2007 IEEE International Conference on
Robotics and Automation
Roma, Italy, 10-14 April 2007

Pre-positioning Assets to Increase Execution Efficiency

Laura M. Hiatt
Computer Science Department
Carnegie Mellon University
5000 Forbes Ave.
Pittsburgh, PA 15213
lahiatt@cs.cmu.edu

Reid Simmons
Robotics Institute
Carnegie Mellon University
5000 Forbes Ave.
Pittsburgh, PA 15213
reids@cs.cmu.edu

***Abstract*— In many robotic domains, efficiency is an important component of task execution. One way to improve task efficiency is to lessen the overhead of beginning a task by making sure the necessary agents are near the task site when execution begins, minimizing travel time delays – in other words, pre-positioning agents for their future tasks. In static, certain domains, this can easily be done in advance and incorporated into the initial plan. In dynamic domains such as search and rescue, however, there is not enough certainty about task execution to plan for this ahead of time. To address this, we present here a planner that adds pre-positioning to a plan during execution. The planner strategically positions groups of idle robots whose future task assignments are uncertain in order to minimize travel time by the group as a whole once its members are allocated tasks. Because this planner must run in real time, we present five versions of the planning algorithm, addressing the trade-off of computation time and solution quality that results. We then show that by adding in this type of planning, the overhead of beginning a task can be reduced by up to 90%.**

I. Introduction

In robotic domains such as search and rescue, efficiency is a critical component of task execution. Because of the nature of the situation, every second counts, and any team, whether robot, human or both, that works in search and rescue has to meet this demand.

Imagine that there are two groups of robots responsible for survivor rescue. One is made of search robots, which perform initial sweeps for survivors. The second is of medic robots, which trail behind and, when a survivor is found, treat him.

In static, certain domains, in which all task assignments and start times are known at plan time, an initial plan would include pre-positioning agents for their future tasks by moving them as close as possible to the task site before execution begins. Due to the uncertainty of the above domain, however, such advance knowledge is not possible. It is not known which medic robots will be idle, as that depends on the earliness, lateness and assignment of previous tasks, which building will need a medic robot, since it is hard to predict where a survivor will be found, nor when a medic robot will be needed, due to the difficulty of predicting how long a search will take.

Planning pre-positioning for idle robots at run time, however, when task execution and start times are known with higher probability, can be done and is an effective way of increasing task execution efficiency. To do this we introduce a second, run time planner into the architecture. This planner takes in the current, uncertain plan and pre-positions groups of idle robots with uncertain future tasks without affecting with the overall plan structure. It intelligently chooses pre-positioning locations, optimizing placement of the group as a whole, by minimizing the total expected travel time overhead of beginning the robots' next tasks. Because solving this problem exactly requires exponential search, we have also developed four approximations of the exact result, each with their own balance of the trade-off of computation time versus solution quality.

In the following section, we discuss work related to ours. We next give an overview of each of the algorithms. Then, in Sections IV and V, we present experiments ran in simulation and discuss the results. Finally, we conclude and discuss future work in Section VI.

II. Related Work

A. Run Time Task Allocation and Plan Optimization

In a highly dynamic domain such as search and rescue, it can be useful to integrate planning and execution so that the plan is developed as execution progresses [1]. Jin, et al use a market-like approach, where UAVs use a cost value to 'bid' to a central controller for new tasks, to solve a stochastic version of the dynamic vehicle routing problem [2]. Other work uses a distributed algorithm to dynamically assign UAVs tasks in their respective Voronoi regions [3]. Our pre-positioning is similar in spirit to these approaches, but focuses on taking advantage of known possible future, not current, task assignments.

There is also an increasing body of work being done on optimizing plan efficiency during run time. Distributed approaches such as the market-based approach achieve this by constantly tasking and retasking robots as execution progresses so the plan ideally consists of the current most efficient task allocation [4]. Centralized approaches also address this problem. Estlin, et al optimize during run time by initially construct a full plan and then replanning during execution when necessary [5]. Other work optimizes during task execution by updating parameters of task allocation, such as robot suitability, to improve the quality of allocation as execution progresses [6].

