Multi-agent Path Planning with Multiple Tasks and Distance Constraints

Subhrajit Bhattacharya, Maxim Likhachev and Vijay Kumar

Abstract— **The DPC algorithm developed in our previous work is an efficient way of computing optimal trajectories for multiple robots in a distributed fashion with time-parameterized constraints on the distances between pairs of robots. In the present work we extend DPC to the problem of multiple task execution. While this extended problem inherits all the objectives, complexities and constraints of the basic DPC algorithm, each robot is also given an unordered set of tasks that it has to execute before it reaches its goal. There is no specific order imposed on the tasks assigned to a particular robot. The algorithm decides the order of execution of the tasks such that an optimal solution is attained while the time-parametrized distance constraints are satisfied along with successful execution of the tasks. We solve this problem by designing a "State-task Graph" that represents a product of the state-space graph and the task graph. We then develop an efficient heuristic function for performing searches in this graph.**

I. Introduction

Path planning and coordination among multiple mobile robots in environments with obstacles is a challenging problem in robotics. Planning for goal-directed navigation is often modeled as computing a least-cost path through a graph generated by discretization of the environment [14]. However in multi-robot problems, for obtaining an optimal solution, one needs to typically plan in the product of the state-spaces of the robots (i.e. the joint state-space or the full configuration space). The increase in the dimensionality of the configuration space increases the computational expense exponentially with the number of robots. Having intermediate tasks that the robots need to execute increases the complexity of the problem even further.

The DPC algorithm developed in our previous work [5] is an efficient way of solving the problem of multi-robot path planning under constraints. The problem consists of path planning for a team of robots with time-parameterized constraints on the distances between pairs of robots. Both communication constraints and rendezvous constraints can be modeled in this form. In order to enable scalability, the DPC algorithm performs planning in a decentralized fashion, iterating through each robot planning in its own configuration space, while gradually increasing the penalty due to violation of constraints. A typical application area is search and/or coverage in settings where robots must rendezvous periodically to exchange information and/or maintain relative distance constraints to enable communication. Continuous motion planning is possible for such problems [2], but only practical in environments with limited complexity. Thus, in the DPC algorithm, we took a discrete path planning approach for multiple robots with time-parametrized constraints on the distance between the robots. The DPC algorithm decomposes the one-shot joint state-space planning into planning with a series of lower-dimensional searches converging to an optimal solution. Such decomposition of high dimensional problems has been studied in [9] using sequential planning for each agent, but not guaranteeing optimality. The basic idea in the DPC algorithm is to incorporate constraints through penalty functions in a fashion that is reminiscent of augmented Lagrangian techniques [4]. Similar iterative and distributed techniques have been investigated in the past [11], but for much simpler forms of constraints, mostly collision avoidance. The DPC algorithm is provably complete and returns provably optimal solution when required conditions are satisfied [5].

S. Bhattacharya is a doctoral student with the Department of Mechanical Engineering and Applied Mechanics, University of Pennsylvania, Philadelphia, PA 19104 subhrabh@seas.upenn.edu

M. Likhachev is a professor with the Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA 19104 maximl@seas.upenn.edu

V. Kumar is a professor with the Department of Mechanical Engineering and Applied Mechanics, University of Pennsylvania, Philadelphia, PA 19104 kumar@me.upenn.edu

A. Motivation

An important extension of the basic constrained planning problem is the inclusion of multiple tasks along the paths of the robots. This aspect of the problem is inspired by the problem of multi-robot mapping of an unknown or partially known environment, where the robots need to visit certain points in the environment (the tasks) to explore/map those regions, and also periodically meet & communicate their respective findings with each other in order to build a global map of the environment in a distributed fashion [15]. Assuming that each robot has been assigned an unordered list of tasks in forms of coordinates in space, in this paper we investigate how we can make use of the basic concept of the DPC algorithm to enable the robots execute the tasks, but still satisfy all the time-parametrized distance constraints and ensure optimality of the solution. Essentially the tasks act as additional constraints in the original DPC planning problem.

B. Related work

Robot path planning is probably one of the most extensively studied problems areas in robotics [14]. Multi-robot path planning suffers from the inherent complexity resulting from the necessity of operating in Cartesian products of configuration and state spaces [7]. Broadly speaking path planning algorithms can be divided into two categories: continuous and discrete. The continuous path planning problem is difficult to solve in a centralized setting [21], [16] unless the problem is solved sequentially for each robot [22]. While completeness results are often possible [1], it is difficult to respect multi-robot constraints and to establish completeness and convergence results except in special cases. In most practical settings, discrete graph search methods have been shown to be complete and efficient [20].

The order of execution of assigned tasks (or multiple goals) has been studied using decision theoretic approaches [10] in cases where the tasks have priorities or weights attached to them. However in our present problem all the tasks are considered to be of equal importance and the order is solely chosen to minimize the trajectory cost and to satisfy the constraints. Our approach is to integrate the execution order of tasks as decision variables in the core planning problem so as to minimize the original objectives.

