Distributed Optimization with Pairwise Constraints and its Application to Multi-robot Path Planning

Subhrajit Bhattacharya
Department of Mechanical Engineering and Applied Mechanics
University of Pennsylvania
Philadelphia, PA 19104
Email: subhrabh@seas.upenn.edu

Vijay Kumar
Department of Mechanical Engineering and Applied Mechanics
University of Pennsylvania
Philadelphia, PA 19104
Email: kumar@seas.upenn.edu

Maxim Likhachev
Department of Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104
Email: maximl@seas.upenn.edu

***Abstract*— Distributed approaches to constrained optimization problems have immense applications to multi-robot path planning, scheduling, task allocation and other problems requiring multiple robots to optimize a global objective function. The aim of these approaches is to solve a series of smaller optimization problems for each robot while sharing information among the robots, and in the process, solve the global optimization problem, which otherwise would have been intractable. Distributed approaches to separable convex optimization problems with linear constraints have been studied extensively in the past using techniques of dual and Lagrangian decomposition. In the present work, we investigate a distributed implementation of a general separable optimization problem with pair-wise non-linear constraints. On the theoretical side, we show the conditions under which the algorithm converges to an optimal solution. On the experimental side, we demonstrate the utility of the algorithm on the problem of multi-robot path planning with pair-wise distance constraints in large complex 2-D environments with obstacles.**

I. Introduction

Distributed implementation of optimization problems is an important field of research in distributed systems [15, 4]. In many optimization problems the joint state space of all the search variables is too big or too complex for the optimization problem to be solved centrally. At other times the complete information about all the state variables is not available to any central processor. Thus distributed implementation of such problems become indispensable. One example of particular interest to us is multi-robot planning problems. For instance, robots navigating towards their respective goals while staying within the communication range is one of the common planning problems in multi-robot robotics. Here the search variables are the robot trajectories, each of which theoretically lies in an infinite dimensional Hilbert space. Planning in the joint state-space of all the robots in such a case while satisfying complex constraints may be very expensive, if not practically impossible. Multi-robot path planning suffers from the inherent complexity resulting from the necessity of operating in Cartesian products of configuration and state spaces [7]. The continuous path planning problem is even more difficult to solve in a centralized setting [12] unless the problem is solved sequentially for each robot [16]. Open loop trajectory planning problems can be reduced to optimization problems. While completeness results are often possible [1] for simple problems with no constraints, it is difficult to respect more complex multi-robot constraints.

Another instance of distributed optimization problem is task allocation for multiple robots [10]. In these methods, one can impose rendezvous constraints at intermediate time points as tasks and reformulate the path planning problem as a task allocation problem. This then lends itself to auction-based solutions [8] for the team. However, these methods can produce highly sub-optimal solutions in the environments with obstacles without guarantees on convergence. There also exist an extensive research on distributed planning in a more general sense (a good survey can be found in [9]).

Separable optimization problems [4] (optimization problems that can be split up into simpler sub-problems involving only certain partitions of the variable set) with linear constraints have been studied extensively in the past and solved in a distributed fashion using techniques based on dual decomposition [15, 4]. Augmented Lagrangian type methods have been used for solving similar problems more efficiently [3, 14]. However such methods are limited to problems with linear constraints and rely on convexity of cost functions.

In the present work we investigate a distributed implementation of a separable optimization problem with *non-linear* constraints arising from coupling between pairs of robots. We do not make any assumption on the convexity of the cost or the constraint functions. Our theoretical analysis shows that the algorithm converges to an optimal solution under certain conditions. As a demonstration of the implementation of our algorithm we will mostly concentrate on solving the problem of path planning for teams of robots coupled with constraints on the distances between pairs of robots. Continuous motion planning is possible for such problems [2], but only practical in environments with moderate complexity. In this paper we explore discrete path planning algorithms for solving the individual simpler optimization problems in a large environment with obstacles and solve the global problem using our distributed optimization algorithm. We show that our approach is able to find efficiently optimal paths with complex cost functions, in arbitrarily complex environments, and with non-linear pair-wise constraints. We therefore demonstrate the utility and versatility of the proposed algorithm by solving

large scale optimization problems in a distributed fashion, which otherwise would have been intractable.

II. Problem Definition

A. The Optimization Problem

In this paper we study the following problem: Find

$$\{\pi_1^*, \ldots, \pi_N^*\} = \text{argmin}_{\pi_1 \ldots \pi_N} \sum_{j=1 \ldots N} c_j(\pi_j) \qquad (1)$$

subject to the pairwise constraints

$$\Omega_{ij}(\pi_i^*, \pi_j^*) = 0, \quad i, j = 1 \cdots N \qquad (2)$$

We call each π_i a *partition* of the set of search variables. In the context of multi-robot planning problem π_i will represent the path of robot i, while $c_j(\pi_j)$ will be the cost of path π_j. Ω_{ij} will represent the violation of the constraint between the trajectories of robot i and j.

B. Problem Assumptions

The following assumptions are made about the variables and functions appearing in the problem definition:

1. The optimization variables π_i lie in abstract vector spaces that are continuous, differentiable and simply connected. In the general case they may be considered as vectors in a finite dimensional Euclidean space, but they can lie (as we'll discuss later) in more complex spaces like infinite dimensional Hilbert spaces. We represent the space in which π_i lies as $\mathbb{H}$
2. The cost functions $c_i : \mathbb{H} \rightarrow \mathbb{R}_+$ are assumed to be continuous and smooth. In context of path planning for mobile robots, an example of such a cost function is the Euclidean length of the trajectories in an environment without obstacles. However, in our implementation we will later relax the condition. Note that we don't make any immediate assumption on the convexity of the cost functions.
3. The functions $\Omega_{ij} : \mathbb{H} \times \mathbb{H} \rightarrow \mathbb{R}$, defined for the unordered pair $\{i, j\}$, are continuous. Also, we require that the second derivatives of Ω_{ij} with respect to each of its parameters as well as the first mixed derivative are defined. That is, $\Omega_{ij}^{(0,2)}, \Omega_{ij}^{(2,0)}$ and $\Omega_{ij}^{(1,1)}$ are defined, where the superscripts denote the order of partial derivatives w.r.t. the respective parameters. A simple example of such a constraint function in context of multi-robot path planning is constraint on the distance between trajectories of two robots.
4. The functions Ω_{ij} are assumed to have the following properties
   i. Ω_{ij} is symmetric in its two parameters (i.e. $\Omega_{ij}(\pi_i, \pi_j) = \Omega_{ij}(\pi_j, \pi_i)$), and
   ii. $\Omega_{ij}^{(1,0)}(\pi_i, \pi_j) = -\Omega_{ij}^{(0,1)}(\pi_i, \pi_j)$.

   It is easy to note that these properties will be true if Ω_{ij} has the functional form $\Omega_{ij}(\pi_i, \pi_j) = G_{ij}(\pi_i - \pi_j)$, where $G_{ij} : \mathbb{H} \rightarrow \mathbb{R}$ is a continuous, smooth even function. Note that we don't make any immediate assumption on the convexity of Ω_{ij}.
6. We assume that besides c_r and Ω_{ij}, the quantities $c_r^{(2)}$, $\Omega_{ij}^{(1,0)}$, $\Omega_{ij}^{(2,0)}$ and $\Omega_{ij}^{(1,1)}$ are readily computable for a given set of input variables. This may be achieved either by knowing the expressions of the derivatives in closed form, or by computing them numerically.

