcal{X}$, `FindMeetingLocation`$(\eta.\kappa[k], x)$ terminates with a correct subtree of total cost $\eta.\kappa[k].w(x)$.

Condition 2: Given a node η in Υ, for $1 \leq k \leq \eta.K$ and all $x \in \mathcal{X}$ a call to `FindMeetingLocation`$(\eta.\kappa[k], x)$ terminates with $\phi(\eta.\kappa[k]) =$ `True`, $\eta.\kappa[k].z_L = x$, and $h(\eta.\kappa[k]) = \eta.\kappa[k].w(x)$.

The following condition is used in Lemmas 4 and 5. It is a precondition for the correct execution of the **else** block of Algorithm 4. It says roughly that given a state $x \in \mathcal{X}$, for all states $\tilde{x}$ with lesser $\eta.w$, `FindMeetingLocation`$(\eta, \tilde{x})$ terminates with a correct subtree of total cost $\eta.w(\tilde{x})$.

Condition 3: Given a state $x \in \mathcal{X}$, for all $\tilde{x} \in \mathcal{X}$ such that $\eta.w(\tilde{x}) < \eta.w(x)$ a call to `FindMeetingLocation`$(\eta, \tilde{x})$ terminates with $\phi(\eta) =$ `True`, $\eta.z_L = \tilde{x}$, and $h(\eta) = \eta.w(\tilde{x})$.

The next two lemmas are used as inductive steps in the inductive argument of Lemma 5. The following lemma considers the recursion occurring in the **then** block of Algorithm 4.

Lemma 3: Let η be any node except ρ in Υ and $x \in \mathcal{X}$ such that

$$\eta.w(x) = \eta.v(x). \tag{23}$$

Let Conditions 1 and 2 hold. Then, a call to `FindMeetingLocation`(η, x) terminates with $\phi(\eta) =$ `True`, $\eta.z_L = x$, and $h(\eta) = \eta.w(x)$.

Proof: Let η be any node except ρ in Υ and $x \in \mathcal{X}$ such that (23) holds. Let Conditions 1 and 2 hold. Call `FindMeetingLocation`(η, x).

Because (23) holds, the **then** block in Algorithm 4 is entered. After executing Line 2,

$$\eta.p = x. \tag{24}$$

After executing Line 3,

$$\eta.z_1 = \eta.z_L = [\eta.p] = [x]. \tag{25}$$

Due to Condition 2, for $1 \leq k \leq \eta.K$ the call to `FindMeetingLocation`$(\eta.\kappa[k], \eta.p)$ terminates. Since the **for** loop iterates a finite $\eta.K$ times, the call to `FindMeetingLocation`(η, x) terminates. Also due to the Condition 2, for $1 \leq k \leq \eta.K$, we have $\phi(\eta.\kappa[k]) =$ `True` and $\eta.\kappa[k].z_L = \eta.p$. As a result and since $\eta.z_1 = \eta.p$, we have

$\phi(\eta) = \texttt{True}$ by (21). Furthermore, we have

$$\begin{aligned} h(\eta) &= \sum_{k=1}^{\eta.K} h(\eta.\kappa[k]) + \eta.c(\eta.p) \\ &= \sum_{k=1}^{\eta.K} \eta.\kappa[k].w(\eta.p) + \eta.c(\eta.p) \\ &= \eta.v(\eta.p) = \eta.w(x) \end{aligned}$$

The first equality is by (22) because $\lambda_{\eta.d}(\eta.z) = \lambda_{\eta.d}([x]) = 0$. The second equality is because $h(\eta.\kappa[k]) = \eta.\kappa[k].w(\eta.p)$ for $1 \leq k \leq \eta.K$, due to Condition 2. The third equality is by Condition 1 and (8). The final equality is by (24) and (23). Therefore, a call to $\texttt{FindMeetingLocation}(\eta, x)$ terminates with $\phi(\eta) = \texttt{True}$, $\eta.z_L = x$, and $h(\eta) = \eta.w(x)$. ■

The following lemma considers the recursion occurring in the **else** block of Algorithm 4.