1) The task allocation problem: One of the first assumptions that we start with in the present work is that each robot has already been assigned an unordered set of tasks that it needs to execute. The problem of task allocation deals with assigning the tasks to the the robots. Task allocation for mobile robots has been extensively studied in the past. Finding an optimal allocation is strongly NP-hard [17]. Since typically the total number of available tasks is small, integer and linear programming approaches have been effectively used to solve the problem [3], [6]. In similar lines, auction-based and market-based approaches have been used extensively [23], [13], [8] to improve the task completion time and reduce the communication overheads. The problem can also be translated into a version of the traveling salesman problem (TSP) with the robots being represented by multiple salesmen following paths instead of tours [19]. Distributed algorithms using Voronoi diagrams have been implemented for assigning tasks to multiple agents [18]. Thus the problem of allocating the tasks to multiple robots has been well-studied, and hence is not a focus of the present work. However typically these solutions don't take into consideration the other constraints on the robot trajectories, such as time-parametrized distance constraints in our case. As we'll see in the first example under the "Results" section, the DPCT algorithm proposed in the present paper considers different orders of execution of the assigned tasks to ensure optimality.

II. Problem Definition

The problem consists of path planning and execution of assigned tasks for a team of N robots with time-parameterized constraints on the distances between pairs of robots. This particular type of constraint has a broad scope in multi-robot coordination problems, which includes, but is not limited to, the problem of rendezvousing in order to exchange information and the problem of maintaining communication while executing tasks. In our problem each robot is given a goal coordinate. The constraints are defined between pairs of robots and are modeled as the minimum distance of separation between the robots as a function of time.

Our approach towards solving the problem is to make sure that we get an optimal or near-optimal solution without needing to perform a search in a joint state-space of the robots. We use an iterative approach using soft constraints to attain this. For just a single robot we can construct a search graph by discretizing space and time and use A* algorithm to plan an optimal path to the goal. When there are multiple tasks for the robot, the search graph can be extended by adding an additional dimension for keeping track of the tasks. In case of a multi-robot scenario we start off by planning the unconstrained optimal paths for each robot, and then gradually increase the weight on the penalty for violating the constraints, as we keep on iterating among the robots, with each robot planning paths that minimize the sum of its path's cost and the weighted penalties for the constraints that it violates. The idea is to make the robots adjust their trajectories gradually in order to satisfy the constraints so that optimality is achieved. The benefit of such approach is that it allows us to complete planning and reach an optimal solution that satisfies all the constraints, without having to perform planning in the joint state-space of all the robots. We have also shown that the algorithm, under certain conditions on our choice of incrementing the penalty weights and the choice of the cost functions, is complete and will return an optimal solution [5]. The tasks are in the form of coordinates in space that the robots need to visit. However there is no specific order imposed on the tasks assigned to a particular robot.

A. Graph Construction

Planning for goal-directed navigation for an individual robot $R_i, 1 \leq i \leq N$ is often modeled as computing a least-cost path through a graph G_i formed by discretization of the configuration space. Each state $\mathbf{s} \in \mathbf{V}(G_i)$ is given by $\{x, y\}$ coordinates of the corresponding cell. For permissible and neighboring states, $\mathbf{s}$ and $\mathbf{s}'$ (free states inside the configuration space) the edge $\mathbf{s} \rightarrow \mathbf{s}' \in \mathbf{E}(G_i)$ is associated with a strictly positive cost $c(\mathbf{s}, \mathbf{s}')$. A common choice for the costs is the Euclidean distances in between the centers of the corresponding cells.

In addition we augment each state in the graph G_i with an additional variable - time index t, such that a state in the augmented graph becomes $\{\mathbf{s}, t\}$. Edges in this augmented graph (a directed graph) $H_i = G_i \times \{0, 1, \cdots, T\}$ are defined such that a particular state $\{\mathbf{s}, t\}$ connects only to the states $\{\mathbf{s}', t+1\}$, such that the state $\mathbf{s}' \in \mathbf{V}(G_i)$ is either same as $\mathbf{s}$, or $\mathbf{s} \rightarrow \mathbf{s}' \in \mathbf{E}(G_i)$. Planning in such an H_i ensures that the planning is done both in space and time, and the time parametrized trajectory returned by the planner is consistent with the fact that the robots can move only forward in time. Such planning enables us to incorporate time-paramertized distance constraints.

We assume that all trajectories of interest to us are at most T timesteps. Thus, the solution to a typical planning problem for a single robot is a T-step path in H_i from $\{Start_i, 0\}$ to $\{Goal_i, T\}$, and can be represented by the ordered set $\pi_i = \{s_0 = Start_i, s_1, \ldots, s_T = Goal_i\}$. Thus, $\pi_i(t)$ refers to the coordinate s_t (in other words, the robot R_i is at this location at time t when following path π_i). The cost of a path is given as $c(\pi_i) = \sum_{j=1\dots T} c(\mathbf{s}_{j-1}, \mathbf{s}_j)$.

