Proceedings of the
47th IEEE Conference on Decision and Control
Cancun, Mexico, Dec. 9-11, 2008

Efficient Dynamic Programming for Optimal Multi-Location Robot Rendezvous

Ken Alton and Ian M. Mitchell
Department of Computer Science
University of British Columbia
Vancouver, BC, V6T 1Z4, Canada
{kalton,mitchell}@cs.ubc.ca

Abstract— **We present an efficient dynamic programming 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 continuous robot arm problem demonstrates the algorithm's practicality.**

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. 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 dynamic programming (DP) solution. More specifically, we examine optimal coordinated multi-robot multi-location rendezvous, an extension of the frugal feeding problem considered in [1]. In a multiple-robot scenario it is often useful for robots to meet to exchange fuel, cargo, and/or information.

Given a hierarchical structure that describes which robots are to meet and which robots are to continue on to future meetings, our DP 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 DP 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 a continuous problem involving robotic arms cooperating to deliver some cargo from a source to a destination through a workspace with obstacles. This robot arm problem is a form of hybrid system because of the continuous arm dynamics and discrete meeting events.

The main contribution of this paper is the presentation of a tree (or outer) DP principle to find optimal solutions to a broad category of hierarchical multi-location robot rendezvous problems. Furthermore, we show that for certain structures of the robot state space, a state (or inner) DP principle holds allowing commute costs for individual robots to be computed efficiently. An important property of both the tree and state DP principles is that no calculation is done in a state space with dimension greater than that of the individual robots.

We use the application of multi-location robot rendezvous as a concrete stand-in for any kind of optimal meeting problem with a fixed tree structure. Another such application might be the optimization of an industrial supply chain with a fixed tree structure. Moreover, the words meeting or supply chain seem to imply a temporal flow from leaves to root in the tree. However, our analysis also applies to dispersion or distribution problems with the opposite temporal flow or problems with no clear temporal character as long as they have the appropriate tree structure.

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 tree DP principle can be used to efficiently compute costs of potential solutions. Section V presents two special cases of the state space and resulting state DP principles that allow well-known DP algorithms to be used. Finally, in Section VI we use the algorithm to solve a continuous robot-arm rendezvous problem.

II. Related Work

Research in robot path planning is considered a central endeavor in the development of mobile robotics. Approaches to robot path planning are diverse and include potential function methods, sampling-based methods, trajectory planning methods and combinations of these [2], [3]. Most related to this paper are the DP algorithms for solving shortest path problems on grids [4], [5]; however, most path planning research, including that described in the above references, focuses on single robots.

In this paper, we investigate a DP 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 [6]. Given a set of vertices, the Steiner tree problem is to find the lowest cost tree connecting these vertices [7]. The hierarchical facility location problem [8]

978-1-4244-3124-3/08/$25.00 ©2008 IEEE

involves finding the location of facilities of several levels to serve customers most efficiently. Both the Steiner and facilities problems are more difficult than the rendezvous problem because the combinatorial aspect of determining the tree structure must be solved in addition to finding the optimal locations of intermediate nodes (i.e. meeting nodes) in the tree. For the rendezvous problem, we assume that this tree structure already exists and develop an efficient algorithm to optimize the location of meetings.

This work was motivated by [1], 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 any monotone meeting cost aggregation function, and show that algorithmic 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. This continuous robot meeting problem is an example of a hybrid system. The fast marching method, a DP algorithm, was used to find an optimal path through a hybrid system in [9]. Our dynamic programming algorithm for the continuous case solves a distinct hybrid problem involving a hierarchy of meetings.

III. Problem Description

A robot meeting involves one or more robots colocating at a state within a state 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 determining the meeting tree, which may be required for a general hierarchical facility location problem [8] or the Steiner tree problem. We wish to minimize, over all possible meeting locations, the total cost of a given meeting tree. The total cost of a meeting tree considers the cost of robot commutes between meetings as well as the cost of the meetings themselves. The avoidance of collisions between the robots during the commutes is not considered.