Lemma 4: Let η be any node except ρ in Υ and $x \in \mathcal{X}$ such that (23) fails to hold. Let Conditions 1 and 3 hold. Then, a call to $\texttt{FindMeetingLocation}(\eta, x)$ terminates with $\phi(\eta) = \texttt{True}$, $\eta.z_L = x$, and $h(\eta) = \eta.w(x)$.

Proof: Let η be any node except ρ in Υ and $x \in \mathcal{X}$ such that (23) fails to hold. Let Conditions 1 and 3 hold. Call $\texttt{FindMeetingLocation}(\eta, x)$.

Because (23) fails to hold, the **else** block is entered. After executing Line 6, we know that

$$\eta.w(x) = \eta.w(y^*) + \eta.d(y^*, x) \tag{26}$$

by Condition 1, (13) and the fact that (23) does not hold. Since $\eta.d(y^*, x)$ is positive we have $\eta.w(y^*) < \eta.w(x)$. Thus, by Condition 3, after executing $\texttt{FindMeetingLocation}(\eta, y^*)$, FindMeetingLocation terminates with $\phi(\eta) = \texttt{True}$, $\eta.z_L = y^*$, and $h(\eta) = \eta.w(y^*)$. Executing Line 8 extends $\eta.z$ by appending x. It is clear from (21) that $\phi(\eta) = \texttt{True}$ continues to hold. Also, after Line 8, $\eta.z_L = x$. Finally, in executing Line 8, $h(\eta)$ increases by $\eta.d(y^*, x)$. Thus, due to Condition 3 and (26) we have

$$h(\eta) = \eta.w(y^*) + \eta.d(y^*, x) = \eta.w(x).$$

Therefore, a call to $\texttt{FindMeetingLocation}(\eta, x)$ terminates with $\phi(\eta) = \texttt{True}$, $\eta.z_L = x$, and $h(\eta) = \eta.w(x)$.

■

Lemma 5: Let Condition 1 hold. Then, for all nodes η except ρ in Υ and for all $x \in \mathcal{X}$, a call to $\texttt{FindMeetingLocation}(\eta, x)$ terminates with $\phi(\eta) = \texttt{True}$, $\eta.z_L = x$, and $h(\eta) = \eta.w(x)$.

Proof: Let Condition 1 hold. The following is an inductive proof.

First, we consider the base case where η is a leaf and $\check{x}$ is as defined in (11). Because η has no children, Condition 2 holds vacuously. Furthermore, by Corollary 1, we have $\eta.w(\check{x}) = \eta.v(\check{x})$. Thus by Lemma 3, a call to $\texttt{FindMeetingLocation}(\eta, x)$ terminates with $\phi(\eta) = \texttt{True}$, $\eta.z_L = \check{x}$, and $h(\eta) = \eta.w(\check{x})$.

Second, we use induction on $x \in \mathcal{X}$ in order of increasing $\eta.w(x)$. We assume Condition 3 for the inductive step. When (23) holds, by Lemma 3, a call to $\texttt{FindMeetingLocation}(\eta, x)$ terminates with $\phi(\eta) = \texttt{True}$, $\eta.z_L = x$, and $h(\eta) = \eta.w(x)$. Alternatively, when (23) does not hold, by Condition 3 and Lemma 4, a call to

`FindMeetingLocation`(η, x) terminates with $\phi(\eta) = \texttt{True}$, $\eta.z_L = x$, and $h(\eta) = \eta.w(x)$. The recursion is finite since $|\mathcal{X}|$ is finite and $\eta.w(x)$ decreases for each recursive call to `FindMeetingLocation`(η, x). Thus, by induction, we have that for all $x \in \mathcal{X}$ a call to `FindMeetingLocation`(η, x) terminates with $\phi(\eta) = \texttt{True}$, $\eta.z_L = x$, and $h(\eta) = \eta.w(x)$.