B. Introducing the Tasks

The robot R_i has M_i tasks assigned to it denoted by $\tau_i^0, \tau_i^1, \cdots, \tau_i^{M_i-1}$, where each task is a coordinate in space, and more specifically is a node in the graph G_i. Thus, $\tau_i^j = (x_i^j, y_i^j) \in \mathbf{V}(G_i)$. The planning needs to be done in such a way that the planned trajectory of robot R_i passes through each of $\tau_i^0, \tau_i^1, \cdots, \tau_i^{M_i-1}$, i.e. $\tau_i^j \in \pi_i \forall j = 0, 1, 2, \cdots, M_i - 1$. The order of execution of the tasks is obtained as part of the solution to the planning problem.

To model the tasks and in order to incorporate them in the search graph we first define a fourth state coordinate (after x, y and t) and call it "task indicator", β, which is essentially a variable that we will represent as a binary number consisting of M_i bits (for robot R_i), each bit being the flag or indicator of whether the corresponding task has been completed. For notational convenience we define a function B such that $\beta = B_M(\{j_1, j_2, \cdots, j_k\})$ is a M-bit long binary number with 1's at the positions $j_1, j_2, \cdots, j_k$, and 0's at the rest. Thus $B_7(\{2, 4, 5\}) = 0110100$ and $B_3(\{0\}) = 001$. Note that $B_M(\{j_1, j_2, \cdots, j_k\})$ in fact represents the state in which only the tasks $\tau^{j_1}, \tau^{j_2}, \cdots, \tau^{j_k}$ have been executed. We define the inverse function of B to be the one that returns the indices of the non-zero bits in a given β, i.e., $B^{-1}(\beta) = \{j_1, j_2, \cdots, j_k\}$

We define a task graph Υ_i for robot R_i such that the vertices of Υ_i are M_i-bit binary numbers. Connection between two vertices of Υ_i exist iff the vertices differ by exactly one bit, and the edge points from the vertex with lower value to the one with higher value. An example of such a graph for $M_i = 4$ is shown in Figure 1. For a robot R_i with M_i tasks to execute, the starting value of "task indicator" will be $B_{M_i}(\{\})$ which represents the state where no task has been executed, and its goal value will be $B_{M_i}(\{0, 1, 2, \cdots, M_i - 1\})$ which represents the state where all the tasks have been executed. Since the tasks are executed one at a time, the two vertices β_1 and β_2 can be connected if and only if β_1 and β_2 differ by a single bit.

However we also note that the j_l^{th} bit in the state coordinate β is to be turned on for robot R_i only when the robot executes the j_l^{th} task assigned to it, i.e. when the spatial coordinate of the robot is $s = \tau_i^{j_l}$. Using this fact we construct the State-task Graph K_i for robot R_i such that $\{\mathbf{s}, t, \beta\} \in \mathbf{V}(K_i)$ are the vertices of the graph representing the full states of the robot. The graph K_i is defined such that,

i. $$\begin{aligned} \mathbf{V}(K_i) &= \mathbf{V}(H_i) \times \mathbf{V}(\Upsilon_i) \\ &= \mathbf{V}(G_i) \times \{0, 1, \cdots, T\} \times \mathbf{V}(\Upsilon_i) \end{aligned}$$

ii. An edge from vertex $\kappa_1 = \{\mathbf{s}_1, t_1, \beta_1\} \in \mathbf{V}(K_i)$ to vertex $\kappa_2 = \{\mathbf{s}_2, t_2, \beta_2\} \in \mathbf{V}(K_i)$ exists iff exactly one of the following holds

a. $\{\mathbf{s}_1, t_1\} \rightarrow \{\mathbf{s}_2, t_2\} \in \mathbf{E}(H_i)$,
and $\mathbf{s}_2 \notin \{\tau_i^l |\ l^{th}$ bit of β_1 is $0\}$,
and $\beta_1 = \beta_2$.

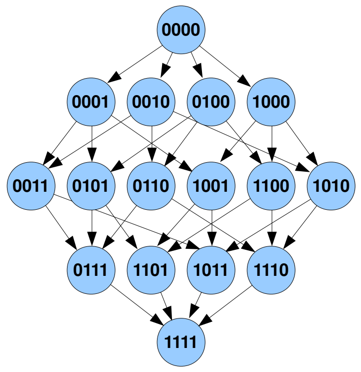


Fig. 1. The task graph Υ_i showing the possible transitions of the task indicator value, β for robot R_i with four tasks

b. $\{\mathbf{s}_1, t_1\} \rightarrow \{\mathbf{s}_2, t_2\} \in \mathbf{E}(H_i)$,
and $\mathbf{s}_2 \in \{\tau_i^l |\ l^{th}$ bit of β_1 is $0\}$ with $\mathbf{s}_2 = \tau_i^\lambda$,
and $\beta_1 \rightarrow \beta_2 \in \mathbf{V}(\Upsilon_i)$ such that the λ^{th} bit of β_2 is 1.