B. Path Planning Approximations

A common downside to run time planning is that the time it takes to plan can often grow to outweigh the benefits provided by the extra planning. Recent work has addressed this by exploring the use of Generalized Voronoi Diagrams (GVDs) on discretized grids of an environment to reduce the complexity of path planning and other related search algorithms [7], [8]. A GVD of an environment represents the set of points that are equidistant from the two (or more) nearest obstacles, essentially reducing the search space to well-placed paths through the environment [9]. By constraining the search space of an environment to its GVD, planning for a 2-dimensional grid can be reduced to a 1-dimensional, linear problem. Further, the GVD can be generated in time linear to the size of the grid. This is useful for a run time planner such as ours.

III. Problem Description

Our run time pre-positioner is useful in conjunction with plans that have too much uncertainty to allow planning pre-positioning at plan time. For our purposes, plans have two types of uncertainty: when a task will begin, and whether or not a task will require a robot for execution. In this paper, we separate these two types of uncertainty into two different problems, called Task Start Uncertainty (TSU) and Task Allocation Uncertainty (TAU). We do not yet consider the case where there is uncertainty of both types.

We present below different approximations of two algorithms to solve these problems and minimize the expected distance to travel after task assignment. We assume that all tasks are single-agent and that when a task begins and needs allocation the closest idle robot is assigned to it. We also assume that the robots' workspace is discretized into a grid; thus, all statements of optimality with respect to task travel time are optimal modulo this approximation.

A. Task Start Uncertainty Problem

For the Task Start Uncertainty problem, there is uncertainty about which task will start next. A demonstrating scenario is several search robots waiting for fire-fighters to put out fires in many adjacent buildings before being able to enter and look for survivors. We want to minimize the time that passes between when a task is allocated (a fire is put out) and the assigned robot arrives at the task site (the search robot begins its search). This is a hard problem because we are optimizing placement for a group of idle robots as a whole, and so the robots' positions are dependent on one another. An example is shown in Figure 1. The solution is a single position for each robot to wait for a task to be allocated.

At a high level, given an occupancy grid and task information, the algorithm considers each grid cell for each robot, and returns the configuration that minimizes expected travel. The expected distance to travel that we are minimizing, shown in (1), is the sum of the Euclidean distances of the shortest path between each task and the nearest robot, where each distance is weighted by the probability that task will start first. The probability a task will be the first to start can be calculated from the probability distributions of each task's start times. The expected distance can be solved in $O(rt)$ time, where r is the number of robots and t is the number of tasks. The psuedocode for this straight-forward algorithm is shown in Algorithm 1. Since it considers all valid plans, it is guaranteed to find the optimal solution. The algorithm runs in $O(rt(mn)^r)$ time, where mn is the size of the grid, r is the number of robots to consider and t is the number of tasks.

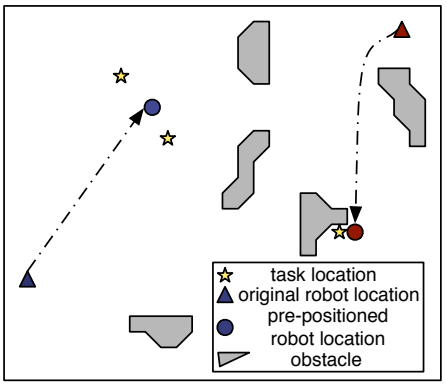

Fig. 1. A sample problem and solution to the TSU problem, with 2 robots, and 3 tasks with equal probability of starting first.

$$\text{ExpDist}(\text{robots\_pos}, \text{tasks}) = \sum_{i}^{\text{numTasks}} p(\text{tasks}[\text{i}]) * \text{distance}(\text{closest}(\text{robots\_pos}, \text{tasks}[\text{i}]), \text{tasks}[\text{i}]) \quad (1)$$