Next, we consider the case where η has children and $\check{x}$ is as defined in (11). We assume Condition 2 for the inductive step. By Corollary 1, we have $\eta.w(\check{x}) = \eta.v(\check{x})$. Thus by Lemma 3, a call to `FindMeetingLocation`(η, x) terminates with $\phi(\eta) = \texttt{True}$, $\eta.z_L = \check{x}$, and $h(\eta) = \eta.w(\check{x})$. Last, we consider the case where η has children and Condition 3 holds. Using an inductive argument identical to that above for the case where η is a leaf, we have that for all $x \in \mathcal{X}$ a call to `FindMeetingLocation`(η, x) terminates with $\phi(\eta) = \texttt{True}$, $\eta.z_L = x$, and $h(\eta) = \eta.w(x)$.

Therefore, by induction on the finite node hierarchy of the subtree rooted at η, for all nodes η except ρ in Υ and for all $x \in \mathcal{X}$, a call to `FindMeetingLocation`(η, x) terminates with $\phi(\eta) = \texttt{True}$, $\eta.z_L = x$, and $h(\eta) = \eta.w(x)$. ■

Theorem 3: A call to `FindSolution`(ρ), terminates with $\phi(\eta) = \texttt{True}$, $\Upsilon.p = \Upsilon.p^*$, and $\Upsilon.z = \Upsilon.z^*$.

Proof: Begin executing `FindSolution`(ρ). After executing Line 1 in Algorithm 1, Condition 1 holds by Lemma 2. In particular, by Condition 1 we have $\rho.\kappa[k].v = \rho.\kappa[k].v^*$ and $\rho.\kappa[k].w = \rho.\kappa[k].w^*$ for $1 \leq k \leq \rho.K$. After executing Line 2, by (8) we have

$$\rho.v = \rho.v^*. \tag{27}$$

Following the execution of Line 3, we have

$$\begin{aligned} \rho.v(\rho.p) &= \min_{x \in \mathcal{X}} \rho.v(x) \\ &= \min_{x \in \mathcal{X}} \rho.v^*(x) \\ &= \min_{x \in \mathcal{X}} \left[\min_{\Upsilon.p,\ \rho.p = x} f(\rho) \right] \\ &= \min_{\Upsilon.p} f(\rho). \end{aligned} \tag{28}$$

The first equality is from Line 3, since $\rho.p$ minimizes $\rho.v$. The second equality is by (27) and the third equality is by (6). The final equality holds because minimizing $f(\rho)$ for each possible $\rho.p = x \in \mathcal{X}$, and then minimizing over all $\rho.p = x \in \mathcal{X}$ is equivalent to minimizing $f(\rho)$ over all possible $\Upsilon.p$.

After executing Line 4 we have $\rho.z_1 = \rho.z_L = [\rho.p]$. Since Condition 1 holds, by Lemma 5, execution of Line 5 terminates and for $1 \leq k \leq \eta.K$, we have $\phi(\rho.\kappa[k]) = \texttt{True}$, $\rho.\kappa[k].z_L = \rho.p$, and $h(\rho.\kappa[k]) = \rho.\kappa[k].w(\rho.p)$.

As a result and since $\rho.z_1 = \rho.p$, we have $\phi(\rho) = \texttt{True}$ by (21). Furthermore, we have

$$\begin{aligned} h(\rho) &= \sum_{k=1}^{\rho.K} h(\rho.\kappa[k]) + \rho.c(\rho.p) \\ &= \sum_{k=1}^{\rho.K} \rho.\kappa[k].w(\rho.p) + \rho.c(\rho.p) \\ &= \rho.v(\rho.p) = \min_{\Upsilon.p} f(\rho). \end{aligned} \tag{29}$$

The first equality is by (22) because $\lambda_{\rho.d}(\rho.z) = \lambda_{\rho.d}([\rho.p]) = 0$. The second equality is because $h(\rho.\kappa[k]) = \rho.\kappa[k].w(\rho.p)$, for $1 \leq k \leq \rho.K$. The third equality is by Condition 1, (27), and (8). The final equality is by (28).