The first part of the problem instance definition is a *meeting tree*. Let there be a set of meeting tree nodes Υ. The term *node* or *meeting* may be used to refer to a meeting tree node $\eta \in \Upsilon$. We let $\rho \in \Upsilon$ be the root node in the meeting tree. The function $K : \Upsilon \to \{0\} \cup \mathbb{Z}^+$ specifies the number of children of each node $\eta \in \Upsilon$. The function $\kappa : \Upsilon \times \mathbb{Z}^+ \to \Upsilon$ specifies the children of each node $\eta \in \Upsilon$.

[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.

symbol	type	description
Υ	set of nodes	meeting tree node set
$K(\eta)$	$\Upsilon \to \{0\} \cup \mathbb{Z}^+$	number of children
$\kappa(\eta, k)$	$\Upsilon \times \mathbb{Z}^+ \to \Upsilon$	children
$\mathcal{X}$	set of states	state space
$\xi(\eta, x, \boldsymbol{\omega})$	$\Upsilon \times \mathcal{X} \times \mathbb{R}^{K(\eta)} \to \mathbb{R}$	meeting cost aggregation
$\delta(\eta, x, y)$	$\Upsilon \times \mathcal{X}^2 \to \{0\} \cup \mathbb{R}^+$	commuting cost
$p(\eta)$	$\Upsilon \to \mathcal{X}$	meeting location
$f(\eta)$	$\Upsilon \to \mathbb{R}$	subtree cost
$g(\eta, x)$	$\Upsilon \times \mathcal{X} \to \mathbb{R}$	subtree-plus-commute cost
$p^*(\eta)$	$\Upsilon \to \mathcal{X}$	optimal meeting location
$v(\eta, x)$	$\Upsilon \times \mathcal{X} \to \mathbb{R}$	optimal subtree cost
$w(\eta, x)$	$\Upsilon \times \mathcal{X} \to \mathbb{R}$	optimal subtree-plus-commute cost

TABLE I

SYMBOLS: THE UPPER BLOCK INCLUDES SYMBOLS USED TO DEFINE A PROBLEM INSTANCE. THE MIDDLE BLOCK INCLUDES SYMBOLS USED TO DEFINE A SOLUTION. THE LOWER BLOCK INCLUDES SYMBOLS USED TO COMPUTE AN OPTIMAL SOLUTION (SEE SECTION IV).

For convenience, we also define a function $\iota : \Upsilon \to \Upsilon$ that specifies the parent of each node $\eta \in \Upsilon$. The expression $\iota(\rho)$ is considered undefined.

The second part of a problem instance definition is the *state space* $\mathcal{X}$. The term *state* or *location* may be used to refer to an element $x \in \mathcal{X}$.

The last part of the problem instance definition specifies the costs of components of a potential meeting tree solution. Define the *commuting cost function* $\delta : \Upsilon \times \mathcal{X}^2 \to \{0\} \cup \mathbb{R}^+$. The expression $\delta(\eta, x, y)$ is the cost of commuting from $x \in \mathcal{X}$ to $y \in \mathcal{X}$ after the meeting $\eta \in \Upsilon$. Let $\delta(\eta, x, y) = 0$ if $x = y$ and $\delta(\eta, x, y) > 0$ if $x \neq y$. If $\delta(\eta, x, y) = \infty$ then it is not possible to commute from x to y. Define the *meeting cost aggregation function* $\xi : \Upsilon \times \mathcal{X} \times \mathbb{R}^K \to \mathbb{R}$. The expression $\xi(\eta, x, \boldsymbol{\omega})$ aggregates the costs $\boldsymbol{\omega} \in \mathbb{R}^{K(\eta)}$ of arriving at meeting $\eta \in \Upsilon$ at location $x \in \mathcal{X}$. We assume this function to be nondecreasing in each element ω_k of $\boldsymbol{\omega}$. If $\xi(\eta, x, \boldsymbol{\omega}) = \infty$ for all $\boldsymbol{\omega}$ then the meeting η cannot take place at location x. Useful examples of ξ include

$$\xi(\eta, x, \boldsymbol{\omega}) = \sum_{k=1}^{K} \omega_k + \gamma(\eta, x) \tag{1}$$

and

$$\xi(\eta, x, \boldsymbol{\omega}) = \max_{1 \leq k \leq K} \omega_k + \gamma(\eta, x),$$

where $\gamma(\eta, x)$ is the cost of meeting η being located at $x \in \mathcal{X}$. The $\sum$ definition is applicable for the problem of minimizing the sum of all commuting costs and meeting costs for a meeting tree, which is essentially what is done in [1]. The $\max$ definition is applicable for minimizing the critical path in a meeting tree; for example, minimizing the time elapsed before the root meeting is completed.