Algorithm 1: The TSU Algorithm

```
1  CalcExactTSU(tasks)
2      cell[] solution = new cell[]
3      bestDist ← TSUHelper(1, tasks, solution)
4  return solution
5  TSUHelper(curRobot, tasks, cells[])
6      bestDist ← INT_MAX
7      curCells[1...curRobot − 1] = cells[1...curRobot − 1]
8      foreach free cell c in grid do
9          curCells[curRobot] ← c
10         if curRobot == numRobots then
11             curBestDist ← ExpDist(curCells, tasks)
12         else
13             curBestDist = TSUHelper(curRobot + 1,
               tasks, curCells)
14         if curBestDist < bestDist then
15             cells[curRobot...numRobots] ←
               curCells[curRobot...numRobots]
16             bestDist ← curBestDist
17 return bestDist
```

1) Generalized Voronoi Diagram Approach: As the grid size or the number of robots increases, the TSU algorithm can take a prohibitively long time to run. One way to reduce

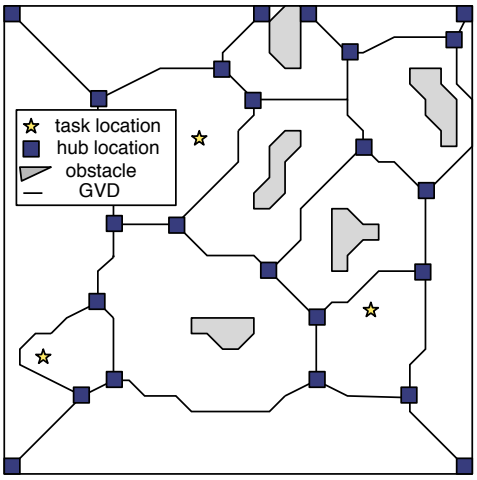

Fig. 2. A sample workspace, with the GVD and hub points shown.

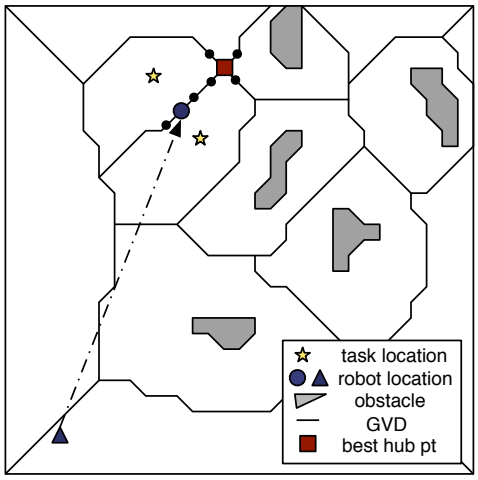

Fig. 3. A sample workspace, with the GVDGA solution and gradient search space shown.

the run time of the algorithm is to reduce the number of points it considers while searching for the best set of locations for the robots. The Generalized Voronoi Diagram Approach (GVDA) does this by considering only cells that are on the Generalized Voronoi Diagram of the grid. This reduces the search space to well-placed paths through the grid which intersect at 'hub points'. Calculating the GVD can be done in a trivial $O(mn)$ time by using a Euclidean Distance Transform algorithm [10]. A workspace with the corresponding GVD is shown in Figure 2. By using the GVD, which has $O(\sqrt{mn})$ cells, we reduce the asymptotic run time to $O\left(rt\left(mn\right)^{r/2}\right)$.

2) Generalized Voronoi Diagram Gradient Approach: Even with the reduction of run time that the above approach gives us, if the grid size or number of robots increases further a faster algorithm is desired. Thus, we developed a third approximation, the Generalized Voronoi Diagram Gradient Approach (GVDGA).

In this approach, we begin by solving for the GVD as above. We then turn our attention to the hubs of the Voronoi diagram, which correspond to the set of points that are on the border of 3 or more Voronoi regions. Because the GVD in general is well-placed throughout the grid, the hub points are, as well. Figure 2 shows an example workspace displaying the hubs of its GVD. Identifying the hubs can be done in time linear to the size of the GVD, or in $O(\sqrt{mn})$ time.

The algorithm begins by finding the best hub point for each robot. Starting from there, it searches along the Voronoi to find a local minimum approximation of the exact GVD solution. It considers the neighbors on the GVD of the current cell, moves in the direction of the cell that decreases the expected distance the most, and repeats until it reaches a local minimum. This is similar to a gradient descent algorithm. An example is shown in Figure 3. In this example, the algorithm first returned the best hub after considering all hubs, and then searched along the GVD as marked by the small black dots until it found a local minimum, shown as the marked robot pre-position. It considered far fewer points than the GVDA would have, despite having the same asymptotic complexity.