Since $\phi(\rho) = \texttt{True}$ the meeting tree Υ is proper. Furthermore, the total cost $h(\rho)$ of the meeting tree Υ is optimal by (29). Therefore, $\Upsilon.p = \Upsilon.p^*$ and $\Upsilon.z = \Upsilon.z^*$. ■

We have not only proved that $\Upsilon.p = \Upsilon.p^*$, but in the process we have demonstrated that $\Upsilon.v = \Upsilon.v^*$, $\Upsilon.w = \Upsilon.w^*$, and $\Upsilon.z = \Upsilon.z^*$. Accordingly, in the sequel we consider $\Upsilon.(v, w, p, z)$ that result from the execution of Algorithm 1 to be optimal, i.e., $\Upsilon.(v, w, p, z) = \Upsilon.(v^*, w^*, p^*, z^*)$.

B. Complexity

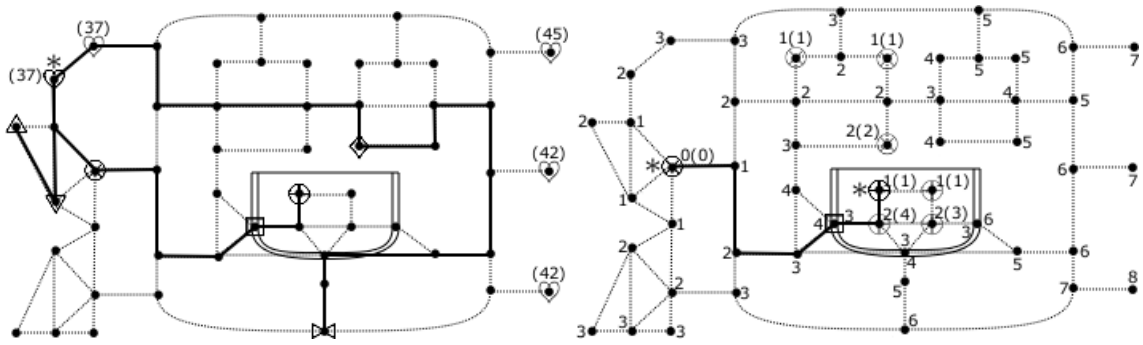
Algorithm 1 consists of a single leaves-to-root pass where at each meeting node $\eta.w$ is computed using `FindCommuteValue`, followed by a single root-to-leaves pass where at each meeting node $\eta.p$ is computed using `FindMeetingLocation`. We assume that $|\mathcal{N}(x)| < a$ and $|\mathcal{N}^{-1}(x)| < b$ for some constants a and b independent of $|\mathcal{X}|$ (e.g., if $\mathcal{X}$ is a Cartesian grid, then $a = b = 2^d$, where d is the number of dimensions, no matter what the resolution of the grid). In this case, `FindCommuteValue` can be implemented efficiently using Dijkstra's algorithm (i.e. Algorithm 3), which has complexity $\mathcal{O}(|\mathcal{X}||\mathcal{N}(x)| \log |\mathcal{X}|) = \mathcal{O}(|\mathcal{X}| \log |\mathcal{X}|)$ if the $\arg\min$ in Line 4 is implemented by maintaining a min-heap sorted on $\eta.w(x)$. On the other hand, the complexity of `FindMeetingLocation` is $\mathcal{O}(|\mathcal{X}||\mathcal{N}^{-1}(x)|) = \mathcal{O}(|\mathcal{X}|)$, since the meeting-to-meeting trajectory $\eta.z$ can pass through at most every node $x \in \mathcal{X}$ and at each node consider at most $|\mathcal{N}^{-1}(x)|$ neighbors to extend the trajectory. Consequently, the complexity of `FindCommuteValue` dominates that of `FindMeetingLocation`. Thus, the overall complexity of Algorithm 1 is $\mathcal{O}(M|\mathcal{X}| \log |\mathcal{X}|)$, where M is the number of nodes in Υ.

VI. EXAMPLES

We solve two example problems: The first is the discrete robot clinic consultation defined earlier, and the second is a continuous robot arm cargo transport which we describe below.