If in a particular problem the cost and constraint functions are not smooth, they can always be approximated by smooth functions at the non-smooth regions using one of the many Mollification techniques [13]. We also note that inequality constraints can be converted to equality constraints (as required by 2) by taking $\max$ or $\min$ with zero, followed by Mollification treatments.

It is to be noted that in spite of the conditions imposed on Ω_{ij}, they can represent a wide variety of constraints, especially in robotics applications. The proposed functional form of the constraints can model constraints on communication, visibility, rendezvous, convoying, collision avoidance, and other more complex coordination between pairs of robots involving their trajectories as well as their derivatives (say, to incorporate dynamic and kinematic constraints).

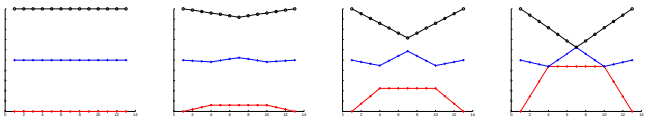
III. The Algorithm

A pseudo-code of our algorithm is presented in Figure 1. The global optimization problem is decomposed into a series of lower-dimensional unconstrained optimization problems, each of which is solved in a single *partition*, π_r, of the search variables. In each iteration the penalty weights on the violation of constraints, W^k, are incremented along the direction given by V^k. The intuitive concept behind the algorithm is that we start off by solving the global unconstrained problem, which is completely decoupled (line 1). Then we gradually increase the penalty weights (line 6) for the constraints which are modeled as soft constraints. The directions in which we can change the weights to guarantee optimality and convergence are described later in the *Theoretical Analysis* section. With the new set of weights we solve a sub-problem, which is an unconstrained optimization problem on only a single partition, namely π_r (line 7). Thus we note that in each iteration only the variables in a single partition are changed, while the others remain unchanged.

An application of the algorithm in goal-directed navigation of multiple robots with rendezvous constraints at various points on the trajectory is illustrated in Figure 2. We note that the algorithm starts off with unconstrained trajectories at $k = 0$ and in each iteration only a single robot plans its trajectory. As the iterations progress, the robots gradually change their trajectories due to increase in penalty weights and try to satisfy the rendezvous constraints. Eventually they reach the global solution. It is to be noted that the gradual increment in the weights play a key role in ensuring that the solution is optimal. One-shot increase in the weights to large values to satisfy the constraints would have resulted in suboptimal trajectories since the robots are planning sequentially. This particular example is described in more details in the "Results" section of the paper.

procedure DistributedOptimization()
1 compute $\pi_i^0 = \text{argmin}_{\pi_i} c_i(\pi_i),\ \ \forall i \in \mathcal{N}^N$;
2 set $W_{ij}^0 = 0$ for all unordered $\{i,j\} \in \mathcal{P}^N$;
3 $r = r_0 \in \mathcal{S}$, $k = 0$;
4 while $\left(\Omega_{ij}(\pi_i^k, \pi_j^k) \neq 0\ \forall \{i,j\} \in \mathcal{P}^N\right)$
5 set $V^k = ComputeStepDirection(W^k, \{\pi\}^k, r)$
6 set $W^{k+1} = W^k + \epsilon^k V^k$;
7 compute $\pi_r^{k+1} = \text{argmin}_{\pi_r}\big[c_r(\pi_r) + \sum_{\{ir\} \in \mathcal{P}_r^N} W_{ir}^{k+1} \Omega_{ir}(\pi_i^k, \pi_r)\big]$;
8 set $\pi_j^{k+1} = \pi_j^k$ for all other $j \neq r$;
9 set $k = k+1$;
10 set $r = r_k \in \mathcal{S}$;
11 end

Fig. 1. Distributed Optimization Algorithm



(a) Unconstrained plans, $k = 0$ (b) $k = 50$ (c) $k = 150$ (d) Converged solution, $k = 216$

Fig. 2. Demonstration of convergence towards global optimal solution with progress of iterations. In this example there are 3 robots with unconstrained trajectories 12 units long, parallel and separated by 5 units. Trajectories are defined by displacements along Y of 11 unit-spaced points on each.

The sets $\mathcal{N}^N$ is the set of all natural numbers from 1 to N, and $\mathcal{P}^N$ is the set of all unordered pairs of numbers in $\mathcal{N}^N$. The set S gives a sequence of partitions for which the individual sub-problems are solved in each iteration. $\mathcal{S} = \{r_0, r_1, r_2, \cdots\}$, where $r_i \in \{1, 2, \cdots, N\}\ \forall i \in \mathbb{N}$ can be constructed such that the partitions appear almost at equal frequency in the sequence. As we will see later, the sequence does not influence the convergence or optimality of the solution, but may influence the number of iterations required to converge.

Also, the step-sizes, ϵ^k, are small and determine the precision of the solution. The theoretical analysis in the paper are guaranteed to hold when the step-sizes are infinitesimal. But for practical purpose we choose finite and small step-sizes. The sequence $\mathcal{E} = \{\epsilon^0, \epsilon^1, \epsilon^2, \cdots\}$ may be predefined, set to a fixed constant step-size, or may be adaptive.

The procedure "*ComputeStepDirection*" in the Algorithm computes the direction in which to increase the penalty weights, W. We call this direction a *Step Direction*. In the following section (and especially Theorem 3) we will discuss the ways of computing such a direction.

IV. Theoretical Analysis

In this section we investigate the conditions under which the proposed algorithm will converge to an optimal solution. Theorem 2 along with Theorem 1 proves that under certain conditions the algorithm is guaranteed to converge to an optimal solution. Theorem 3 gives a prescription, using which will ensure that the said conditions hold.

A. Notations and preliminaries

1) Unordered pair: We define the set of unordered pairs of natural numbers from 1 to N, and its subsets as follows:

$$\mathcal{P}^N = \{\{1,2\}, \{1,3\}, \cdots, \{1,N\}, \{2,3\}, \{2,4\}, \cdots, \{N-1,N\}\}$$

$$\text{and, } \mathcal{P}_r^N = \{\{1,r\}, \cdots, \{r-1,r\}, \{r+1,r\}, \cdots, \{N,r\}\}$$

In the following discussions, the subscripts "ij" of quantities (like W) or functions (like Ω) are elements from $\mathcal{P}^N$. Thus the order of the subscripts does not matter. When we write W (or W_1, or W_2), we mean the vector of length $\frac{1}{2}N(N-1)$ of all the W_{ij}'s. Also, often we will write $\{i,j\} \equiv \{k,r\} \in \mathcal{P}_r^N$ to indicate that $\{i,j\}$ is such that exactly one of i or j is equal to r, while the other one, which is not equal to r, is denoted by k.

