Proceedings of the 2016 IEEE International Conference on
Simulation, Modeling, and Programming for Autonomous Robots
San Francisco, USA, Dec 13-16, 2016

Decentralized Allocation of Tasks with Temporal and Precedence Constraints to a Team of Robots

Ernesto Nunes[1], Mitchell McIntire[2], Maria Gini[1]

***Abstract*— We propose an auction-based method for a team of robots to allocate and execute tasks that have temporal and precedence constraints. The robots use our priority-based iterated sequential single-item auction algorithm to allocate tasks among themselves and keep track of their individual schedules. The key idea is to decouple precedence constraints from temporal constraints and deal with them separately. In this paper we demonstrate how the allocation scheme can be extended to handle failures and delays during task execution. We demonstrate the effectiveness of our method in simulation and with real robot experiments.**

I. Introduction

Service robots operating in large areas may have tasks that are distributed in space and need to be executed within specific time intervals while satisfying precedence constraints. Examples include robots in warehouses, hospitals, and offices. Our framework enables autonomous mobile robots to divide work among themselves, so that the *makespan* – the difference between the overall latest finish and the overall earliest start time – is minimized without violating any constraint. In this paper we show how robots can handle delays or failures during the execution of the tasks.

This task allocation problem is NP-hard even in its most basic form, a single-robot without precedence constraints, which is a special case of the elementary shortest path problem with resource constraints [1]. Our problem falls under the XD [ST-SR-TA = Single-Task robot, Single-Robot task, Time-extended Assignment] category in the iTax taxonomy [2]. The cross-scheduling constraints [XD] among robots induced by precedence constraints make the problem harder: tasks assigned to a robot might depend on tasks assigned to other robots. Any delay or execution failure in one robot's schedule will affect other robots' schedules.

For scalability and robustness we use a decentralized approach, where each robot owns its schedule for the subset of tasks assigned to it. A schedule is represented as a simple temporal network (STN) [3], which stores the execution times. To handle the precedence constraints, we use our prioritized iterated auction (pIA) [4]. pIA uses a hierarchical approach to "peel" off layers of the precedence graph. The tasks in each precedence layer are incrementally allocated and moved from more to less constrained. Once they become unconstrained they are auctioned off using a modified temporal sequential single-item auction [5].

[1] Ernesto Nunes and Maria Gini are with the Department of Computer Science and Engineering, University of Minnesota. enunes@cs.umn.edu, gini@cs.umn.edu. [2] Mitchell McIntire is with the Department of Computer Science at Stanford University. mcint286@stanford.edu.

Our main contribution is a framework that uses a variant of pIA for task allocation at planning time and combines it with an executive that monitors the execution of tasks, reallocating tasks via a one-shot greedy auction when needed because of execution delays or failures. In addition to the precedence constraints already handled in pIA [4], this framework extends pIA to handle general time window constraints. Unlike [4], which is solely a planning algorithm, the framework herein proposed supports task execution and recovery via a planning-execution-replanning cycle. We present a thorough evaluation of the framework in simulation and through experiments with real robots. The robot experiments are done using the Robot Operating System (ROS) and three Turtebot2 robots. Our algorithm is compared to a simple but effective greedy auction, and to a centralized implementation of a mixed integer linear program (MILP) with two different optimization objectives.

II. Related Work

Methods for task allocation with precedence constraints have been proposed in several areas, such as multiprocessor scheduling, and vehicle routing problems (VRP). However, most problems studied in the parallel processing literature are simpler than ours because they do not account for robot travel, and standard VRP methods are unsuitable for our problem because they are centralized.

Distributed Constraint Optimization Problem (DCOP) [6] algorithms provide a viable option for modeling constraint problems in a distributed way. However, solving DCOP exactly is NP-hard and impractical for real problems [7]. To overcome this shortcoming approximate methods such as Max-Sum have been proposed [8], yet we are not aware of any DCOP algorithm that handles task allocation with precedence and time window constraints.

Auction-based approaches have become popular for their flexibility, decentralized nature, and robustness to failure. The sequential single-item (SSI) auction [9] is effective and computationally inexpensive compared to other auction schemes, but it does not easily extend to include precedence and temporal constraints (see [5] for an extension).

Decentralized solutions have been proposed in [10], where the coupling introduced by precedence constraints is maintained by adding to each robot schedule a set of "remote" nodes, which have inter-dependency with the local tasks. Our approach avoids the very large temporal representation they need for dense precedence graphs when predecessors and successors of tasks are assigned to other robots, and the high

978-1-5090-4616-4/16/$31.00 ©2016 IEEE

computation and communication costs needed to update the temporal model when the environment changes rapidly.

In [11], a distributed solution is presented for tasks that have precedence and resource constraints, but no temporal constraints. In [12] a multi-tier auction and a genetic algorithm are proposed for tasks that have intra-path and precedence constraints, but again no temporal constraints. Luo et al. [13] introduced an auction-based algorithm for multi-robot task assignment for a special case when precedence constraints are in a form called *set* precedence constraints. The tasks are partitioned into disjoint sets of size at most equal to the number of robots. Robots can do at most one task per set. Instead, we allow general precedence constraints in addition to the temporal constraints.