The cost of an edge $\{\mathbf{s}_1, t_1, \beta_1\} \rightarrow \{\mathbf{s}_2, t_2, \beta_2\} \in \mathbf{E}(K_i)$ is same as the cost of the edge $\{\mathbf{s}_1, t_1\} \rightarrow \{\mathbf{s}_2, t_2\} \in \mathbf{E}(H_i)$

Any possible state of the robot R_i defined by its spatial coordinates, time and tasks executed can be represented by a state in the graph K_i. The connection between the states of K_i define how the transition from one state to another can take place.

III. Algorithm

A. Graph Search

1) The constraints: We represent constraints between pair of robots, R_i and R_j, as time parametrized functions of maximum distance between them. For any pair of states $\mathbf{s} \in \mathbf{V}(G_i)$, $\mathbf{s}' \in \mathbf{V}(G_j)$, where $i \neq j$, we define a distance $d(\mathbf{s}, \mathbf{s}')$ as a non-negative finite scalar-valued function satisfying commutativity (e.g., $d(\mathbf{s}, \mathbf{s}') = d(\mathbf{s}', \mathbf{s})$). Thus, in case graphs G_i were derived from a 2D grid-world, the distance function $d(\mathbf{s}, \mathbf{s}')$ can be a simple Euclidean distance in between the centers of the cells that correspond to state $\mathbf{s}$ for the first robot and state $\mathbf{s}'$ for the second robot. In other cases, the distance function can model more complex factors.

We thus specify time-parameterized distance constraints between all pairs of the robots $\phi_{i,j}$ for all $i \neq j$. Thus, $\phi_{i,j}$ is a vector of T non-negative scalar values such that $\phi_{i,j}(t)$ implies that the distance $d(\cdot, \cdot)$ in between robots R_i and R_j at time t should be no more than $\phi_{i,j}(t)$. The $\phi_{i,j}(t) = \infty$ therefore implies the absence of any constraint in between these robots at time t.

2) Objective function: Given our formulation of the problem, the goal of an optimal planning algorithm would be to find N paths π_i^* $(1 \leq i \leq N)$ through the corresponding graphs G_i such that:

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

subject to the constraint that

$$d(\pi_i^*(t), \pi_j^*(t)) \leq \phi_{i,j}(t) \qquad (2)$$
$$\forall\ 1 \leq t \leq T, 1 \leq i \leq N, 1 \leq j \leq N, i \neq j$$

and

$$\tau_i^j \in \pi_i \qquad \forall\ j = 1, 2, \cdots, M_i, \text{ and } 1 \leq i \leq N$$

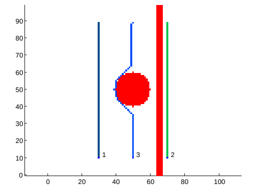
3) Search in the State-task Graphs: We can formulate the above mentioned problem equivalently as a search in the product of the N State-task Graphs $[K_1, K_2, \cdots, K_N]$ from the initial state of $[\{Start_1, 0, B_{M_1}(\{\})\}, \cdots, \{Start_N, 0, B_{M_N}(\{\})\}]$ to the final state of $[\{Goal_1, T, B_{M_1}(\{0, 1, \cdots, M_1 - 1\})\}, \cdots, \{Goal_N, T, B_{M_N}(\{0, 1, \cdots, M_N - 1\})\}]$ such that (2) is satisfied (Note that π_i^* is readily obtained from the states in the State-task Graph K_i by noting that $\pi_i^*(t)$ corresponds to s in the solution state $\{\mathbf{s}, t, \beta\} \in K_i$)

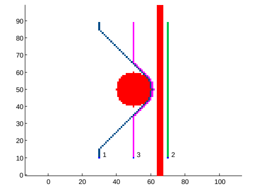
As it will be described in the next sections, instead of planning in the product space of $K_1 \times K_2 \times \cdots \times K_N$, the approach we take in the DPC algorithm is to plan in each of K_i in an iterative fashion with the constraints in (2) modeled as soft constraints, and then gradually increasing the penalty weights on the constraint violations.

B. DPC Algorithm with multiple tasks

The original DPC Algorithm is explained in greater details at [5]. We have theoretical proof of convergence, completeness and optimality of the algorithm for the case of an empty environment. The algorithm is an efficient way of computing optimal paths for multiple heterogeneous robots under constraints. The constraints we have studied mostly consist of time-parametrized distance constraints, but the constrains between pairs of robots can be more varied and complex. Optimal planning in the joint state-space of N robots with complex constraints is computationally prohibitive. The DPC algorithm was implemented on real robots platforms called Scarabs. Figure 2 shows the trajectories produced by DPC on a particular problem (more details in [5]) being executed on scarab robots. In the following section we extend the algorithm to incorporate the tasks into the search graph. We call this algorithm DPCT (DPC with tasks).