2) Other notations:

i. For notational convenience we define the sets $\mathcal{N}^N = \{1, 2, \cdots, N\}$ and $\mathcal{N}_{-r}^N = \{1, 2, \cdots, r-1, r+1, \cdots N\}$

ii. We denote the set $\{\pi_1, \pi_2, \cdots, \pi_N\}$ as $\{\pi\}$. On similar lines, if there are arbitrary functions $\Upsilon_i : \mathbb{A} \to \mathbb{B}$, $i = \{1, 2, \cdots N\}$, we denote the collection of all these functions as $\{\Upsilon\} : \mathbb{A} \to \mathbb{B} \times \mathbb{B} \times \cdots \times \mathbb{B}$. Conversely, the j^{th} element of $\{\Upsilon\}$ is denoted as $[\{\Upsilon\}]_j$ or Υ_j

iii. The subset of $\{\pi\}$ without the r^{th} element is denoted by $\{\pi\}_{-r}$. Thus, $\{\pi\}_{-r} = \{\pi_1, \pi_2, \cdots, \pi_{r-1}, \pi_{r+1}, \cdots, \pi_N\}$

iv. If we have a smooth function $f : \mathcal{A}_1 \times \mathcal{A}_2 \times \cdots \mathcal{A}_n \to \mathbb{R}$, its derivatives are represented by $f^{(a_1, a_2, \cdots, a_n)}$. Thus, $f^{(1)} \equiv \nabla f$, $f^{(2)} \equiv \nabla^2 f$, $f^{(1,1)} \equiv \nabla_x \nabla_y f$, etc.

v. We define the Lagrangian of the global problem $\overline{U}$, and the Dual function $\overline{\Psi}$,

$$\begin{aligned} \overline{U}(\{\pi\}, W) &:= \sum_{k \in \mathcal{N}^N} c_k(\pi_k) + \sum_{\{kl\} \in \mathcal{P}^N} W_{kl} \Omega_{kl}(\pi_k, \pi_l) \\ \{\overline{\Pi}\}(W) &:= \arg\min_{\{\pi\}} \left[\overline{U}(\{\pi\}, W)\right] \\ \overline{\Psi}(W) &:= \min_{\{\pi\}} \left[\overline{U}(\{\pi\}, W)\right] = \overline{U}(\{\overline{\Pi}\}(W), W) \end{aligned} \quad (3)$$

In other words, $\{\overline{\Pi}\}(W)$ is the global optimum for the penalized objective function with W as penalty weights, and $\overline{\Psi}(W)$ is the optimal value.

vi. Similarly, we define for each *partition* π_r,

$$\begin{aligned} U_r(\pi_r, W_1, W_2) &:= c_r(\pi_r) + \sum_{\{kr\} \in \mathcal{P}_r^N} W_{1,kr} \Omega_{kr}(\overline{\Pi}_k(W_2), \pi_r) \\ \Pi_r(W_1, W_2) &:= \arg\min_{\pi_r} \left[U_r(\pi_r, W_1, W_2)\right] \\ \Psi_r(W_1, W_2) &:= \min_{\pi_r} \left[U_r(\pi_r, W_1, W_2)\right] \\ &= U_r(\Pi_r(W_1, W_2), W_1, W_2) \end{aligned} \quad (4)$$

That is, for a given value, $\{\overline{\Pi}\}_{-r}(W_2)$, of the partitions (except the r^{th} one), $\Pi_r(W_1, W_2)$ gives the optimum for an individual sub-problem with W_1 as penalty weights.

vi. Finally, we define the following,

$$\begin{aligned} \mathbf{M}_r(W) &= c_r^{(2)}(\overline{\Pi}_r(W)) \\ &\quad + \sum_{\{lr\} \in \mathcal{P}_r^N} W_{lr} \Omega_{lr}^{(0,2)}(\overline{\Pi}_l(W), \overline{\Pi}_r(W)) \\ \mathbf{N}_{lr}(W) &= \Omega_{lr}^{(1,1)}(\overline{\Pi}_l(W), \overline{\Pi}_r(W)) \end{aligned} \quad (5)$$

Also, we note that the r^{th} component of $\{\overline{\Pi}\}(W)$ is given by,

$$\begin{aligned} &\overline{\Pi}_r(W) \\ &= \arg\min_{\pi_r} \Big(c_r(\pi_r) + \sum_{\{kr\} \in \mathcal{P}_r^N} W_{kr} \Omega_{kr}(\overline{\Pi}_k(W), \pi_r) \\ &\qquad + \sum_{k \in \mathcal{N}_{-r}^N} c_k(\overline{\Pi}_k(W)) \\ &\qquad + \sum_{\{kl\} \in \mathcal{P}^N / \mathcal{P}_r^N} W_{kl} \Omega_{kl}(\overline{\Pi}_k(W), \overline{\Pi}_l(W))\Big) \\ &= \arg\min_{\pi_r} \Big(c_r(\pi_r) + \sum_{\{kr\} \in \mathcal{P}_r^N} W_{kr} \Omega_{kr}(\overline{\Pi}_k(W), \pi_r)\Big) \\ &= \Pi_r(W, W) \end{aligned} \quad (6)$$

$$\text{Thus,} \quad \overline{\Pi}_r^{(1)}(W) = \Pi_r^{(1,0)}(W, W) + \Pi_r^{(0,1)}(W, W) \quad (7)$$

B. Theorems

Definition 1: [*Separable Optimal Flow*] Given the functions c_r, $\Omega_{ir}\ \forall\{ir\} \in \mathcal{P}_r^N$ we call V a *Separable Optimal Flow Direction* and ϵ a *Separable Optimal Flow Step* at W for Ψ_r if and only if the following holds,

$$\begin{aligned}&\Psi_r(W+\epsilon V, W) - \Psi_r(W,W)\\ &\quad\leq \Psi_r(W+\epsilon V, W+\epsilon V) - \Psi_r(W, W+\epsilon V) \qquad (8)\\ &\text{and, } V_{ij} = 0,\ \ \forall\{i,j\} \text{ such that } r \notin \{i,j\}\end{aligned}$$

Together, V and ϵ are said to define a *Separable Optimal Flow* at W for Ψ_r.

Theorem 1: [*Optimality at each Iteration*] If the *Step Direction*, V^k, returned by procedure $ComputeStepDirection$ at the k^{th} iteration in Line 5 of the Algorithm, along with the chosen *Step Size*, ϵ^k, define a *Separable Optimal Flow* at W^k for $\Psi_{r_\kappa},\ \forall\ k$, then $\forall\ k$:

$$\begin{aligned}&\{\pi_1^k, \ldots, \pi_N^k\}\\ &\quad= \arg\min_{\{\pi\}} \left[\textstyle\sum_{i\in\mathcal{N}^N} c(\pi_i) + \sum_{\{ij\}\in\mathcal{P}^N} W_{ij}^k \cdot \Omega_{ij}(\pi_i,\pi_j)\right]\end{aligned}$$

Proof: Detailed proof can be found in Appendix ■

The result of the Theorem 1, in brief, can be stated as

$$\pi_i^k = \overline{\Pi}_i(W^k),\quad \forall i,k \qquad (9)$$

The implication of the result is that there are specific directions (which we call *Separable Optimal Flow Directions*) in which we can increment the penalty weight vector W, such that the global optimum for the new set of penalty weights differs from the previous global optimum (i.e. optimum for the previous set of weights) in only one partition of the optimization variables, namely π_r. Thus, by moving along such a direction in k^{th} iteration, we only need to change π_{r_k}, and still remain at an optimum of the penalized net cost.

Definition 2: [*Ascent Direction*] We call V an *Ascent Direction* at W if and only if the following holds,

$$\sum_{\{ij\}\in\mathcal{P}^N} V_{ij}\Omega_{ij}(\overline{\Pi}_i(W), \overline{\Pi}_j(W)) > 0 \qquad (10)$$

Theorem 2: [*Convergence of the Algorithm*] If the conditions in Theorem 1 hold, and the *Step Direction*, V^k, returned by the procedure $ComputeStepDirection$ at the k^{th} iteration in Line 5 of the Algorithm is also an *Ascent Direction* at W^k for every k, then the Algorithm converges to an optimal solution if one exists.