A potential solution to a problem instance is given by a *meeting location function* $p : \Upsilon \to \mathcal{X}$. The expression $p(\eta)$ is the location of meeting $\eta \in \Upsilon$. The total cost of a potential solution is defined recursively in terms of the commuting cost function and meeting cost aggregation function. The recursive definition includes a *subtree cost function* $f : \Upsilon \to$

$\mathbb{R}$ and a *subtree-plus-commute cost function* $g : \Upsilon \times \mathcal{X} \to \mathbb{R}$. The expression $f(\eta)$ is the total cost of the meeting subtree rooted at node η, while the expression $g(\eta, x)$ is the cost of a meeting subtree rooted at node η in addition to the commuting cost from the meeting location $p(\eta)$ to the state $x \in \mathcal{X}$. The combined definition of functions f and g is

$$f(\eta) = \xi\left(\eta, p(\eta), [g(\kappa(\eta,k), p(\eta))]_{k=1}^{K(\eta)}\right) \quad (2a)$$

$$g(\eta, x) = f(\eta) + \delta(\eta, p(\eta), x), \quad (2b)$$

where

$$[\omega_k]_{k=1}^{K} = \begin{bmatrix} \omega_1 \\ \omega_2 \\ \vdots \\ \omega_K \end{bmatrix} = \boldsymbol{\omega} \in \mathbb{R}^K.$$

Note that f and g take a single node η as a parameter, but are really functions of all locations p of the meetings in the subtree rooted at η. The subtree cost $f(\eta)$ is defined as an aggregation ξ of *subtree-plus-commute* costs $g(\kappa(\eta,k), p(\eta))$, each associated with child $\kappa(\eta, k)$. In turn, each cost $g(\kappa(\eta,k), p(\eta))$ is the sum of the cost $f(\kappa(\eta,k))$ of the subtree rooted at $\kappa(\eta,k)$ and the commuting cost $\delta(\kappa(\eta,k), p(\kappa(\eta,k)), p(\eta))$ from the child meeting location to the current meeting location. The base case for the recursion occurs when η is a child since

$$f(\eta) = \xi\left(\eta, p(\eta), [g(\kappa(\eta,k), p(\eta))]_{k=1}^{0}\right) = \xi(\eta, p(\eta), []). \quad (3)$$

An optimal solution to a problem instance is a meeting location function p^* which minimizes the total cost of the meeting tree:

$$p^* \in \arg\min_{p} f(\rho). \quad (4)$$

Table I provides a summary of symbols for defining a problem instance and solution.

IV. Tree Dynamic Programming

Define the *optimal subtree cost function* $v : \Upsilon \times \mathcal{X} \to \mathbb{R}$ as

$$v(\eta, x) = \min_{p,\ p(\eta)=x} f(\eta). \quad (5)$$

In other words, v specifies the minimal total cost of the subtree rooted at η subject to the condition that the meeting η is located at x. Define the *optimal subtree-plus-commute cost function* $w : \Upsilon \times \mathcal{X} \to \mathbb{R}$ as

$$w(\eta, x) = \min_{p} g(\eta, x). \quad (6)$$

In other words, $w(\eta, x)$ specifies the minimal total cost of the subtree rooted at η and the commuting cost from the meeting location $p(\eta)$ to x. Because of the recursive definition of f and g in (2), $v(\eta, x)$ and $w(\eta, x)$ do not depend on any ancestor nodes of η.

We observe two important properties of the meeting tree problem that allow an optimal solution p^* to be found efficiently. The first observation is that the functions v and w can be computed in a single pass through the meeting tree from the leaves towards the root, as shown in Theorem 1. The second observation is that the optimal meeting locations p^* can be computed in a single pass through the meeting tree from the root towards the leaves, given that we already know the value function v, as shown in Theorem 2.