A. Patient Consultation

We solve the problem defined in Section III-B using Algorithm 1. Figure 2(a) shows the resulting minimal-cost solution tree including optimal meeting locations $\Upsilon.p^*$ and optimal meeting-to-meeting trajectories $\Upsilon.z^*$. After meeting the pharmacist, the nurse backtracks over its previous trajectory to its starting location and then on to the



(a) (b)

Fig. 2. (a) A minimal-cost solution tree for the patient consultation problem. Optimal meeting locations $\Upsilon.p^*$ are shown using the symbols defined in Figure 1(c). The *hold consultation* meeting values $\heartsuit.v^*$ are indicated in parentheses at each of the 5 possible locations. An optimal meeting location $\heartsuit.p^*$ is indicated with $*$. Corresponding optimal meeting-to-meeting trajectories $\Upsilon.z^*$ are indicated with thick solid lines. The pharmacy counter and back wall is indicated with thin solid lines. (b) Value functions and optimal trajectories of the *begin pharmacist* $\bigoplus$ and *begin nurse* $\bigotimes$ meeting nodes. The optimal meeting values $\eta.v^*$ are indicated in parentheses and the optimal meeting-plus-commute values $\eta.w^*$ have no parentheses. The values $\bigoplus.v^*$ and $\bigoplus.w^*$ are indicated inside the pharmacy counter while the values $\bigotimes.v^*$ and $\bigotimes.w^*$ are indicated outside. Optimal meeting locations $\bigoplus.p^*$ and $\bigotimes.p^*$ are indicated with $*$. The optimal *get meds* location $\square.p^*$ is indicated with $\square$.

consultation. The trajectories of the wheelchair (with patient) and the nurse overlap for one segment just before the optimal consultation location $\heartsuit.p^*$ is reached. Figure 2(a) shows that the root meeting location $\heartsuit.p$ is a location with minimal $\heartsuit.v(\heartsuit.p) = 37$, in accordance with the assignment of $\heartsuit.p$ in Line 3 of Algorithm 1. Note that the two consultation locations in the left-most consultation room both have a meeting value of 37 and it does not matter which is chosen.

We use Figure 2(b) to illustrate how Algorithm 1 calls Algorithms 2 and 3 to calculate $\eta.v^*$ and $\eta.w^*$, respectively. The figure indicates the value functions of two meeting nodes $\bigoplus$ (for nodes inside the pharmacy counter) and $\bigotimes$ (for nodes outside the pharmacy counter), each a leaf node which Algorithm 2 takes as its η parameter at the bottom of its recursive hierarchy. For η a leaf node, Line 2 assigns the meeting value $\eta.v = \eta.c$ (see the numbers in parentheses in Figure 2(b)). Then, in Line 3 the call to `FindCommuteValue`(η) computes for each reachable state the meeting-plus-commute value $\eta.w$ in order of increasing value (see the unparenthesized numbers in Figure 2(b)). In particular, consider the values for $\bigoplus$ within the counter in the figure computed by the call `FindCommuteValue`$(\bigoplus)$. For x one of the upper two states within the counter, $\bigoplus.w(x) = \bigoplus.v(x) = 1$, since it is cheaper to begin the pharamacist at x than it is to begin the pharmacist at some other state and commute to x. Alternatively, for x one of the lower two states within the counter, $\bigoplus.w(x) = 2 < \bigoplus.v(x)$, since it is cheaper to begin the pharamacist at the state directly above x and commute down to x. Finally, for x on the counter, the value $\bigoplus.w(x) = 3$ results from a commute from a neighboring state, since the pharmacist cannot start on the counter. Analogously, the meeting-plus-commute values $\bigotimes.w(x)$ for states x outside and on the counter are calculated by the call `FindCommuteValue`$(\bigotimes)$.