Proof: Detailed proof can be found in Appendix ■

The result of Theorem 2 implies that if we always increment the penalty weights along directions that are both *Ascent Direction*s and *Separable Optimal Flow Direction*s, we will eventually converge to the global optimum, if it exists. Knowing the π_i^k's from previous iterations, it is easy to choose such a direction V^k for the current iteration as one that has positive inner product with the vector of all constraint violations at k^{th} iteration (i.e., the vector made of $\Omega_{ij}(\pi_i^k, \pi_j^k)$'s).

Theorem 3: [*Computation of Separable Optimal Flow Direction*] If the functions c_r and $\Omega_{ir}\ \forall\{ir\} \in \mathcal{P}_r^N$ abide by the *Problem Assumptions*, we can find a *Separable Optimal Flow Direction* for Ψ_{r_k} at W^k defined by V^k, if it exists, along with a small enough *Step Size*, ϵ^k, at Line 5 of the Algorithm, using only the following quantities that are readily computable:

$W^k,\ \ c_i^{(2)}(\pi_i^k)\ \forall i \in \mathcal{N}^N,$
$\Omega_{ij}^{(0,2)}(\pi_i^k, \pi_j^k),\ \Omega_{ij}^{(1,1)}(\pi_i^k, \pi_j^k)$
and $\Omega_{ij}^{(1,0)}(\pi_i^k, \pi_j^k)\quad \forall\{ij\} \in \mathcal{P}^N,$

In general we get to choose from a large set of possible *Separable Optimal Flow Direction*s, thus giving us the opportunity to make it as an *Ascent Direction* as well.

Proof: Detailed proof can be found in Appendix ■

The theorem implies that we can compute *Separable Optimal Flow Directions*, if one exists, in terms of quantities that can be easily computed using the variables from the previous iteration. Within the limits of the error introduced by the finite step-size, we can compute a *Separable Optimal Flow*.

It is important to note that in general the freedom of choosing from a large set of possible *Separable Optimal Flow Direction* (set of linear combinations of the eigenvectors such that (30) is satisfied) gives us the opportunity to choose a vector V^k that is an *Ascent Direction* as well. In case the intersection between the set of *Separable Optimal Flow Directions* and the set of *Ascent Directions* is empty, we can choose to skip to the next element in the sequence $\{r_k\}$.

V. Results

A. An Exact Implementation

We demonstrate the algorithm using a MATLAB implementation of an idealized planning problem. The specific problem under consideration may be considered as a multi-robot goal-directed path planning in an environment without obstacles, where the trajectories of the robots are defined by displacements of unit-spaced points on the trajectories in vertical direction. Thus the trajectory, of a robot is given by $\pi_r = [start_r, y_{r2}, y_{r3}, \cdots, y_{rL}, goal_r]^T$, where $y_{ri}, i = 2\cdots L$ are the search variables and represent displacement of a point at x_i of the r^{th} robot's trajectory. The constraints between robots a and b are rendezvous constraints defined by

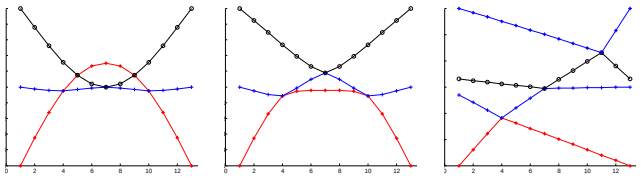
$$\Omega_{ab}(\pi_a, \pi_b) = \left((y_{ac_1} - y_{bc_1})^2 + (y_{ac_2} - y_{bc_2})^2 + \cdots\right)^{1/2} = 0$$

where $c_1, c_2, \cdots$ are the points where a and b need to rendezvous. There are two components to the costs functions c_r - the length of the trajectory, and the integral of the square of accelerations over the trajectory. Assuming the robots have constant X-velocity, the net cost is thus given by a weighted sum of the two components,

$$\begin{aligned}c_r(\pi_r) = \alpha&\left((y_{r2} - start_r)^2 + (y_{r3} - y_{r2})^2 + \cdots\right)^{1/2}\\ &+ \beta\left(((y_{r4} - y_{r3}) - (y_{r3} - y_{r2}))^2 + \right.\\ &\qquad\left.((y_{r5} - y_{r4}) - (y_{r4} - y_{r3}))^2 + \cdots\right)\end{aligned}$$

The individual unconstrained optimization problems in Line 7 of the Algorithm were solved using MATLAB's *fminunc*.

Figure 2 demonstrates how the algorithm approaches the optimal feasible solution with progress of iterations. In this example we chose $\alpha = 1, \beta = 0$ and a constant step size of $\epsilon^k = 0.01$. The constraints are that the top robot needs to rendezvous with the middle one at the middle point (i.e. at x_7), while the bottom and middle ones rendezvous at two points (y_4 and y_{10}). Since it is a symmetric case with only trajectory length as cost, it's easy to compute the optimal solution separately by optimizing over just 2 variables. The optimal cost that way is found to be 43.946, while our algorithm terminates at a cost of



(a) Symmetric case: $\alpha = 0, \beta = 1$ (b) Symmetric case: $\alpha = 1, \beta = 0.2$ (c) An asymmetric case: $\alpha = 1, \beta = 0$

Fig. 3. Converged Solutions

43.962 for the said step size. Figures 3(a) and (b) illustrate the results for other values of α and β demonstrating the ability of the algorithm in dealing with complex cost functions. Figure 3(c) demonstrates an asymmetric case with 4 robots.

We attempted to solve these problems in a centralized fashion using MATLAB's Pseudo-Newton search method implemented in *fmincon*. Even when the gradient and Hessian functions were explicitly provided, it failed to find a solution satisfying the required tolerance with computation time limit of 20 minutes and 5000 as the limit on number of iterations. In contrast, our algorithm solved these problems in about a minute and a few hundred iterations.

B. A Discrete Implementation

In the following examples we perform a more realistic multi-robot path planning with pair-wise distance constraints in a complex environment with large number of obstacles. Thus, even the individual sub-problems that we need to solve at Line 7 of the Algorithm become significantly complex.

The three-dimensional state-space of each robot, $X - Y - Time$, was discretized into uniform cells, each cell representing nodes of the graph, and an A* graph search technique was employed to obtain an optimal path in the graph as an approximation of the trajectory of r^{th} robot, π_r. The connectivity of the graph was such that a cell in the x, y, t space connected to its 8 neighboring cells and itself in x, y but with the time step incremented by one. This implies that any path in the graph will be discrete approximation of an element of $\mathbb{H}$. While an 8-connected grid is quick and efficient to perform search in, it confines the motion of the robot to 8 directions (45° orientations). A consequence of this is that some seemingly sub-optimal solutions are actually optimal (minimum cost paths) in the 8-connected graph.

The cost function, c, is the Euclidean length of the trajectory. The constraints between the pairs of robots are defined by maximum distance, $\phi_{ij}(t)$, that the robots i and j can be apart at a given instant of time. Thus, the constraints are defined by,

$$\Omega_{ij}(\pi_i, \pi_j) = \int_0^T \varpi(\pi_i(t), \pi_j(t), \phi_{ij}(t)) = 0$$

where, $\varpi(\mathbf{s}, \mathbf{s}', p) = \max(0, d(\mathbf{s}, \mathbf{s}') - p)$ and d is a distance function.