3) Randomized Gradient Approach: We also complemented the GVDGA with a Randomized Gradient Approach (RGA). This algorithm, instead of using the hubs of the GVD, chooses a comparable number of random cells of the grid. Like the above approach, it follows the gradient starting from the best hub until it reaches a local minimum; however, it considers all 8 neighboring grid cells of the current point instead of just the neighbors along the GVD. The asymptotic run time of this algorithm is the same as that of the GVDGA.

B. Task Allocation Uncertainty Problem

For the Task Allocation Uncertainty problem, although the task start times are known, there is uncertainty about whether or not tasks will need a robot for execution. An example search and rescue scenario is one in which search robots may, or may not, find survivors at any given search location, and so the closest medic robot to that location may, or may not, be assigned the task of treating a survivor. Here we want to minimize the total time that passes between when each task is allocated (a survivor is found) and when the assigned robot arrives (a medic robot arrives to treat the survivor). To do this, each robot begins with an initial pre-position. Then, when a task's start time passes, the remaining idle robots reposition in order to be closer to the next tasks to start. An example is shown in Figure 4. The solution space for this problem is a series of points for each robot to follow, one per task.

Because the start times of each task are known, we constrain the search space to account for travel time and consider only points that are reachable by the robot before the next possible task begins, given some a priori speed for the robot. This means that the optimal position for a robot to go to at any time is not necessarily directly towards the location of the next task to start: this could lead to the robot

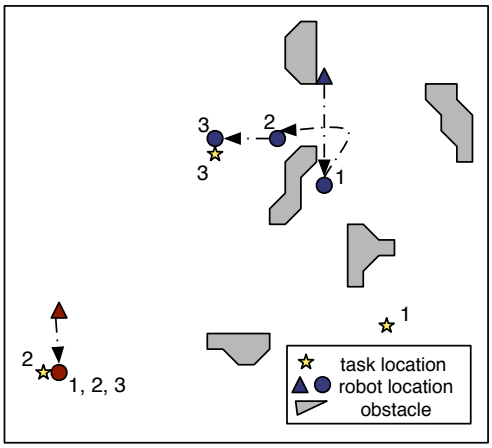

Fig. 4. A sample problem and solution to the TAU problem, with 2 robots, and 3 tasks.

being much farther away from a second, later task and a higher overall expected distance. This problem is thus even harder than the TSU problem because the robots' positions are coupled in two ways - between positions of robots for the same task, and between sequential positions of the same robot. It is also more complicated because, although for the first task we know which robots are idle and therefore which robot will be closest to it when it begins, for later tasks this is only probabilistic since the set of robots that are still idle is uncertain.

To solve this problem, given an occupancy grid and task information, the algorithm considers each reachable grid cell for each robot each time a task could possibly start, calculates the expected total distance to be traveled during the task assignment process, and returns the solution with the lowest expected distance. The expected distance equation that we are minimizing is shown in (2), where p is the probability that the task will need a robot for execution. The base case of this equation is when either $\mathsf{i} > \mathsf{size(tasks)}$ or $\mathsf{robots\_pos} = \{\ \}$, making $\mathsf{ExpDist\,(robots\_pos, i)} = 0$. It is solved in $O\left(r2^t\right)$ time, where r is the number of robots,and t is the number of tasks. The psuedocode of the algorithm is shown in Algorithm 2. It is guaranteed to find the optimal solution since it considers all possible valid plans.

$$\begin{aligned}\mathsf{ExpDist\,(robots\_pos, i, tasks)} &= p\,(\mathsf{tasks\,[i]}) \\ &\quad * (\mathsf{distance\,(closest\,(robots\_pos, tasks\,[i])\,, tasks\,[i])} \\ &+ \mathsf{ExpDist(robots\_pos} - \{\mathsf{closest\,(robots\_pos, tasks\,[i])}\}, \\ &\qquad \mathsf{i} + 1, \mathsf{tasks})) + (1 - p\,(\mathsf{tasks\,[i]})) \\ &\qquad * \mathsf{ExpDist\,(robots\_pos, i} + 1, \mathsf{tasks}) \quad (2)\end{aligned}$$

Solving this algorithm exactly runs in $O\left(r2^t\left(mn\right)^{rt}\right)$, where mn is the size of the grid, r is the number of robots, and t is the number of tasks. For a small occupancy grid and only a few robots, using this approach is feasible; beyond that, however, its use becomes impractical as its additional exponential complexity makes the run time prohibitively large.