A. Theory

The following theorem establishes a DP principle for v and w. It shows how v and w can be computed recursively. The theorem defines the optimal subtree cost $v(\eta, x)$ of node η at location x in terms of the children's optimal subtree-plus-commute costs $w(\kappa(\eta,k), x)$ at the same location. In turn, the theorem defines the optimal subtree-plus-commute costs $w(\eta, x)$ of node η at location x in terms of the optimal subtree costs $v(\eta, y)$ at each location y.

Theorem 1: The optimal subtree cost function v satisfies for all nodes $\eta \in \Upsilon$ and for all $x \in \mathcal{X}$

$$v(\eta, x) = \xi\left(\eta, x, [w(\kappa(\eta,k), x)]_{k=1}^{K(\eta)}\right). \quad (7)$$

Furthermore, the optimal subtree-plus-commute cost function w satisfies for all nodes $\eta \in \Upsilon$ such that $\eta \neq \rho$ and for all $x \in \mathcal{X}$

$$w(\eta, x) = \min_{y \in \mathcal{X}} \left[v(\eta, y) + \delta(\eta, y, x)\right]. \quad (8)$$

Note that to keep the paper concise the proofs in this section have been omitted, but they can be found in [10]. The proof first shows that (7) holds for the case where η is a leaf. It then shows that (7) holds for the alternative case where η has children. Finally, it shows that (8) holds for all nodes η other than ρ.

The following theorem shows how p^* can be computed recursively with $p^*(\rho)$ as the base case, given that v is already known. The theorem defines the optimal root meeting location $p^*(\rho)$ as the location that minimizes over all locations $x \in \mathcal{X}$ the optimal subtree cost $v(\rho, x)$. The theorem then defines the optimal meeting locations $p^*(\eta)$ for $\eta \neq \rho$ in terms of optimal subtree costs $v(\eta, y)$ at each location $y \in \mathcal{X}$ and the parent's optimal meeting location $p^*(\iota(\eta))$.

Theorem 2: Let $p^* : \Upsilon \to \mathcal{X}$ be a solution to (4). For the root meeting ρ, the following holds:

$$p^*(\rho) \in \operatorname*{argmin}_{x \in \mathcal{X}} v(\rho, x). \quad (9)$$

Furthermore, for all nodes $\eta \in \Upsilon$ such that $\eta \neq \rho$, the following holds:

$$p^*(\eta) \in \operatorname*{argmin}_{y \in \mathcal{X}} \left[v(\eta, y) + \delta(\eta, y, p^*(\iota(\eta)))\right]. \quad (10)$$

The proof assumes that (9) is false and arrives at a contradiction. A similar strategy is used to prove (10).

B. Algorithm Complexity

A DP algorithm based on the results of Theorems 1 and 2 can be used to find an optimal meeting location function p^*. The algorithm first computes in a leaves-to-root pass the value functions v and w for all nodes $\eta \in \Upsilon$ such that $\eta \neq \rho$ using (7) and (8). It then computes $v(\rho, x)$ using (7). Next it computes $p^*(\rho)$ using (9). Finally, the algorithm computes in a root-to-leaves pass the meeting locations $p^*(\eta)$ for each $\eta \in \Upsilon$ such that $\eta \neq \rho$ using (10).

We assume that the complexity of evaluating $\xi(\eta, x, \boldsymbol{\omega})$ or $\delta(\eta, y, x)$ is $\mathcal{O}(1)$. For a single $\eta \in \Upsilon$ and all $x \in \mathcal{X}$, the complexity of computing $v(\eta, x)$ using (7) is $\mathcal{O}(|\mathcal{X}|)$, while the complexity of computing $w(\eta, x)$ using (8) involves a minimization over all states $y \in \mathcal{X}$ and so is $\mathcal{O}(|\mathcal{X}|^2)$. On the other hand, for a single node η the complexity of computing $p^*(\eta)$ using (10) involves a minimization over all states $y \in \mathcal{X}$ and so is $\mathcal{O}(|\mathcal{X}|)$. Since there are $|\Upsilon|$ nodes and v and w are computed in a single leaves-to-root pass of the meeting tree, while p^* is computed in a single root-to-leaves pass of the meeting tree, the complexity of computing w dominates the other costs and the overall complexity of the algorithm is $\mathcal{O}(|\Upsilon||\mathcal{X}|^2)$. In the next section we examine how w can be computed more efficiently in order to improve the overall complexity in cases where the state space $\mathcal{X}$ contains additional structure.