1) Dealing with Obstacles: The algorithm and theoretical analysis described so far relies on the fact that the values of all the variables can be changed in small steps in each iterations. Unfortunately the presence of obstacles in the environment violates that condition. However the Optimality condition of Theorem 1 still holds in a single Homotopy class of all the trajectories. This means that as long as a trajectory changes in small steps and does not make big jump from one side of an obstacle to the other, Theorem 1 and the related results from Theorem 3 hold. Thus, whenever we get stuck in a particular homotopy class for all the trajectories, or whenever one of the trajectories makes a jump to a new homotopy class, we block this homotopy class as invalid and restart the planning. It is worth noting here that we use only *large* connected obstacles to characterize homotopy classes and avoid unnecessary creation of homotopy classes by small obstacles that do not effect optimality significantly. We primarily employed two different methods of blocking homotopy classes:

i. *Use of Blacklists* - We maintain a list of blocked regions (or balls) in the joint state-space of the robots violating one or more constraints. Every time we need to restart the planning as above, we update the "Blacklist" with the latest violation information in the homotopy class without a feasible solution. Thus, when we restart the planning we eventually will avoid the homotopy class, provided our choice of blocking ball was proper.

ii. *Systematic identification and blocking of Homotopy class* - Homotopy classes can be systematically identified using geometric methods [11] or complex analysis. We integrated such a method with our A* searches which enabled us to restrict searches in specific homotopy classes or block some homotopy classes.

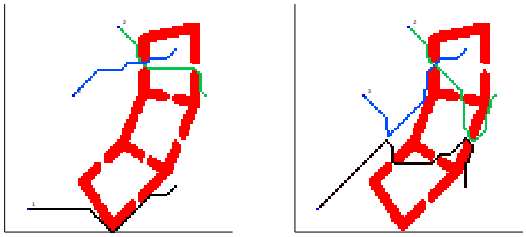
In the following subsections we will present the performances of our algorithm in different environments.

C. Three interconnected rooms

The environment shown in Figure 4 consists of 3 interconnected rooms and 3 robots. The environment is made of discretized cells 89×94 in space and 200 time-steps. This results in the joint state-space of all the robots to have 10^{14} states. Our planner avoids planning in this huge state-space, but rather breaks up the problem into a series of plans in individual state-spaces with $< 2 \times 10^6$ states. The robots start from the left side of the environment and need to reach their goals on the right end. Figure 4(a) shows the unconstrained optimal plan. A constraint is defined between R_1 (black) and R_3 (blue) such that they need to be within a distance of 2 discretization units at $t = 40$. And the constraint between R_1 (black) and R_2 (green) is such that they need to be within a distance of 2 discretization units at $t = 120$. The solution shown in Figure 4(b) was found in about one minute and 72 iterations on a computer running on 1.2GHz processor.

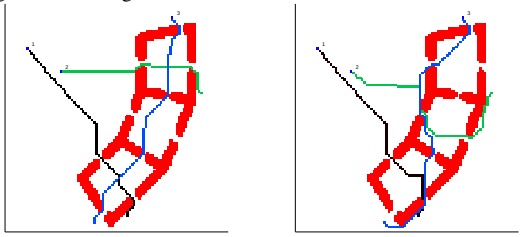
D. Extended rendezvous

The environment shown in Figure 5 is similar to the previous example in size and discretization. In this example R_1 (black) and R_2 (green) need to traverse the environment from left to right, while R_3 (blue) needs to traverse it from top to bottom. Figure 5(a) shows the unconstrained optimal plan. Constraints between R_2 and R_3 is that they need to be within 2 discretization units distance from $t = 32$ to $t = 68$, and the constraints between R_1 and R_3 is that they need to be



(a) Unconstrained plans (b) Converged feasible solution

Fig. 4. Planning in environment with three interconnected rooms



(a) Unconstrained plans (b) Converged feasible solution

Fig. 5. Extended rendezvous in environment with three interconnected rooms

within 2 discretization units distance from $t = 84$ to $t = 120$ The solution in Figure 5(b) was found after 75 iterations with $\epsilon = 0.1$ as the fixed step-size.

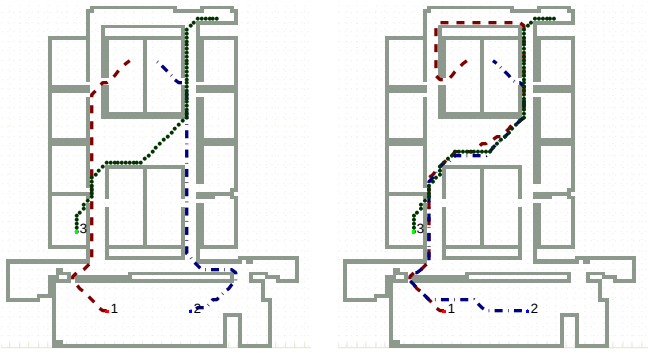
E. Extended rendezvous in a real environment

Figure 6 shows a part of the 4^{th} floor of Levine hall in University of Pennsylvania. The original map is 35 meters by 35 meters, discretized into 100x100 cells (each cell 35cmx35cm, almost the dimension of the Scarab robot, described later). There are 170 discretization steps in time. There are 3 robots and the unconstrained objectives, resulting in the solution in Figure 6(a), are that both Robot 1 (dashed trajectory) and Robot 2 (dash-dot trajectory) need to start at $t = 0$ inside the big room at the bottom and need to reach their respective goals by $t = 170$. Robot 3 (dotted trajectory) needs to start at $t = 45$ inside the small lower cubicle on the left side of the map and needs to reach the small storage space at the top right by $t = 120$. Figures 6(b) shows the converged feasible solution that satisfy the following constraints: A. Robot 2 needs to stay within 3 discretization units of robot 3 from $t = 60$ to $t = 80$; and B. Robot 1 then needs to stay within 3 discretization units of robot 3 from $t = 90$ to $t = 110$. Looking at the converged solution we observe that in order to satisfy constraint A, robot 2 loops it's trajectory around the central and lower cubicles, while to satisfy constraint B, robot 1 loops its trajectory around the central and upper cubicles.

F. Performance

We have successfully implemented the algorithm on teams of up to six robots and in 3 dimensions, that is planning in $X - Y - Z$. We do not present those results in the paper due to limitation on space, as well as to focus on the key features of the algorithm rather than the multitude of applications that it can have. More results and applications can be found in [5].

We tested the performance of our algorithm by running the previous example (*Extended rendezvous in a real environment*) multiple times, but with randomized initial & goal states and randomized time spans for the constraints. The implementation of the problem was made in C++ and was run on a computer with Intel 1.2 GHz processor. The table below gives a summary of the run-time from 54 runs.



(a) Unconstrained plans (b) Converged feasible solution

Fig. 6. Extended rendezvous in a real environment

Min	Max	Average
28 s	169 s	59.7 s

The joint state-space of the three robots contains $\sim 170 \times (100 \times 100)^3 \simeq 1.7 \times 10^{14}$ states. Planning in such a huge graph is highly expensive, if not impossible. However our distributed planning technique was able to find the optimal solution up to the desired precision consistently and sufficiently fast, thus demonstrating the ability of the algorithm to solve large problems.

We also tested our algorithm in simulation using Gazebo, an open-source multi-robot simulator with accurate simulation of rigid-body physics. Experiment were also performed with *Scarab* mobile robot platforms along with an overhead LED tracking system to track the positions of the robots. In order to account for the non-zero radii of the robots, we performed a greedy collision avoidance during run-time. For controlling the non-holonomic robots, a feedback linearization technique was adopted. More details can be found in [5].