Algorithm 2: The TAU Algorithm

```
1  CalcExactTAU(tasks)
2      cell[][] solution = new cell[][]
3      bestDist ← TAUHelper(1, 1, solution)
4  return bestDist
5  TAUHelper(curRobot, curTask, cells[][])
6      bestDist = INT_MAX
7      curCells ← cells
8      foreach reachable, free cell c do
9          curCells[curTask][curRobot] = c
10         if curRobot == numRobots&&curTask ==
           numTasks then
11             curDist ← ExpDist(curCells, 1, tasks)
12         else if curRobot == numRobots then
13             curDist ←
               TAUHelper(1, curTask + 1, curCells)
14         else
15             curDist ←
               TAUHelper(curRobot + 1, curTask, curCells)
16         if curDist < bestDist then
17             bestDist ← curDist
18             cells[curTask][curRobot] = c
19 return bestDist
```

1) Generalized Voronoi Diagram Approach: To make the problem more tractable, we first reduce the search space to be restricted to the cells of the Generalized Voronoi Diagram (GVD), as we did with the above TSU problem. This makes the asymptotic run time $O\left(r2^t\left(mn\right)^{rt/2}\right)$.

2) Generalized Voronoi Diagram Gradient Approach and Randomized Gradient Approach: We adjusted the GVDA algorithm in the same manner as before to create the Generalized Voronoi Diagram Gradient Approach and Randomized Gradient Approach. Their complexity is again the same as that of the GVDA.

3) Greedy Approach: Because of the higher complexity of this problem, we developed a greedy approach. This algorithm solves each task location independently of future locations, ignoring the coupling between sequential positions of a robot. In other words, it does not consider future implications of moving to a point; instead, it solves for the best points it can get to at each task start time, given previous positions and how far it can travel.

This algorithm thus uses the same basic structure as the exact solution, but does not recurse for each task. The asymptotic run time of this algorithm is $O\left(rt2^t\left(mn\right)^r\right)$.

IV. Experiments and Results

In order to compare these algorithms, we ran them in simulation using scenarios with different sizes of randomly generated occupancy grids and different numbers of robots and tasks.

A. Method

In our simulated scenarios, grids had a random number of obstacles, each in a random location and of a random size.

TABLE I
TASK START UNCERTAINTY RESULTS

algorithm	average expected distance	average difference	average run time (s)
1 robot, 4 tasks, 100X100 grid, 100 trials			
Exact	57.9	–	0.024
GVDA	59.3	1.35	0.022
GVDGA	59.9	1.91	0.021
RGA	58.1	0.13	0.022
2 robots, 4 tasks, 100X100 grid, 100 trials			
Exact	20.3	–	19.9
GVDA	24.1	3.83	0.506
GVDGA	25.2	4.92	0.037
RGA	21.0	0.68	0.040
3 robots, 4 tasks, 30X30 grid, 100 trials			
Exact	1.95	–	153.6
GVDA	4.06	2.12	10.0
GVDGA	4.34	2.40	0.165
RGA	3.54	1.60	0.100

The robots' initial locations, the tasks' locations and the task probabilities were also randomly generated.

For each algorithm and each problem, we recorded the expected distance to travel without pre-positioning, the run time, and the expected distance after pre-positioning.

B. Results

The results for the TSU problem are shown in Table I[1]. For a simple scenario with a single robot, all of the algorithms run equally fast. For scenarios with two robots, the exact algorithm slows down significantly, but the others remain relatively fast. Once three robots are considered, the exact algorithm takes prohibitively long and an approximation is needed. For all scenarios, the random algorithm performs the best of all the approximation algorithms.