V. State Dynamic Programming

We examine two special cases of the problem formulated in Section III. In the first case, the state space $\mathcal{X}$ is the set of nodes of a discrete spatial graph and the cost of commuting between two neighboring states is specified. In the second case, $\mathcal{X}$ is a continuous state space and a limit on the speed of commuting in each direction is specified. In both these cases instead of specifying δ directly, we exploit the structure of the state space and only specify the commuting costs locally. We then define δ using a state DP principle, distinct from the tree DP principle of Theorem 1. This state DP principle can be used to compute w for a single node η using a DP algorithm that is more efficient than applying (8) directly.

Computing $w(\eta, x)$ for any node $\eta \in \Upsilon$ and state $x \in \mathcal{X}$ can be viewed as an optimal starting problem. We optimize over starting states y as well as over possible trajectories to determine the optimal subtree-plus-commute cost $w(\eta, x)$ from optimal subtree costs $v(\eta, y)$. Let ζ be a trajectory through $\mathcal{X}$ and let $\lambda(\zeta)$ be the total cost of traversing trajectory ζ. We have

$$\delta(\eta, x, y) = \min_{\zeta \in \mathcal{Z}(x,y)} \lambda(\zeta), \tag{11}$$

where $\mathcal{Z}(x, y)$ is the set of all trajectories that begin at x and end at y. Then from (8)

$$w(\eta, x) = \min_{y \in \mathcal{X}} \left[v(\eta, y) + \min_{\zeta \in \mathcal{Z}(y,x)} \lambda(\zeta) \right]. \tag{12}$$

We examine for each special case a state DP principle based on this expression. Note that the dynamic programming is done for each robot independently.

A. Discrete State Space

Let $\mathcal{X}$ be the finite set of nodes of a directed spatial graph. In this case, Dijkstra's algorithm can be used to compute $w(\eta, x)$ for each node η in a single pass through the states in $\mathcal{X}$. A discrete robot clinic consultation problem that fits into this category is solved in [11].

We let $\mathcal{N}(x)$ be 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 states for which x is a neighbor. Let $c : \Upsilon \times \mathcal{X}^2 \to \mathbb{R}^+$ be a positive direct commuting cost function, where $c(\eta, x, y)$ gives the cost of commuting directly from state $x \in \mathcal{X}$ to $y \in \mathcal{X}$. If $y \notin \mathcal{N}(x)$ then $c(\eta, x, y) = \infty$ for all $\eta \in \Upsilon$.

Let $\zeta(l)$, $1 \le l \le L^\zeta$ be a trajectory of L^ζ states through $\mathcal{X}$ with each $\zeta(l) \in \mathcal{X}$ and such that $\zeta(l+1) \in \mathcal{N}(\zeta(l))$, for $1 \le l \le L^\zeta - 1$. We may use L in place of L^ζ where the trajectory ζ is obvious. Define $\lambda(\zeta)$ as

$$\lambda(\zeta) = \sum_{l=1}^{L-1} c(\eta, \zeta(l), \zeta(l+1)).$$

We observe that the value $w(\eta, x)$ for any node $\eta \in \Upsilon$ and any state $x \in \mathcal{X}$ can be computed using only the value $v(\eta, x)$ at the same state $x \in \mathcal{X}$ and the values $w(\eta, y)$ at the neighboring states $y \in \mathcal{N}^{-1}(x)$. The following proposition establishes a DP principle based on this property and is equivalent to Theorem 2 in [11].