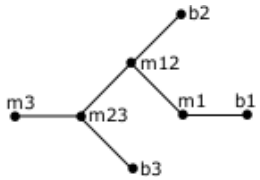
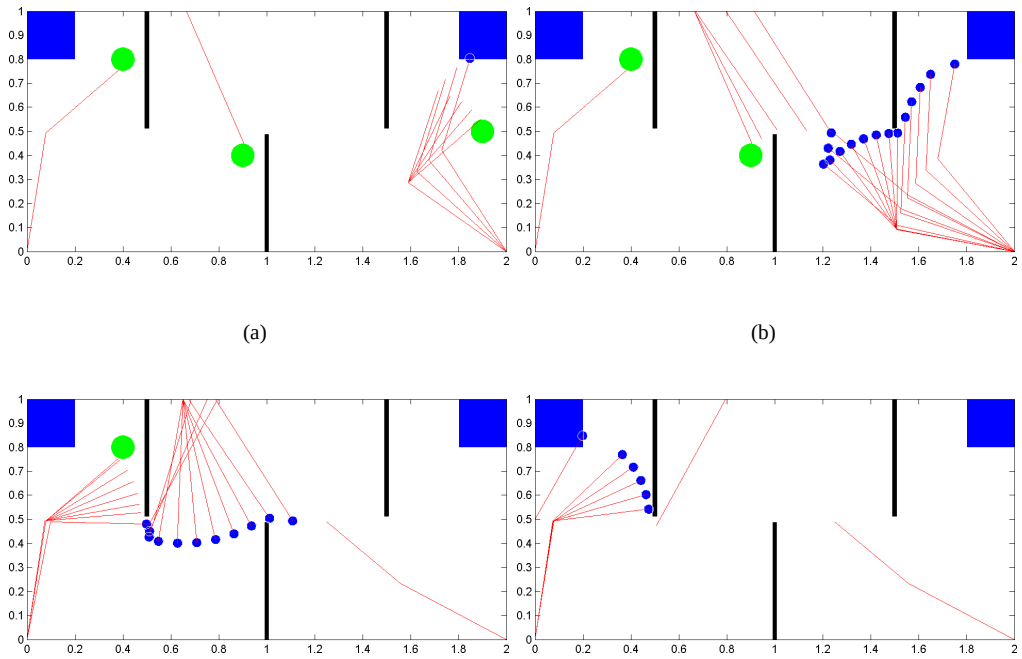


Fig. 3. Meeting tree for the robot arm problem. Begin robot i is indicated by bi. Pickup cargo for robot 1 is indicated by m1. Pass cargo from robot i to robot j is indicated by mij. Dropoff cargo for robot 3 is indicated by m3.

We also use Figure 2(b) to illustrate how Algorithm 1 later calls Algorithm 4 to calculate $\eta.z^*$ and $\eta.p^*$. Nodes $\bigoplus$ and $\bigotimes$ are leaf nodes, each of which is passed as the η parameter to Algorithm 4 at the bottom of its recursive hierarchy. For any meeting node η, the **while** loop in Algorithm 4 computes $\eta.z$ by following a steepest descent path on $\eta.w$ until a node x is reached such that $\eta.w(x) = \eta.v(x)$, with x being assigned to the meeting location $\eta.p$. In particular, consider the call to `FindMeetingLocation`$(\bigoplus, \square.p^*)$ with $\square.p^*$ indicated in the Figure. At first, $x = \square.p^*$, which is not an allowed *begin pharmacist* location (i.e., $\bigoplus.v(x) = \infty \neq \bigoplus.w(x)$), so the **while** loop condition is met. The only neighboring state is the lower-left state inside the counter so it is prepended to $\bigoplus.z$ in Line 8 of Algorithm 4. For the lower-left state, the meeting-plus-commute value is less than the meeting value, so the loop condition is met. The upper-left state inside the counter is the minimizing neighbor in Line 6, so it is prepended to $\bigoplus.z$. For this state, the meeting-plus-commute value equals the meeting value so the loop is terminated. The *begin pharmacist* location $\bigoplus.p$ is assigned the upper-left state. Analogously, $\bigotimes.z$ and $\bigotimes.p$ are calculated by the call `FindMeetingLocation`$(\bigotimes, \square.p^*)$.