Table II shows the results for the TAU problem. This problem introduced the notion of 'failures'. A failure occurs when no solution can be found that a robot can reach before the next task starts. Thus, failures are possible in the GVDA, GVDGA and RGA since they have constrained search spaces.

When there is only one robot, the exact algorithm finds the optimal solution in a reasonable amount of time. The greedy algorithm finds a reasonable solution in very fast time. For situations with more than one robot, however, the exact algorithm is not a viable option for run time use, and the greedy algorithm's solution deteriorates. The RGA also does not scale here as well as it does for the TSU problem. The GVDA, on the other hand, has a reasonable run time, fewer failures than the GVDGA for 1 robot, and fewer failures than the GVDGA and RGA for 2 robot scenarios.

[1]The distances reported in all tables are in terms of the number of grid cells; i.e., we treat grid cells as having a dimension of 1x1. Also, the average expected distance here denotes the average of the differences of each approximation with the exact algorithm, not the difference of the averages.

TABLE II
TASK ALLOCATION UNCERTAINTY RESULTS

algorithm	average expected distance	average difference	average run time (s)	success rate (%)
1 robot, 3 tasks, 100X100 grid, 100 trials				
Exact	27.2	–	58.6	100
GVDA	30.0	2.87	4.6	92
GVDGA	31.4	4.26	1.32	85
RGA	27.6	0.469	4.96	92
Greedy	29.1	1.94	0.018	100
2 robots, 3 tasks, 30X30 grid, 30 trials				
Exact	8.08	–	24718	100
GVDA	9.57	1.48	2319.1	93.3
GVDGA	10.41	2.33	70.4	83.3
RGA	8.72	0.64	2355.6	76.7
Greedy	11.23	3.15	0.013	100

TABLE III
OBSTACLE RESULTS FOR TSU

algorithm	normal obstacles	fewer obstacles	more obstacles
2 robots, 4 tasks, 100X100 grid, 100 trials			
	average expected distance		
Exact	20.3	21.9	21.1
GVDA	24.1	26.8	24.1
GVDGA	25.2	27.9	24.7
RGA	21.0	22.5	21.8
	average run time		
Exact	19.9	20.1	19.4
GVDA	0.506	0.318	0.903
GVDGA	0.037	0.030	0.042
RGA	0.040	0.036	0.034

To ensure that the mean of the randomly generated number of obstacles did not bias the results, we ran further tests varying the number of obstacles. The results are shown in Table III and compare the expected distance and run time of grids with the mean number of obstacles used in previous experiments with those of grids with both fewer and more obstacles. No algorithm is much affected by these variations.

Table IV compares the expected distance with no pre-positioning with the expected distance when pre-positioning is used. The 'average % decrease' column indicates the average of what percentage of the travel time without pre-positioning can be avoided by using pre-positioning. For each of the algorithms, a large savings in distance traveled is achieved.

V. DISCUSSION

For both problems, the exact algorithm, GVDA and GVDGA provide a nice demonstration of the trade-off of run time and optimality, with expected distance increasing and run time decreasing as we consider them in order. Thus although the GVDA and GVDGA are not guaranteed to find

TABLE IV
TSU AND TAU IMPROVEMENTS

algorithm	average expected distance without pre-positioning	average expected distance with pre-positioning	average % decrease
TSU, 3 robots, 4 tasks, 30X30 grid, 100 trials			
Exact	19.4	1.95	89.9
GVDA		4.06	77.0
GVDGA		4.34	75.6
RGA		3.54	80.5
TAU, 2 robots, 3 tasks, 30X30 grid, 45 trials			
GVDA	18.0	8.11	55.1
GVDGA		9.19	50.0
RGA		6.86	62.1
Greedy		9.94	46.0

a solution in the TAU case, they compensate for their less optimal solutions by running in less time and being tractable in larger spaces than the more optimal algorithms.

Although the RGA's solutions are generally better than the other approximations', for the TAU problem its reliability decreases as the number of robots increases. This is because the RGA's chosen hubs are not guaranteed to be as well-spaced throughout the environment as the GVDGA's, making it less likely that a hub is within the necessary range of a robot and a solution is found.