1) DPCT Algorithm: The basic idea behind the DPCT algorithm, very similar to the original DPC algorithm, is to run a series of graph searches on graphs K_i, $1 \leq i \leq N$. At each iteration $iter$, the search processes some graph K_r and computes a path that minimizes the weighted sum of the path-cost plus the amount to which the path violates the constraints with respect to the paths computed for other robots previously. This makes the robots to increase their path-costs slowly and converge to a good, often optimal, solution. We have theoretical proof of completeness, convergence and optimality for environments without obstacles [5]. In an environment with obstacles the optimality is not theoretically guaranteed, but as we will discuss later, we can explore the various homotopy classes using a *Blacklist*, with optimality being guaranteed within each homotopy class, and hence obtain an optimal solution most of the time.


(a) Unconstrained trajectories


(b) Solution satisfying constraints

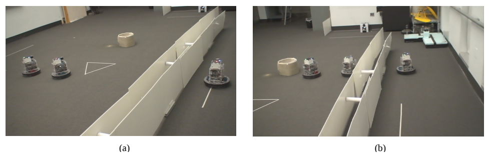

(c) Plan execution on Scarab robot platforms

Fig. 2. Screenshots of execution of the plan obtained from the DPC algorithm on Scarab robot platforms

```
1  procedure BasicDPCT()
2  compute $\pi_i^0 = \text{argmin}_{\pi_i} c(\pi_i)$ for all $1 \leq i \leq N$ by performing
           search through $K_i$;
3  set $w_{i,j}^0(t) = 0$ for all $i, j, t$;
4  $r = 1$, $iter = 0$;
5  while ($\sum_{i=1 \ldots N} \sum_{j=i+1 \ldots N} \Omega(\pi_i^{iter}, \pi_j^{iter}) \neq 0$ AND
           $min_{i,j} w_{i,j}^{iter} \cdot \Omega(\pi_i^{iter}, \pi_j^{iter}) \leq 2 * maxpathcost$)
6      ↪ set $w_{i,j}^{iter+1}(t) = w_{i,j}^{iter}(t)$ for all $i, j, t$;
7      ↪ $w_{r,j}^{iter+1} = w_{j,r}^{iter+1} = w_{j,r}^{iter} + \epsilon$ for all $j, t$;
8      ↪ compute $\pi_r^{iter+1} = \text{argmin}_{\pi_r} \{c(\pi_r) +$
               $\sum_{j=1 \ldots N, j \neq r} w_{r,j}^{iter+1} \cdot \Omega(\pi_r, \pi_j^{iter})\}$
               by performing search through $K_i$
9      ↪ set $\pi_j^{iter+1} = \pi_j^{iter}$ for all other $j \neq r$;
10     ↪ $iter = iter + 1$;
11     ↪ $r = r + 1$;
12     ↪ if $r > N$
13         ↪ $r = 1$;
```

Fig. 3. Basic DPCT

To penalize for the violation of constraints, the algorithm introduces the penalty function $\Omega(\pi_i, \pi_j)$, $i \neq j$ as follows:

$$\Omega(\pi_i, \pi_j) = \sum_{t=0,1,\cdots,T} \varpi(\pi_i(t), \pi_j(t), \phi_{i,j}(t))$$
$$\text{where, } \varpi(\mathbf{s}, \mathbf{s}', p) = \max(0, d(\mathbf{s}, \mathbf{s}') - p) \qquad (3)$$

Note that Ω depends on $\phi_{i,j}$ as well. However since typically $\phi_{i,j}$ is a constant throughout the problem (i.e. does not change with the iterations), for notational simplicity we use $\Omega(\pi_i, \pi_j, \phi_{i,j}) \equiv \Omega_{ij}(\pi_i, \pi_j) \equiv \Omega(\pi_i, \pi_j)$. The convention is that we look at the indices of the π's to identify which ϕ to use for defining Ω.

The penalty function is a way of transforming the hard distance constraints into soft constraints. To gradually increase the weight of each constraint violation, each constraint $\phi_{i,j}$ is associated with a dynamically adjusted weight w_{ij} (the weights have to be symmetric, so $w_{ij} = w_{ji}$).

The pseudocode of the algorithm is shown in Figure 3. It first computes unconstrained least-cost paths for each of the robot (line 2). The computation of the paths can be done with any graph search such as A* search [12]. After that, the algorithm iterates over the robots repeatedly (robot index is incremented on lines 11-13). Within each iteration $iter$, the algorithm increases the penalty weights for the selected robot R_r by a small increment ϵ (line 7), and computes a path for the robot R_r from its start state $\{Start_r, 0, 00\cdots 0\}$ to its goal state $\{Goal_r, T, 11\cdots 1\}$ in graph K_r that minimizes the summation of all the transition costs and all the weighted penalties (line 8). To compute this path, one can once again use any graph search for a least-cost path, but the costs of all the edges, however, need to be modified. In particular, the cost of any transition $\{\mathbf{s}, t-1, \beta\} \rightarrow \{\mathbf{s}', t, \beta'\}$ becomes:

$$c(s, s') + \textstyle\sum_{j=1\dots N, j\neq r} w_{rj}^{iter+1} \varpi(s', \pi_j^{iter}(t), \phi_{r,j}(t))$$

2) DPCT with Superiterations: In cluttered environments, DPCT needs to eleminate infeasible homotopy classes of trajectories in an efficient fashion so that we are able to arrive at the optimal solution without needing to increase the penalty weights arbitrarily for switching from one homotopy class of trajectories to another. This is done by maintaining and populating a *blacklist* of pairwise configurations that violate the constraints. DPCT with Superiterations is in essence the very same as the DPC with Superiterations in [5]. The DPCT algorithm maintains the *blacklist*, adding 4-D blocked balls to it after every superiteration that fails to return a solution, and avoiding the blocked regions in the searches of the following superiterations. For more details see [5].