Proposition 1: The value function w satisfies for all nodes $\eta \in \Upsilon$ and for all $x \in \mathcal{X}$

$$\max \left\{ \begin{array}{l} w(\eta, x) - v(\eta, x), \\ \max_{y \in \mathcal{N}^{-1}(x)} \left[w(\eta, x) - w(\eta, y) - c(\eta, y, x) \right] \end{array} \right\} = 0. \tag{13}$$

As long as $|\mathcal{N}(x)|$ and $|\mathcal{N}^{-1}(x)|$ are independent of $|\mathcal{X}|$, the values w for each node η can be computed efficiently using a slight modification of Dijkstra's algorithm. The update in Dijkstra's algorithm is replaced by (13). Also, the locations $p^*(\eta)$ for each node η can be computed using a discrete steepest descent algorithm, which moves backwards along an optimal meeting-to-meeting trajectory in $\arg\min_{\zeta \in \mathcal{Z}(p^*(\eta), p^*(\iota(\eta)))} \lambda(\zeta)$ until the condition $w(\eta, \zeta(l)) = v(\eta, \zeta(l))$ is satisfied. At this point an optimal meeting location $p^*(\eta)$ has been reached. The implementation of these algorithms in the context of the tree DP algorithm is presented in [11].

B. Continuous State Space

Let $\mathcal{X} \subset \mathbb{R}^d$ be a compact, connected state space. Let $\zeta(t)$, $t \in [T_0^\zeta, T_f^\zeta]$ be a trajectory through $\mathcal{X}$. Constrain

$$\dot{\zeta}(t) = \frac{d\zeta(t)}{dt} \in \mathcal{A}(\eta, \zeta(t)) \tag{14}$$

where $\mathcal{A}(\eta, x) \subset \mathbb{R}^d$ is a compact, convex action set containing the origin in its interior. We may use T_0 and T_f in place of T_0^ζ and T_f^ζ where the trajectory ζ is obvious. Define $\lambda(\zeta)$ as

$$\lambda(\zeta) = T_f^\zeta - T_0^\zeta. \tag{15}$$

A DP principle for this problem can be stated in the form of a differential variational inequality that includes a Hamilton-Jacobi PDE as a component. A closely related variational inequality for an optimal stopping problem is derived in [12].

Proposition 2: The value function w satisfies in the viscosity sense for all nodes $\eta \in \Upsilon$ and for all $x \in \mathcal{X}$

$$\max \left\{ w(\eta, x) - v(\eta, x), \max_{a \in \mathcal{A}(\eta, x)} \left[D_x w(\eta, x) \cdot a - 1 \right] \right\} = 0. \tag{16}$$

It is usually not possible to solve this variational inequality exactly. Instead we may compute an approximate solution $\bar{w}$ by solving a discretized inequality on a grid. For example, the state space may be discretized into an orthogonal grid and the spatial gradient $D_x w(\eta, x)$ in (16) replaced with an upwind, first-order finite difference approximation. Approaches for solving the discretized Hamilton-Jacobi PDE include sweeping methods [13], the Fast Marching Method (FMM) [14], [15], and Ordered Upwind Methods (OUMs) [16]. The discretized Hamilton-Jacobi PDE can be modified to form a monotone discretization of the variational inequality [17] which can be solved by any of the above methods. Other appropriate DP algorithms [5] may also be used to solve the discretized variational inequality.

Approximate meeting locations $\bar{p}^*(\eta)$ for each η can be computed using a continuous steepest descent algorithm. Assuming $\bar{v}$ and $\bar{w}$ have been calculated, the steepest descent solves the following ODE to determine an approximately optimal trajectory from $\bar{p}^*(\iota(\eta))$ backwards towards $\bar{p}^*(\eta)$:

$$\frac{d\bar{\zeta}}{-dt} = -\operatorname*{argmax}_{a \in \mathcal{A}(\eta,x)} \left[D_x \bar{w}(\eta, x) \cdot a \right]. \tag{17}$$

For example, in a simple implementation, the ODE (17) is discretized using forward Euler and the gradient $D_x \bar{w}(\eta, x)$ is determined by a first-order finite difference scheme. The computation of the trajectory is completed and $\bar{p}^*(\eta)$ is found when $\bar{w}(\eta, \bar{\zeta}(t)) \geq \bar{v}(\eta, \bar{\zeta}(t))$.