III. Problem Definition and Model

We assume a set $\mathcal{R}$ of m robots. Each robot r_i has an initial pose, a maximum velocity, and a set of sensors. The maximum velocity is the same for all robots, but robots can travel at different speeds. Every robot is given a graph representation of the environment. The graph's vertices are waypoints and its edges connect pairs of vertices between which there are no obstacles.

Additionally, we have $\mathcal{T}$, a set of n tasks, each with a location, an earliest start time ES_{t_j}, a latest finish time LF_{t_j}, and a duration (DU_{t_j}). Together, (ES_{t_j}, LF_{t_j}) define the bounds for the task's time window. Robots need to arrive to a task before its latest start time LS_{t_j}, which, if not specified, can be computed as $LS_{t_j} = LF_{t_j} - DU_{t_j}$.

While time windows impose in-schedule constraints for individual robots, precedence constraints can create cross-scheduling constraints since those tasks can be allocated to different robots. We use a directed acyclic graph (DAG) to model precedence constraints. Nodes in the graph represent tasks, and edges represent precedence relations. For example, $t_1 \prec t_2$ means that t_1 precedes t_2, or equivalently, $(t_1, t_2) \in E$, where E is the set of directed edges in the DAG. In our problem, a valid schedule consists of a partition of $\mathcal{T}$ across $\mathcal{R}$ in which a task is assigned to a single robot, and the execution times assigned to tasks respect their time windows and precedence constraints.

IV. Background on Prioritized Iterated Auction

pIA [4] is an auction-based algorithm in which precedence-free tasks are auctioned in each iteration. The auction alternates between precedence graph layering and rounds of a re-purposed temporal sequential single-item auction [5] until all feasible tasks are allocated.

Hierarchical Decomposition of the Precedence Graph. In each iteration the auctioneer divides the precedence graph ($G_{\mathcal{P}}$) into three layers: the *free layer* (T_F), the *second layer* (T_L), and the *hidden layer* (T_H). T_F contains tasks without any predecessor, T_L contains tasks with parents in T_F, and all the remaining tasks not yet touched by the planning algorithm are in T_H.

Graph layering leads to a decomposition that allows individual robots to bid on tasks that are independent (precedence-wise) in each iteration of the auction.

Task Prioritization. Tasks are assigned a priority depending on their criticality. Not all tasks in the free layer affect equally the temporal problem. This means that more "critical" tasks, i.e., tasks that are precedence constraints for longer chains of tasks, should be auctioned off first. We use a simple priority assignment heuristic (also used in [4]) that is based on the shape of the precedence graph.

We define $U(t)$ and $L(t)$ for $t \in \mathcal{T}$ to be the length of the longest path in $G_{\mathcal{P}}$ rooted at t, respectively with and without travel time between tasks. More precisely, we let

$$L(t_k) = du_{t_k} + \max_{t_j \in \text{children}(t_k)} (L(t_j)) \ ,$$

$$U(t_k) = du_{t_k} + \max_{t_j \in \text{children}(t_k)} (tt(t_k, t_j) + U(t_j)) \ ,$$

where we use the convention that $\max_{\emptyset}(x) \equiv 0$, so that L and U for a task without children are equal to the task's duration. Then we let

$$\text{prio}_{\beta}(t_k) = (1-\beta)L(t_k) + \beta U(t_k) \ , \ 0 \leq \beta \leq 1 \qquad (1)$$

where $U(t_k)$ represents the total time it would take a single robot (in the absence of any constraint) to execute a task chain that starts with t_k. $L(t_k)$ is the least unconstrained time required to execute these tasks. The priority function in (1) ranks tasks depending on how much they add to the longest path in $G_{\mathcal{P}}$'s. The tasks in the set of free tasks that add more will be chosen, provided that their critical value is higher than that of the most critical second-layer task. This allows the auction to defer non critical tasks to the next auction iteration. The parameter β balances its two components. β can be thought of roughly as the proportion of travel time that is accounted for in task priorities. β can be adapted according to the tasks and precedence configurations. The values of $\text{prio}_{\beta}(t_k)$ are computed in linear time by using bottom-up dynamic programming starting from tasks that do not have successors.

Auction and Robot Bidding. In each iteration of the auction, a subset of high priority tasks in T_F are auctioned-off.

Upon receiving a list of tasks up for auction, each robot computes a bid for each task. To compute the bid the robot temporarily inserts one task at a time in its schedule. If the insertion is feasible, the bid value is computed as $\alpha \times m + (1-\alpha) \times \Delta tt$, where m is the makespan after inserting the task and Δtt is the additional travel time the robot incurs for the new task. α is a parameter used to specify the relative importance of makespan vs. travel time. Travel time is estimated using the distance and a constant fraction of the robot maximum speed.