3) Heuristic: The choice of the heuristic function, h, is extremely crucial in any A* search. A heuristic function is a positive scalar function of the states in the search graph. For A* to return an optimal solution, the heuristic function needs to be such that it never overestimates the actual minimum cost for reaching the goal. However to make the search more efficient and minimize the number of states expanded, the heuristic must be as close as possible to the actual minimum cost to the goal. One obvious and commonly used heuristic for planning in G_i is the Euclidean distance to the goal, i.e. $h_E(\mathbf{s}) = \|\mathbf{s} - Goal\|$. But for an 8-connected grid, with costs defined by Euclidean distances between centers of cells, which G_i is, one can use a more efficient heuristic given by $h_8(\mathbf{s}) = \sqrt{2}\min(\Delta x, \Delta y) + |\Delta x - \Delta y|$, where $\Delta x = |\mathbf{s}_x - Goal_x|$ and $\Delta y = |\mathbf{s}_y - Goal_y|$.

However in a highly cluttered environment even h_8 turns can be highly inefficient. In such a case, before executing the actual planning, we perform a Dijkstra's search in G_i starting from the goal coordinate $Goal$ till we expand all the reachable states in G_i. Let $D_{Goal}(\mathbf{s})$ be cost-to-goal computed by Dijkstra's for each state $\mathbf{s} \in G_i$. Although the process of Dijkstra search is itself more expensive than A* search in G_i, the advantage of this pre-computation becomes clear when we attempt to plan in higher dimensional graphs H_i or K_i. Once D is precomputed for all the states in G_i, while planning in H_i we can simply use the more efficient heuristic function, $h_H(\{\mathbf{s}, t\}) = D_{Goal}(\mathbf{s})$.

But when we have tasks to execute before reaching the final goal, we can design even a more informative heuristic. Since we know that the robot will be visiting all the task locations, the heuristic associated with a particular state $\{\mathbf{s}, t, \beta\}$ can be defined as follows (note that the index of the robot, i, is dropped all throughout for notational convenience),

$$\begin{aligned} h_K(\{\mathbf{s}, t, \beta\}) \;=\; & \min_{\langle j_1, j_2, \cdots, j_l\rangle \in Perm(B^{-1}(\sim\beta))} \Big[D_{\tau^{j_1}}(\mathbf{s}) + \\ & D_{\tau^{j_2}}(\tau^{j_1}) + D_{\tau^{j_3}}(\tau^{j_2}) + \cdots + D_{\mathbf{s}_{goal}}(\tau^{j_l})\Big] \end{aligned} \tag{4}$$

where $\sim\beta$ is the binary number obtained by flipping all the bits of β (i.e. $B^{-1}(\sim\beta)$ is the list of indices of the remaining tasks), l is the number of remaining tasks, i.e. $l = n(B^{-1}(\sim\beta))$, $Perm(\cdot)$ returns a list of ordered sets with all the possible permutation of elements passed to it, and $\langle\cdot\rangle$ represents an ordered set.

The above heuristic function basically looks for all the possible orders in which the tasks can be visited, and choses the one with the minimum cost. Note that computing $D_{\tau^{j_a}}(\mathbf{s})$ requires that we run Dijkstra's search for each of the states $\tau^1, \tau^2, \cdots, \tau^M$. However we note that even without the knowledge of $\mathbf{s}$ we can precompute

$$\begin{aligned} L(\beta, j_1) \;=\; & \min_{\langle j_2, \cdots, j_l\rangle \in Perm(B^{-1}(\sim\beta \backslash j_1))} \Big[D_{\tau^{j_2}}(\tau^{j_1}) + \\ & D_{\tau^{j_3}}(\tau^{j_2}) + \cdots + D_{\mathbf{s}_{goal}}(\tau^{j_l})\Big] \end{aligned} \tag{5}$$

Then during run-time we have a quick way of computing the heuristic for the vertices in the State-task graph,

$$h_K(\{\mathbf{s}, t, \beta\}) \;=\; \min_{j \in B^{-1}(\sim\beta)} \left[D_{\tau^j}(\mathbf{s}) + L(\beta, j)\right] \tag{6}$$

IV. Results

Although the core essence of our algorithm lies in the fact that it can be implemented in a decentralized fashion, in all the following simulations the implementation was done on a single computer with a 2GHz processor and 4MB RAM. The decentralized implementation of the DCPT algorithm on real hardware is in progress and will make the computation much faster, reducing the computation time N folds. All implementation is done in C++, and MATLAB is used for visualization of output data. [1]

All the environments in the examples are two dimensional and are 8-connected grids. The algorithm can be easily generalized for three dimensional environments with more complex discretization. The robots can be heterogeneous (*e.g.* ground veichles as well as areal vehicles, and with different speed limits) and hence have different environment maps and different graph connectivities.