C. Algorithm Complexity

We first consider the case of the discrete spatial graph. Dijkstra's algorithm has complexity $\mathcal{O}(|\mathcal{X}||\mathcal{N}(x)| \log |\mathcal{X}|) = \mathcal{O}(|\mathcal{X}| \log |\mathcal{X}|)$ if it is implemented by maintaining a min-heap sorted on $w(\eta, x)$. The complexity of the steepest descent algorithm is $\mathcal{O}(|\mathcal{X}||\mathcal{N}^{-1}(x)|) = \mathcal{O}(|\mathcal{X}|)$, since the optimal meeting-to-meeting trajectory can pass through each node $x \in \mathcal{X}$ at most once and at each node consider at most $|\mathcal{N}^{-1}(x)|$ neighbors to extend the trajectory. Consequently, the complexity of computing w using Dijkstra's algorithm still dominates the other costs. However, the overall complexity of the tree DP algorithm has been reduced to $\mathcal{O}(|\Upsilon||\mathcal{X}| \log |\mathcal{X}|)$.

We now consider the use of FMM or OUMs for the continuous state space case. FMM and OUMs are generalizations of Dijkstra's algorithm which have complexity $\mathcal{O}(|\bar{\mathcal{X}}| \log |\bar{\mathcal{X}}|)$, where $\bar{\mathcal{X}}$ is the discretized state space. So, in this case the overall complexity of the tree DP algorithm is $\mathcal{O}(|\Upsilon||\bar{\mathcal{X}}| \log |\bar{\mathcal{X}}|)$.

VI. Example

We solve a continuous robot arm cargo transport problem. 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 total cost of transporting the cargo across the workspace. The corresponding meeting tree is shown in Figure 1, and the resulting robot arm motions are depicted in Figure 2.

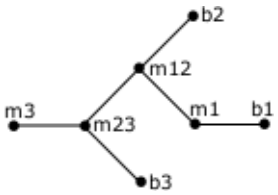

Fig. 1. Meeting tree for the robot arm problem. Begin robot i is indicated by bi. Pickup cargo for robot 1 is indicated by m1. Pass cargo from robot i to robot j is indicated by mij. Dropoff cargo for robot 3 is indicated by m3.

Each robot arm has 2 degrees of freedom, one for each of its joints. Consequently, each has a continuous 2-dimensional state space. Let a robot's state space $\mathcal{X}$ exclude those states that result in the intersection of the robot with an obstacle. Let $\mathcal{A}(\eta, x)$ from (16) be defined as

$$\mathcal{A}(\eta, x) = \{a \mid \|a\|_1 \leq 1\}$$

for all η and all $x \in \mathcal{X}$. We use the Manhattan norm to constrain $\mathcal{A}(\eta, x)$ because we wish to minimize the sum of the joint costs for each robot [18]. The total cost of transporting the cargo is the sum of the costs of the robot state trajectories, so we define ξ as in (1). There are no meeting costs incurred when two robots meet (meetings mij in Figure 1). 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 the algorithm described in Section V-B to solve the robot arm problem. We use FMM to solve the discretized variational inequality. More specifically, for each meeting node we approximately solve an Eikonal-type variational inequality

$$\max\left\{w(\eta, x) - v(\eta, x), \|D_x w(\eta, x)\|_\infty - 1\right\} = 0.$$

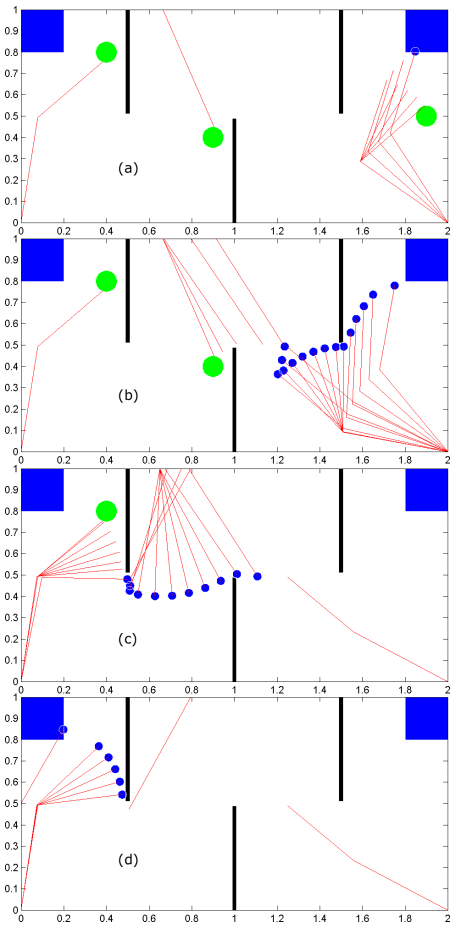

Fig. 2. Robot arm motions for a solution meeting tree. The sequence of figures is a time series with (a) showing the beginning of the robot arm motions and (d) showing the completion of the motions. The starting area for the end effector of each robot arm is indicated by a large circle. The square box 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.