B. Robot Arms

We solve a robot arm meeting problem using an extension of Algorithm 1 modified to handle the continuous state spaces of the robot arms. The problem involves three robot arms cooperating to transport cargo from a cargo pickup location in the top-right of the workspace to a cargo dropoff location in the top-left of the workspace. Robot 1 is attached to the lower-right corner of the workspace and has two rotational joints. The primary joint has an angular range of $\pi/2$ radians, such that the primary segment cannot swing out of the workspace, and the secondary joint has an angular range of 2π radians, but the secondary segment cannot swing through the primary segment. Robot 3 is attached to the lower-left corner of the workspace and otherwise has the same properties as robot 1. Robot 2 moves on a sliding joint along the top of the workspace. It also has a rotational joint that moves through an angle of π radians, such that the arm cannot swing out of the workspace. Each robot begins such that its end effector is in a circular starting area. Robot 1 must pick up the cargo and pass it to robot 2, then robot 2 must pass the cargo to robot 3, who drops off the cargo. For two robots to "meet," their end effectors must approach within a small neighborhood of one another. The goal is to find meeting locations and connecting state trajectories that minimize the cost of transporting the cargo across the workspace. The corresponding meeting tree is shown in Figure 3, and the resulting robot arm motions are depicted in Figure 4.



(a) (b)

(c) (d)

Fig. 4. Robot arm motions for a solution meeting tree. The sequence of figures is a time series with (a) showing the beginning of the robot arm motions and (d) showing the completion of the motions. The starting area for the end effector of each robot arm is indicated by a circle. The square box on the top-right/top-left is the pickup/dropoff location for the cargo. A small circle at the end effector of a moving robot arm indicates that the robot is currently carrying the cargo. (a) Robot 1 moves from its starting location to cargo pickup area. (b) Robot 1 moves from the cargo pickup box to meet robot 2, while robot 2 moves from its starting location to meet robot 1. (c) Robot 2 moves from its meeting with robot 1 to meet robot 3, while robot 3 moves from its starting location to meet robot 2. (d) Robot 3 moves from its meeting with robot 2 to the cargo dropoff area.

Each robot arm has 2 degrees of freedom, one for each of its joints. Consequently, each has a continuous 2-dimensional state space. Let

$$\zeta(t) = (\zeta_1(t), \zeta_2(t)), \text{ for } t_0 < t < t_f,$$

be a continuous trajectory through the 2D state space of a robot, where t_0 is the starting time and t_f is the finishing time. The cost of a robot state trajectory is given by the continuous analogue of (1)

$$\lambda(\zeta(\cdot)) = \int_{t_0}^{t_f} \|\dot{\zeta}(t)\|_1 dt = \int_{t_0}^{t_f} \left[|\dot{\zeta}_1(t)| + |\dot{\zeta}_2(t)| \right] dt. \tag{30}$$

However, the cost of a robot arm passing through an obstacle is considered infinite. The total cost of transporting the cargo is the sum of the costs of the robot state trajectories. There are no meeting costs incurred when two robots meet (meetings mij in Figure 3). Also, there is no cost for a robot to begin (meetings bi) within its circular starting area but there is an infinite cost for a robot to begin outside its starting area. Finally, there is no cost for cargo pickup (meeting m1) or dropoff (meeting m3) within the respective pickup or dropoff area but there is an

infinite cost for cargo pickup or dropoff outside the appropriate area.

We modify Algorithm 1 to solve the continuous robot arm problem. Since the robot arms are operating within a continuous state space we use the Fast Marching Method in place of Algorithm 3 (Dijkstra's Algorithm), which is designed to compute value functions on discrete graphs. More specifically, for each meeting node we approximately solve the Eikonal equation

$$\|\nabla w(x)\|_\infty = 1,$$

on a discretized grid of the robot state space. The reasons for solving such an Eikonal equation and methods for doing so are discussed in more depth in [2].