VI. Conclusion

We have developed an algorithm for solving large optimization problems with pair-wise nonlinear constraints and arbitrary objective functions in a distributed fashion. Our theoretical analysis gives the conditions under which the algorithm converges to an optimal solution and a prescription for making sure that those conditions are satisfied. We have successfully implemented the algorithm to solve a large multi-robot path planning problem in complex environment with arbitrary shaped obstacles and distance constraints using discrete graph searches for individual sub-problems.

Presently we are investigating the conditions under which *Separable Optimal Flow Direction* and *Ascent Direction* are guaranteed to exist. Generalizing each constrain to include more than two partitions is also under progress. In a nut-shell, besides solving a specific complex optimization problem in

distributed fashion, our approach opens up a new direction for solving related classes of distributed optimization problems.

VII. Appendix

A. Proof of Theorem 1

The theorem clearly holds when for $k = 0$ since $W_{ij}^0 = 0$ for all $\{ij\} \in \mathcal{P}^N$.

We prove the theorem by induction. Assume it holds for $k = 0$ through $k = \kappa$. We will now prove that it continues to hold for the $k = \kappa + 1$. By inductive assumption we have:

$$\{\pi_1^\kappa, \pi_2^\kappa, \cdots, \pi_N^\kappa\} = \arg\min_{\pi_1,\pi_2,\cdots,\pi_N} \left(\sum_{i\in\mathcal{N}^N} c_i(\pi_i) + \sum_{\{ij\}\in\mathcal{P}^N} W_{ij}^\kappa \Omega_{ij}(\pi_i,\pi_j)\right)$$

That is,
$$\pi_i^\kappa = \overline{\Pi}_i(W^\kappa), \quad \forall i \in \mathcal{N}^N \tag{11}$$

We also note that, by Line 8 of the Algorithm,

$$\pi_j^{\kappa+1} = \pi_j^\kappa = \overline{\Pi}_j(W^\kappa), \quad \forall\ j \in \mathcal{N}_{-r_\kappa}^N \tag{12}$$

We prove by contradiction. Let us assume that

$$\{\pi_1^{\kappa+1}, \pi_2^{\kappa+1}, \cdots, \pi_N^{\kappa+1}\} \neq \arg\min_{\pi_1,\pi_2,\cdots,\pi_N} \left(\sum_{i\in\mathcal{N}^N} c_i(\pi_i) + \sum_{\{ij\}\in\mathcal{P}^N} W_{ij}^{\kappa+1} \Omega_{ij}(\pi_i,\pi_j)\right)$$

This implies that there exist $\pi_1', \pi_2', \cdots, \pi_N'$ given by

$$\{\pi_1', \pi_2', \cdots, \pi_N'\} = \arg\min_{\pi_1,\pi_2,\cdots,\pi_N} \left(\sum_{i\in\mathcal{N}^N} c_i(\pi_i) + \sum_{\{ij\}\in\mathcal{P}^N} W_{ij}^{\kappa+1} \Omega_{ij}(\pi_i,\pi_j)\right)$$

such that,
$$\pi_i' = \overline{\Pi}_i(W^{\kappa+1}), \quad \forall i \in \mathcal{N}^N \tag{13}$$

and
$$\sum_{i\in\mathcal{N}^N} c_i(\pi_i^{\kappa+1}) + \sum_{\{ij\}\in\mathcal{P}^N} W_{ij}^{\kappa+1} \cdot \Omega_{ij}(\pi_i^{\kappa+1}, \pi_j^{\kappa+1}) > \sum_{i\in\mathcal{N}^N} c_i(\pi_i') + \sum_{\{ij\}\in\mathcal{P}^N} W_{ij}^{\kappa+1} \cdot \Omega_{ij}(\pi_i', \pi_j') \tag{14}$$

Again, by the algorithm, it holds that:

$$\pi_{r_\kappa}^{\kappa+1} = \arg\min_{\pi_{r_\kappa}} \left(c_{r_\kappa}(\pi_{r_\kappa}) + \sum_{\{ir_\kappa\}\in\mathcal{P}_{r_\kappa}^N} W_{ir_\kappa}^{\kappa+1} \cdot \Omega_{ir_\kappa}(\pi_i^\kappa, \pi_{r_\kappa})\right) \tag{15}$$

Noting that $\{ir_\kappa\} \in \mathcal{P}_{r_\kappa}^N \Rightarrow i \in \mathcal{N}_{-r_\kappa}^N$, and using equation (12), we get from (15),

$$\begin{aligned} &\pi_{r_\kappa}^{\kappa+1} = \Pi_{r_\kappa}(W^{\kappa+1}, W^\kappa) \\ \Rightarrow\ &\Psi_{r_\kappa}(W^{\kappa+1}, W^\kappa) \\ &\quad = c_{r_\kappa}(\pi_{r_\kappa}^{\kappa+1}) + \textstyle\sum_{\{ir_\kappa\}\in\mathcal{P}_{r_\kappa}^N} W_{ir_\kappa}^{\kappa+1} \cdot \Omega_{ir_\kappa}(\pi_i^{\kappa+1}, \pi_{r_\kappa}^{\kappa+1}) \end{aligned} \tag{16}$$

Also, from (13) and (6),

$$\pi_{r_\kappa}' = \overline{\Pi}_{r_\kappa}(W^{\kappa+1}) = \Pi_{r_\kappa}(W^{\kappa+1}, W^{\kappa+1})$$

Thus,

$$\Psi_{r_\kappa}(W^{\kappa+1}, W^{\kappa+1}) = c_{r_\kappa}(\pi_{r_\kappa}') + \sum_{\{ir_\kappa\}\in\mathcal{P}_{r_\kappa}^N} W_{ir_\kappa}^{\kappa+1} \cdot \Omega_{ir_\kappa}(\pi_i', \pi_{r_\kappa}') \tag{17}$$

Thus from (14), (16) and (17), upon rearrangement,

$$\begin{aligned} &\sum_{i\in\mathcal{N}_{-r_\kappa}^N} \left(c_i(\pi_i^{\kappa+1}) - c_i(\pi_i')\right) \\ &+ \sum_{\{ij\}\in\mathcal{P}^N/\mathcal{P}_{r_\kappa}^N} W_{ij}^{\kappa+1} \cdot \left(\Omega_{ij}(\pi_i^{\kappa+1}, \pi_j^{\kappa+1}) - \Omega_{ij}(\pi_i', \pi_j')\right) \\ &> \Psi_{r_\kappa}(W^{\kappa+1}, W^{\kappa+1}) - \Psi_{r_\kappa}(W^{\kappa+1}, W^\kappa) \end{aligned} \tag{18}$$

On the other hand, according to our inductive assumption, $\{\pi_1^\kappa, \pi_2^\kappa, \cdots, \pi_N^\kappa\}$ is a minimum for the global objective function with W^κ. Thus,

$$\begin{aligned} &\overline{U}(\{\pi_1^\kappa, \pi_2^\kappa, \cdots, \pi_N^\kappa\}, W^k) \\ &\quad \leq \overline{U}(\{\pi_1', \cdots, \pi_{r-1}', \Pi_r(W^\kappa, W^{\kappa+1}), \pi_{r+1}', \cdots, \pi_N'\}, W^k) \end{aligned} \tag{19}$$