We also performed experiments on real robot platform called Scarab (Figure 2(c)). In order to account for the

[1]The output files corresponding to the examples/results in the paper can be found at http://www.seas.upenn.edu/∼subhrabh/nonWebsite/IterPlanning/index.html.

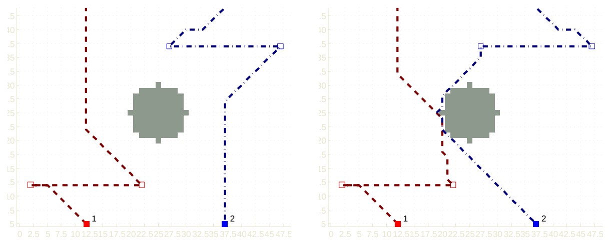

(a) Unconstrained trajectories (b) Solution satisfying constraints

Fig. 4. Planning for two robots each with two tasks and a constraint to meet during their travel. It is interesting to note how in the final converged solution the robot on the right switches the order of execution of its tasks in order to satisfy the imposed constraint.

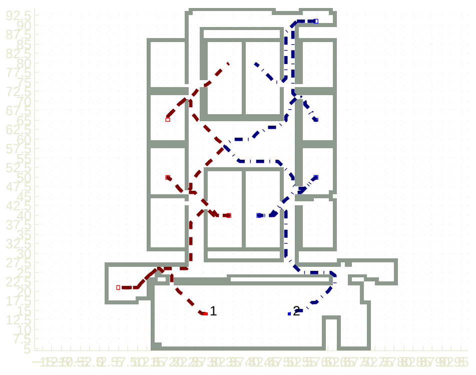


Fig. 5. Two robots exploring certain rooms and rendezvousing to exchange information.

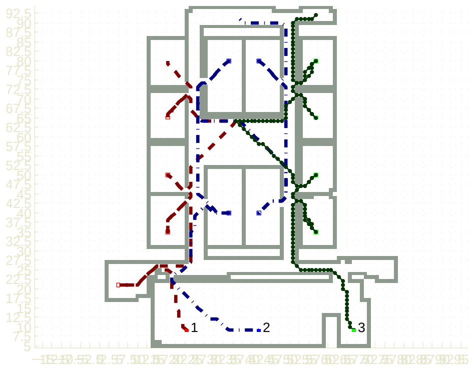


Fig. 6. Three robots exploring certain rooms and rendezvousing to exchange information.

non-zero radii of the robots, we performed a greedy collision avoidance during run-time. For controlling the non-holonomic robots along the planned trajectories, a feedback linearization technique was adopted.

A. *Example of reordering of tasks to satisfy constraints*

In the simple example in Figure 4 we have two robots having to plan their paths from a start to a goal location. The environment contains one central large obstacle. Each of the robots has been assigned two tasks (marked by the empty boxes in color). The first figure shows the case where there is no constraint between the robots. Now we impose a constraint that the robots need to be within a distance of $0.08m$ at $t = 17.5s$ during their journey. It is interesting to note how in the final converged solution the second robot (the one in blue) switches the order of execution of its tasks in order to satisfy the imposed constraint. The environment consisted of 50 discretizations along each spatial direction and 40 discretizations in time. The joint state-space of the robots would hence have $40 \times (50 \times 50 \times 2^2)^2 = 4$ billion states. Even in this simple environment the constrained optimal planning in joint state-space would become very difficult. However our algorithm converges to a solution in about 10 iterations and in less than a minute time. The converged solution is optimal with respect to an 8-connected grid.

B. *Exploration*

Consider the example in Figure 5 where each robot needs to explore the inside of certain rooms in the 4th floor of Levine hall (University of Pennsylvania) assigned to them in the environment, and possibly create a map of the environment. They need to meet intermediately at $t = 120s$ to exchange information about each other's explorations so as to build a global map in a decentralized fashion. This problem perfectly fits our paradigm. We see how the robots explore the assigned rooms and also meet to exchange information. Each robot has been assigned 4 tasks. The environment consisted of 100 discretizations along each spatial direction and 250 discretizations in time. This makes a total of $250 \times (100 \times 100 \times 2^4)^2 = 6.4 \times 10^{12} = 6.4$ trillion states in the joint state-space! Our algorithm finds the solution in 1311 seconds in 17 iterations. The solution is optimal with respect to an 8-connected grid.

C. *Exploration with three robots*

We once again do the exploration of the Levine 4th floor, but now with 3 robots. Figure 6 shows the result. The constraints are between pairs of robots. Thus we used two constraints: The first one between robots 1 and 2 is to meet at $t = 120s$, and second one between robots 2 and 3 is to meet at $t = 120s$. The joint state-space had $250 \times (100 \times 100 \times 2^4)^3 = 1.024 \times 10^{18}$ states. Our algorithm finds a solution to the problem in 3003 seconds and 40 iterations.