Despite their similarity, its run time also increases relative to the GVDGA for the TAU problem. We explain this by remembering that for this problem the RGA has a high overhead because it checks all 8 neighboring points for reachability, instead of the two that the GVDGA does. This leads to an overall higher run time.

The greedy algorithm is the fastest of the algorithms, and does well in domains with only one robot. In scenarios with multiple robots, however, its results degrade as it fails to take into consideration interactions between different robots at different times. Still, its low run time warrants its use in problems which are too large for the other algorithms to realistically solve during runtime.

Based on these results, we can make recommendations about approximation algorithms in this domain. For the TSU problem, the RGA outperforms both the GVDA and GVDGA algorithms and is a good alternative to the exact algorithm. For the TAU problem, the GVDGA has the best overall balance of run time, solution quality and success rate. If the grid, number of robots and number of tasks is too high even for the RGA or GVDGA, however, one can either use the greedy approach to ensure a solution is found in a short amount of time, or use a grosser discretization of the environment to lessen the number of grid cells the algorithms must consider.

The results shown in Table IV make a compelling argument for using run time pre-positioning in a robotic system. The use of pre-positioning in each of these scenarios is clearly warranted by comparing the planner's run time with the expected savings in travel time once a task is started.

VI. CONCLUSIONS AND FUTURE WORK

In conclusion, we have shown that run time pre-positioning can be a useful tool to increase task execution efficiency in uncertain plans. In particular, we have developed a planner that takes advantage of run time task execution probabilities to determine a strategic location for idle robots to wait for their next task assignment. By doing so, the overhead of beginning a task can be reduced by up to 90%.

In the future, we hope to extend this work to include more complicated scenarios. One example is situations in which there are both task start time and task allocation uncertainty. This would further increase the complexity of the TAU case because at any given future time, in addition to uncertainty about what agents are available, there is also uncertainty about which tasks have already started. Another extension is looking at how this would work with multi-agent tasks.

VII. ACKNOWLEDGEMENTS

This research was sponsored in part by NASA grant NNA04CK90A and by National Science Foundation Fellowship No. DGE-0234630. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

REFERENCES

[1] K. Golden, O. Etzioni, and D. Weld, "Planning with execution and incomplete information," Department of Computer Science and Engineering, University of Washington, Tech. Rep. UW-CSE-96-01-09, 1996.

[2] Y. Jin, A. A. Minai, and M. M. Polycarpou, "Cooperative real-time search and task allocation in uav teams," in *Proc. IEEE International Conference on Decision and Control*, 2003.

[3] E. Frazzoli and F. Bullo, "Decentralized algorithms for vehicle routing in a stochastic time-varying environment," in *Proc. IEEE International Conference on Decision and Control*, Paradise Island, Bahamas, December 2004.

[4] B. P. Gerkey and M. J. Matarić, "Sold!: Auction methods for multi-robot coordination," *IEEE Transactions on Robotics and Automation*, vol. 18, no. 5, October 2002.

[5] T. Estlin, F. Fisher, D. Gaines, C. Chouinard, S. Schaffer, and I. Nesnas, "Continuous planning and execution for an autonomous rover," in *Proceedings of the Third International NASA Workshop on Planning and Scheduling for Space*, Houston, TX, October 2002.

[6] C.-H. Fua and S. S. Ge, "COBOS: Cooperative backoff adaptive scheme for multirobot task allocation," *IEEE Transactions on Robotics*, vol. 21, no. 6, pp. 1168–1178, 2005.

[7] M. Foskey, M. Garger, M. Lin, and D. Manocha, "A voronoi-based hybrid motion planner," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2001.

[8] N. Kalra, D. Ferguson, and A. Stentz, "Constrained exploration for studies in multirobot coordination," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, May 2006.

[9] H. Choset, "Incremental construction of the generalized voronoi diagram, the generalized voronoi graph, and the hierarchical generalized voronoi graph," in *Proceedings of the First CGC Workshop on Computational Geometry*, Baltimore, MD, October 1997.

[10] H. Breu, J. Fil, D. Kirkpatrick, and M. Werman, "Linear time euclidean distance transform algorithms," *IEEE Transactions on Pattern Analysis and Machine Inteliigence*, vol. 17, pp. 529–533, May 1995.