Now, since V^κ is a *Separable Flow Direction Direction* according to our hypothesis, it holds that $W_{ij}^{\kappa+1} - W_{ij}^\kappa = V_{ij}^\kappa = 0\ \forall\{ij\} \in \mathcal{P}^N/\mathcal{P}_{r_\kappa}^N$. Again from (12) we have $\pi_j^{\kappa+1} = \pi_j^\kappa\ \forall j \in \mathcal{N}_{-r_\kappa}^N$. Using these, along with (11) and (13), we get from (19),

$$\begin{aligned} &\sum_{i\in\mathcal{N}_{-r_\kappa}^N} \left(c_i(\pi_i^{\kappa+1}) - c_i(\pi_i')\right) \\ &\quad + \sum_{\{ij\}\in\mathcal{P}^N/\mathcal{P}_{r_\kappa}^N} W_{ij}^{\kappa+1} \cdot \left(\Omega_{ij}(\pi_i^{\kappa+1}, \pi_j^{\kappa+1}) - \Omega_{ij}(\pi_i', \pi_j')\right) \\ &\quad \leq \Psi_{r_\kappa}(W^\kappa, W^{\kappa+1}) - \Psi_{r_\kappa}(W^\kappa, W^\kappa) \end{aligned} \tag{20}$$

Thus from (18) and (20),

$$\begin{aligned} &\Psi_{r_\kappa}(W^\kappa + \epsilon^\kappa V^\kappa, W^\kappa + \epsilon^\kappa V^\kappa) - \Psi_{r_\kappa}(W^\kappa + \epsilon^\kappa V^\kappa, W^\kappa) \\ &\quad - \Psi_{r_\kappa}(W^\kappa, W^\kappa + \epsilon^\kappa V^\kappa) + \Psi_{r_\kappa}(W^\kappa, W^\kappa) < 0 \end{aligned} \tag{21}$$

However this is a contradiction to our assumption that the V^κ and ϵ^κ defines a *Separable Optimal Flow* at W^κ for Ψ_{r_κ}. Hence our assumption of the existence of $\{\pi_1', \pi_2', \cdots, \pi_N'\}$ was incorrect. This proves Theorem 1.

B. Proof of Theorem 2

Since $\overline{U}(\{\pi\}, W)$ is linear in W, from [6] we know that $\overline{\Psi}(W) = \min_{\{\pi\}} \overline{U}(\{\pi\}, W)$ is concave in W. Again by optimality condition, $\overline{U}^{(1,0)}(\{\overline{\Pi}\}(W), W) = 0$

Thus,
$$\begin{aligned} &\overline{\Psi}(W) = \overline{U}(\{\overline{\Pi}\}(W), W) \\ \Rightarrow\ &\overline{\Psi}^{(1)}(W) = \overline{U}^{(0,1)}(\{\overline{\Pi}\}(W), W) \\ \Rightarrow\ &\left[\overline{\Psi}^{(1)}(W)\right]_{ij} = \Omega_{ij}(\overline{\Pi}_i(W), \overline{\Pi}_j(W)) \end{aligned} \tag{22}$$

Thus, if V^k is an *Ascent Direction*, from (10) and (22) it will imply that in every iteration we move towards a direction in which $\overline{\Psi}$ increases. On the other hand, $\overline{\Psi}$ being concave, can have only an unique optimum, which is a maximum, when $\left[\overline{\Psi}^{(1)}(W)\right]_{ij} = 0,\ \ \forall\{ij\} \in \mathcal{P}^N$. We thus note that the maximum, if it exists, is attained when all the constraints are satisfied. Hence taking steps towards the maximum means that we will eventually reach the feasible point, if it exists (within errors defined by the *Step Size*).

C. Proof of Theorem 3

From the definitions of U_r, Π_r and Ψ_r, by optimality condition, $U_r^{(1,0,0)}(\Pi_r(W_1, W_2), W_1, W_2) = 0$,

$$\begin{aligned} &\Rightarrow c_r^{(1)}(\Pi_r(W_1, W_2)) + \\ &\quad \textstyle\sum_{\{lr\}\in\mathcal{P}_r^N} W_{1,lr}\Omega_{lr}^{(0,1)}(\overline{\Pi}_l(W_2), \Pi_r(W_1, W_2)) = 0 \end{aligned} \tag{23}$$

Differenciating (23) w.r.t. W_1, using *Problem Assumption* 4.ii., setting $W_1 = W_2 = W$, and using (6) & (7),

$$\mathbf{M}_r(W) \cdot \left[\Pi_r^{(1,0)}(W, W)\right]_{ij} = \begin{cases} \Omega_{kr}^{(1,0)}(\overline{\Pi}_k(W), \overline{\Pi}_r(W)), & \text{when } \{i,j\} \equiv \{k,r\} \in \mathcal{P}_r^N \\ 0, & \text{when } i \neq r, j \neq r \end{cases} \tag{24}$$

Similarly, differentiating (23) w.r.t. W_2, followed by setting $W_1 = W_2 = W$, and using (6) & (7), we get,

$$\mathbf{M}_r(W) \cdot \left[\Pi_r^{(0,1)}(W,W)\right]_{ij} + \sum_{\{lr\}\in\mathcal{P}_r^N} W_{lr}\Omega_{lr}^{(1,1)}(\overline{\Pi}_l(W),\overline{\Pi}_r(W)) \cdot \left[\overline{\Pi}_l^{(1)}(W)\right]_{ij} = 0 \tag{25}$$

Adding (25) and (24), and using (7),

$$\mathbf{M}_r(W) \cdot \left[\overline{\Pi}_r^{(1)}(W)\right]_{ij} + \sum_{\{lr\}\in\mathcal{P}_r^N} W_{lr}\mathbf{N}_{lr}(W) \cdot \left[\overline{\Pi}_l^{(1)}(W)\right]_{ij} = \begin{cases} \Omega_{kr}^{(1,0)}(\overline{\Pi}_k(W),\overline{\Pi}_r(W)), & \text{when } \{i,j\}\equiv\{k,r\}\in\mathcal{P}_r^N \\ 0, & \text{when } i\neq r, j\neq r \end{cases} \tag{26}$$

Equation (26) can be written as follows,

$$\begin{bmatrix} \mathbf{M}_1(W) & W_{12}\mathbf{N}_{12}(W) & W_{13}\mathbf{N}_{13}(W) & \cdots \\ W_{21}\mathbf{N}_{21}(W) & \mathbf{M}_2(W) & W_{23}\mathbf{N}_{23}(W) & \cdots \\ W_{31}\mathbf{N}_{31}(W) & W_{32}\mathbf{N}_{32}(W) & \mathbf{M}_3(W) & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix} \begin{bmatrix} \left[\overline{\Pi}_1^{(1)}(W)\right]_{mn} \\ \left[\overline{\Pi}_2^{(1)}(W)\right]_{mn} \\ \left[\overline{\Pi}_3^{(1)}(W)\right]_{mn} \\ \vdots \end{bmatrix} = \begin{bmatrix} 0 & \\ \Omega_{mn}^{(1,0)}(\overline{\Pi}_m(W),\overline{\Pi}_n(W)) & \rightarrow m^{th} \text{ row} \\ 0 & \\ \cdots & \\ 0 & \\ \Omega_{mn}^{(1,0)}(\overline{\Pi}_n(W),\overline{\Pi}_m(W)) & \rightarrow n^{th} \text{ row} \\ 0 & \\ \cdots & \end{bmatrix} \tag{27}$$

For all $\{m,n\}\in\mathcal{P}^N$.