D. *Computation time statistics*

We chose a relatively simple scenario with two robots navigating from start to goal configuration, one task assigned to each robot, and one rendezvous constraint (Figure 7). The environment was same as before (Levine hall 4th floor), with 100×100 spatial discretization and 150 temporal discretization. We ran our algorithm several time by randomizing the initial positions, goal position and the constraint. The table below gives a summary of the run-time from 10 runs with randomized values.

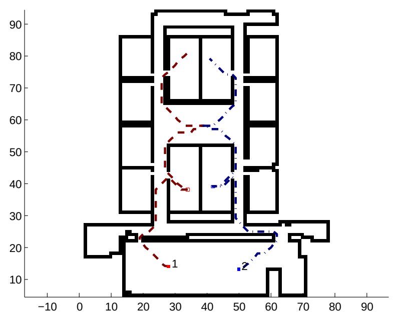


Fig. 7. Two robots with one task each. This is the solution from one of the randomized runs.

Min	**Max**	**Average**
11 s	66 s	33.9 s

V. Conclusion

In this paper we have extended the DPC algorithm to support an unordered set of tasks that the robots need to execute, while obeying the time-parametrized distance constraints. We integrated the task order into the search graph of each robot and as a result obtained the order of execution of the tasks as part of the natural solution to the problem. The simulations demonstrate the efficiency of the algorithm in solving very large problems. The algorithm has therefore the potential of solving extremely complex planning problems with complicated constraints in a relatively short period of time. In addition the algorithm provides strong theoretical guarantees on solution quality. Further optimization of the algorithm and its hardware implementation for true distributed execution is in progress.

VI. Acknowledgement

We gratefully acknowledge support from 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] N. Atay and B. Bayazit. Emergent task allocation for mobile robots. In *Proceedings of Robotics: Science and Systems*, Atlanta, GA, USA, June 2007.

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

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

[6] Bibhuti Bhusan Choudhury and Bibhuti Bhusan Biswal. An optimized multirobot task allocation. In *ICETET '08: Proceedings of the 2008 First International Conference on Emerging Trends in Engineering and Technology*, pages 320–325, Washington, DC, USA, 2008. IEEE Computer Society.

[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] Michael Erdmann and Tomas Lozano-Perez. On multiple moving objects. pages 1419–1424, 1986.

[10] Richard Goodwin and Reid Simmons. Rational handling of multiple goals for mobile robots. In *Artificial Intelligence Planning Systems: Proceedings of the First International Conference (AIPS92*, pages 70–77. Morgan Kaufmann, 1992.

[11] Yi Guo and Lynne E. Parker. A distributed and optimal motion planning approach for multiple mobile robots. In *Proceedings of the 2002 IEEE International Conference on Robotics and Automation*, Washington, DC, May 2002.

[12] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems, Science, and Cybernetics*, SSC-4(2):100–107, 1968.

[13] Matthew Hoeing, Prithviraj Dasgupta, Plamen Petrov, and Stephen O'Hara. Auction-based multi-robot task allocation in comstar. In *AAMAS '07: Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, pages 1–8, New York, NY, USA, 2007. ACM.

[14] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006. Available at http://planning.cs.uiuc.edu/.

[15] Binns Lewis, Valachis Dimitris, Anderson Sean, Gough Dave, Nicholson Dave, and Greenway Phil. Distributed slam. In *Proceedings of Signal processing, sensor fusion, and target recognition Conference XI*, volume 4729, pages 62–68, Orlando, FL, Jan 2002.

[16] 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.

[17] Gao Ping-an and Cai Zi-xing. Multi-robot task allocation for exploration. *Journal of Central South University of Technology*, 13(5):548–551, Oct 2006.

[18] Pedro V. Sander, Denis Peleshchuk, and Barbara J. Grosz. A scalable, distributed algorithm for efficient task allocation. In *AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 1191–1198, New York, NY, USA, 2002. ACM.

[19] A. Stentz, M. Dias, R. Zlot, and N. Kalra. Market-based approaches for coordination of multi-robot teams at different granularities of interaction. *Proceedings of the ANS 10th International Conference on Robotics and Remote Systems for Hazardous Environments.*, 2004.

[20] A. Stentz and M. Hebert. A complete navigation system for goal acquisition in unknown environments. *Autonomous Robots*, 2(2):127–145, 1995.

[21] Milos Zefran. *Continuous methods for motion planning*. PhD thesis, University of Pennsylvania, Philadelphia, PA, USA, 1996. Supervisor-Vijay Kumar.

[22] 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.

[23] Robert Michael Zlot. *An Auction-Based Approach to Complex Task Allocation for Multirobot Teams*. PhD thesis, Robotics Institute, Carnegie Mellon University, 5000 Forbes Ave, December 2006.