on a discretized orthogonal grid of the robot state space. The reasons for solving the corresponding Eikonal PDE and methods for doing so are discussed in [18].

We use (15) to measure the cost of a robot trajectory in the robot state space, but the occurance of a meeting depends on the end effector location in the workspace. For this reason, we modify the algorithm to employ two grids for each meeting node: a workspace grid for the values $\bar{v}$ and a robot state grid for the values $\bar{w}$. For the solution illustrated in Figure 2 we use a workspace grid of 401×201 nodes, a state grid of 401×101 nodes for robots 1 and 3, and a state grid of 201×201 nodes for robot 2. During the computation of $\bar{v}$, $\bar{w}$, and $\bar{p}^*(\eta)$, forward kinematics are used to map robot states to end effector locations in the workspace. More details on how the algorithm uses the workspace grid and robot state grids can be found in [11].

VII. Conclusion

We have specified a class of multi-location robot rendezvous problems for which dynamic programming can be used to find optimal solutions. Appropriate dynamic programming principles have been presented and used to construct an efficient algorithm. The applicability of the algorithm has been demonstrated on a continuous robot arm example.

Acknowledgment

This work was supported by a Discovery grant from the National Sciences and Engineering Research Council of Canada. We thank the authors of [1] for motivation.

References

[1] 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.

[2] 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.

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

[4] E. W. Dijkstra, "A note on two problems in connection with graphs," *Numer. Math.*, vol. 1, no. 1, pp. 269–271, Dec. 1959.

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

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

[7] Wikipedia, "Steiner tree," 2008, [accessed 6-August-2008]. [Online]. Available: http://en.wikipedia.org/wiki/Steiner%5Ftree

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

[9] M. S. Branicky, R. Hebbar, and G. Zhang, "A fast marching algorithm for hybrid systems," in *Proceedings of the 38th Conference on Decision and Control*, 1999, pp. 4897–4902.

[10] K. Alton and I. M. Mitchell, "Efficient dynamic programming for optimal multi-location robot rendezvous with proofs," Department of Computer Science, University of British Columbia, Tech. Rep. TR-2008-11, 2008.

[11] ——, "Efficient optimal multi-location robot rendezvous," Department of Computer Science, University of British Columbia, Tech. Rep. TR-2007-17, 2007.

[12] M. Bardi and I. Capuzzo-Dolcetta, *Optimal Control and Viscosity Solutions of Hamilton-Jacobi-Bellman Equations*. Boston and Basel and Berlin: Birkhauser, 1997.

[13] J. Qian, Y. Zhang, and H. Zhao, "A fast sweeping method for static convex Hamilton-Jacobi equations," *J. Sci. Comput.*, vol. 31, no. 1-2, pp. 237–271, May 2007.

[14] J. N. Tsitsiklis, "Efficient algorithms for globally optimal trajectories," in *Proceedings of the 33rd Conference on Decision and Control*, 1994, pp. 1368–1373.

[15] J. A. Sethian, "A fast marching level set method for monotonically advancing fronts," *Proc. Natl. Acad. Sci.*, vol. 93, pp. 1591–1595, 1996.

[16] J. A. Sethian and A. Vladimirsky, "Ordered upwind methods for static Hamilton-Jacobi equations: Theory and algorithms," *SIAM J. Numer. Anal.*, vol. 41, no. 1, pp. 323–363, 2003.

[17] A. M. Oberman, "Convergent difference schemes for degenerate elliptic and parabolic equations: Hamilton-Jacobi equations and free boundary problems," *SIAM J. Numer. Anal.*, vol. 44, no. 2, pp. 879–895, 2006.

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