We use (30) to measure the cost of a robot trajectory in the robot state space, but the occurance of a meeting depends on the end effector position in the workspace. For this reason, we modify Algorithm 2 to employ two grids for each meeting node: a workspace grid for computing meeting values $\eta.v$ and a robot state grid for computing meeting-plus-commute values $\eta.w$. For the solution illustrated below we use a workspace grid of 401×201 nodes, a state grid of 401×101 nodes for robot 1 and 3, and a state grid of 201×201 nodes for robot 2. The meeting values are computed on the workspace grid using the same formula as in Line 2, but beforehand the child meeting-plus-commute values must be mapped from the child robot state grids onto the workspace grid. Such mapping is done using kinematics to determine the end effector position corresponding to the robot state and then taking the nearest workspace grid node to the end effector position. Next, the meeting values from the workspace grid must be mapped to the state grid before `FindCommuteValue`(η) is called in Line 3 to compute the meeting-plus-commute values. We avoid using inverse kinematics without changing the asymptotic complexity of the algorithm by iterating though all state grid nodes and using kinematics to determine the nearest workspace node in the same manner as above.

Some modifications are also required for Algorithm 4. First, the **while** loop should be replaced by an ODE solver that computes a steepest descent trajectory on an interpolated meeting-plus-commute value function in the robot state space. Second, the loop condition must compare the interpolated meeting-plus-commute value to the interpolated meeting value at the kinematically-determined end effector location in the workspace.

VII. Conclusion

We have specified a class of multi-location robot rendezvous problems for which dynamic programming can be used to find optimal solutions. An appropriate dynamic programming principle has been presented and used to construct an efficient algorithm. The applicability of the algorithm has been demonstrated on a discrete robotic clinic example and the algorithm has been modified slightly and applied to a continuous robot arm example.

There are several ways this work could be extented. Firstly, in this paper we have defined the cost of a meeting tree to be a sum of costs of its constituent trajectories and meetings. However, the summation in (4) could be replaced with any monotone function without compromising the dynamic programming principle stated in Theorem 1. For example, one could use a maximum in place of the summation to solve a minimal-time multi-location robot rendezvous problem. Secondly, the modified algorithm for solving the continuous rendezvous problem can be

examined with more rigour. Lastly, some computed meeting-plus-commute (and meeting) values may be so large that they can be ruled-out as possible contributors to the problem solution. It may be possible to increase algorithm efficiency by computing meeting-plus-commute (and meeting) values at states only if they may have influence.

ACKNOWLEDGMENT

This work was supported by a Discovery grant from the National Sciences and Engineering Research Council of Canada.

REFERENCES

[1] K. Alton and I. Mitchell, "Efficient optimal multi-location robot rendezvous," submitted to ICRA 2008.

[2] ——, "Optimal path planning under different norms in continuous state spaces," in *Proceedings of the International Conference on Robotics and Automation*, 2006, pp. 866–872.

[3] Y. Litus, R. T. Vaughan, and P. Zebrowski, "The frugal feeding problem: Energy-efficient, multi-robot, multi-place rendezvous," in *Proceedings of IEEE ICRA*, 2007, pp. 27–32.

[4] J.-C. Latombe, *Robot Motion Planning*. Boston: Kluwer Academic Publishers, 1981.

[5] H. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations*. Cambridge, Massachusetts and London, England: The MIT Press, 2005.

[6] S. M. Lavalle, *Planning Algorithms*. Cambridge University Press, 2006.

[7] E. W. Dijkstra, "A note on two problems in connection with graphs," *Numer. Math. 1*, pp. 269–271, 1959.

[8] D. P. Bertsekas, *Dynamic Programming and Optimal Control: Second Edition*. Belmont, Massachusetts: Athena Scientific, 2000.

[9] Y. Kupitz and H. Martini, "Geometric aspects of the generalized Fermat-Toricelli problem," *Bolyai Society Mathematical Studies*, vol. 6, pp. 55–129, 1997.

[10] G. Sahina and H. Sralb, "A review of hierarchical facility location models," *Computers and Operations Research*, vol. 34, no. 8, pp. 2310–2331, 2007.