We note that equations (24) and (27) gives prescriptions for computing derivatives of Π_r and $\overline{\Pi}_r$ in terms of $\overline{\Pi}_r$ and the derivatives of c_r & Ω_{ij}. It is important to note that while the later are readily computable, the former aren't.

Again, differentiating (4) w.r.t. W_1 and using (23),

$$\left[\Psi_r^{(1,0)}(W_1,W_2)\right]_{ij} = \begin{cases} \Omega_{kr}(\overline{\Pi}_k(W_2),\Pi_r(W_1,W_2)), & \text{when } \{i,j\}\equiv\{k,r\}\in\mathcal{P}_r^N \\ 0, & \text{when } i\neq r, j\neq r \end{cases} \tag{28}$$

Differentiating (28) w.r.t. W_2, using *Problem Assumptions* 4.ii., followed by setting $W_1 = W_2 = W$, and using (6) & (7),

$$\left[\Psi_r^{(1,1)}(W,W)\right]_{ij,mn} = \begin{cases} \Omega_{kr}^{(1,0)}(\overline{\Pi}_k(W),\overline{\Pi}_r(W)) \cdot \left[\overline{\Pi}_k^{(1)}(W) - \overline{\Pi}_r^{(1)}(W)\right]_{mn} \\ \quad + \Omega_{kr}^{(1,0)}(\overline{\Pi}_k(W),\overline{\Pi}_r(W)) \cdot \left[\Pi_r^{(1,0)}(W,W)\right]_{mn}, \\ \qquad\qquad \text{when } \{i,j\}\equiv\{k,r\}\in\mathcal{P}_r^N \\ 0, \quad \text{when } i\neq r, j\neq r \end{cases} \tag{29}$$

Now using (24) and (27), we can compute $\left[\Psi_r^{(1,1)}(W,W)\right]_{ij,mn}$ in terms of the quantities mentioned in the statement of the theorem. It is important to note that the sole purpose of this dealing was to be able to express the derivatives of Π and $\overline{\Pi}$ in terms of computable quantities.

Once we obtain the complete matrix for $\Psi_r^{(1,1)}(W,W)$, we can choose a *Separable Optimal Flow Direction* as a linear combination of the right eigenvectors corresponding to the non-negative eigenvalues of $\Psi_r^{(1,1)}(W,W)$. Separable Optimal Flow Step, ϵ, is chosen according to a desired accuracy. Thus, if we choose $V = a_1\mathbf{e}_1 + a_2\mathbf{e}_2 + \cdots$, where $\mathbf{e}_1, \mathbf{e}_2, \cdots$ are the eigenvectors corresponding to non-negative eigenvalues $\lambda_1, \lambda_2, \cdots$, we get,

$$\begin{aligned} &\Psi_r(W+\epsilon V, W+\epsilon V) - \Psi_r(W, W+\epsilon V) \\ &\qquad - \Psi_r(W+\epsilon V, W) + \Psi_r(W,W) \\ &\simeq\ (\epsilon V)^T \left[\Psi_r^{(1,1)}(W,W)\right](\epsilon V) \\ &=\ \epsilon^2\ \ (a_1\mathbf{e}_1 + a_2\mathbf{e}_2 + \cdots)^T(a_1\lambda_1\mathbf{e}_1 + a_2\lambda_2\mathbf{e}_2 + \cdots) \\ &=\ \epsilon^2\ \ (\lambda_1 a_1^2 + \lambda_2 a_2^2 + \cdots)\ \ \geq\ 0 \end{aligned} \tag{30}$$

We note that all the V_{ij} for which $i\neq r, j\neq r$ does not influence the condition (30) since $\left[\Psi_r^{(1,1)}(W,W)\right]_{ij,mn} = 0$ when $i\neq r, j\neq r$. Thus we can always choose the V such that $V_{ij} = 0,\ i\neq r, j\neq r$. Thus, within the limits of the error introduced by the finite step-size, we can compute a *Separable Optimal Flow* if one exists. It is easy to see that if no positive real eigenvalue exists, then there does not exist a *Separable Optimal Flow* at W for that particular Ψ_r. Under such circumstances we can choose to skip to the next element in the sequence $\{r_k\}$ and retry.

ACKNOWLEDGMENT

We gratefully acknowledge support from ONR grant no. N00014-09-1-1052, NSF grant no. IIS-0427313, ARO grant no. W911NF-05-1-0219, ONR grants no. N00014-07-1-0829 and N00014-08-1-0696, and ARL grant no. W911NF-08-2-0004.

REFERENCES

[1] Ali Ahmadzadeh, Gilad Buchman, Peng Cheng, Ali Jadbabaie, Jim Keller, Vijay Kumar, and George Pappas. Cooperative control of uavs for search and coverage. *Proceedings of the AUVSI Conference on Unmanned Systems*, 2006.

[2] Ali Ahmadzadeh, James Keller, George J. Pappas, Ali Jadbabaie, and Vijay Kumar. Critical cooperative surveillance and coverage with unmanned aerial vehicles. In Vijay Kumar Oussamma Khatib and Daniela Rus, editors, *International Symposium on Experimental Robotics*, STAR, Rio de Janeiro, July 2006. Springer-Verlag.

[3] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, 2007.

[4] Dimitri P. Bertsekas and John N. Tsitsiklis. *Parallel and Distributed Computation:Numerical Methods*. Prentice Hall, 1989.

[5] Subhrajit Bhattacharya, Maxim Likhachev, and Vijay Kumar. Distributed path consensus algorithm. Technical Report MS-CIS-10-07, University of Pennsylvania, 2010.

[6] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2009.

[7] J. Desai, J. Ostrowski, and V. Kumar. Modeling and control of formations of nonholonomic mobile robots. *IEEE Transactions on Robotics and Automation*, 17(6):905–908, December 2001.

[8] M Bernardine Dias, Robert Michael Zlot, Nidhi Kalra, and Anthony (Tony) Stentz. Market-based multirobot coordination: a survey and analysis. *Proc. of the IEEE*, 94(7):1257 – 1270, 2006.

[9] Edmund H. Durfee. Distributed problem solving and planning. *Lecture Notes in Computer Science*, 2086(2):118–149, 2001.

[10] Brian Gerkey and Maja Mataric. A formal analysis and taxonomy of task allocation in multi-robot systems. *Int'l. J. of Robotics Research*, 23(9):939–954, 2004.

[11] D. Grigoriev and A. Slissenko. Polytime algorithm for the shortest path in a homotopy class amidst semi-algebraic obstacles in the plane. In *ISSAC '98: Proceedings of the 1998 international symposium on Symbolic and algebraic computation*, pages 17–24, New York, NY, USA, 1998. ACM.

[12] Savvas G. Loizou and Kostas J. Kyriakopoulos. Closed loop navigation for multiple holonomic vehicles. In *Proc. of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pages 2861–2866, 2002.

[13] Diego A. Murio. *The Mollification Method and the Numerical Solution of Ill-Posed Problems*. Wiley-Interscience, 1993.

[14] I. Necoara and J.A.K. Suykens. Interior-point lagrangian decomposition method for separable convex optimization. *Journal of Optimization Theory and Applications*, 143:567–588, 2009.

[15] Sikandar Samar, Stephen Boyd, and Dimitry Gorinevsky. Distributed estimation via dual decomposition. 2007.

[16] H. Zhang, V. Kumar, and J. Ostrowski. Motion planning under uncertainty. In *IEEE International Conference on Robotics and Automation*, Leuven, Belgium, May 16-21 1998. IEEE.