Each robot sends to the auctioneer only one bid, its smallest one. The auctioneer keeps a priority list of the bids received and selects the overall minimum bid efficiently (in time logarithmic in the number of bids). The task is assigned to the robot that submitted that minimum bid. Only one task is allocated in each round of the auction. The auctioneer

marks that task as scheduled and removes it from T_F.

The auction terminates when all feasible tasks in T_F are allocated. Some tasks might end up not being allocated because there are not enough robots, or because choices made early in the auction are never revisited. When the auction terminates, the auctioneer updates the earliest start times for the tasks in T_L to account for the maximum finish times over all their predecessors in T_F (see Temporal Model and Validity Checking), and the process restarts with the next layer of the precedence graph.

To prevent the algorithm from looping forever trying to allocate tasks that cannot be allocated, pIA terminates after a maximum number of iterations, which in our implementation is set to the total number of tasks. Once the tasks in T_F are allocated, they are removed from $G_{\mathcal{P}}$ and tasks in T_L that follow the removed tasks are promoted to T_F.

Temporal Model and Validity Checking. Temporal constraints are modeled as a simple temporal network [3]. In our model, each robot keeps its own STN (see [5] for details), which grows as more tasks are allocated to the robot. The auctioneer keeps a DAG with all the tasks' start times that account for the precedence constraints.

To improve efficiency we employ a simple propagation and consistency checking scheme inspired by [14]. Our scheme takes advantage of the hierarchy imposed by the precedence constraints to compute the start and finish times of tasks in time linear in the number of tasks in the schedule. This checking is done during the bidding phase, to ensure that the insertion of a task in a robot's schedule does not lead to an infeasible schedule. The start time of task t_k, S_{t_k}, is set to zero if the task is the dummy task representing the robot's initial location. Otherwise, it is set to $\max(\tilde{S}_{t_k}, ES_{t_k}, F_{t_j} + tt_{t_j,t_k})$ where $\tilde{S}_{t_k}$ is the maximum finish time over all $t_j \prec t_k$. This only if the resulting value of $S_{t_k} \leq LS_{t_k}$, i.e. the start time is not greater than the latest start time. If not, the value of S_{t_k} is set to ∞. The task's finish time is then computed as $F_{t_k} = S_{t_k} + DU_{t_k}$.

After the tasks in T_F are assigned and before the tasks in T_L are promoted to T_F, the auctioneer computes the value $\tilde{S}_{t_k}$ for all the tasks $t_k \in T_L$ and sends them to all the robots. This ensures that the constraints created by the schedule of T_F tasks are accounted for when bidding for T_L tasks.

If no tasks in a robot's schedule have ∞ as start time, the robot inserts the task in its schedule and computes the makespan which is the finish time of its last task. If any of the start times is assigned the value ∞ it means that the schedule is inconsistent, in which case the makespan will also be ∞ and the task cannot be assigned to that robot.

V. Dispatching and Re-Auctioning Tasks

Robots communicate with each other and the auctioneer via ROS publishers and subscribers. We run an auction topic in which requests for bids and bids are sent back and forth. Bidding is done synchronously, the auctioneer waits until all robots send their bids prior to choosing the winner.

After the tasks are allocated, when the start time arrives robots execute the tasks in their schedules in real time in ROS/Gazebo. The robots we used are Turtlebot 2. When delays occur the auctioneer, which monitors execution across schedules, updates the robots' schedules with adjusted start times to ensure they still respect the precedence constraints.

Robot Schedule Dispatching and Task Execution. At any time during execution the robot is either traveling to a task, executing a task, waiting to perform a task, or aborting execution. These states are directly correlated to the following execution outcomes: *succeeded* – the task was successfully executed, *aborted* – the assigned robot estimates that it cannot arrive to the task on time but there is still time to perform the task, *failed* – due to temporal constraint violation no robot in the system can do the task.

When the execution starts, a robot retrieves the first task in its schedule and travels to that task. In each ROS cycle, the robot computes the estimated start time of its next task by adding the travel time from its current location to the task's location to the already elapsed time (since the start of the traveling action). If the estimated start time is smaller than the start time computed during planning, the robot continues execution. If not, there are different cases. If the estimated new start time is smaller than the task's latest start time, the task can still be executed, but the new start time could cause inconsistencies with other tasks for which this task is a precedence constraint. Since those other tasks could have been assigned to other robots, the delayed robot needs to check with the auctioneer for any potential violation. If there are no violations, the robot assigns the estimated new start time as the task's start time and continues executing. Otherwise, the robot notifies the auctioneer that it is unable to execute the task within its temporal constraints. The auctioneer runs an auction to try to reallocate that task.

When a robot has completed the execution of a task, it marks it as succeeded, notifies the auctioneer, and proceeds to its next task. This way the auctioneer can keep track of the overall progress.

Auctioneer Execution Updates. As execution unfolds the auctioneer updates information on the tasks that have been completed and the tasks whose start times need to be updated due to execution delays.

To stay updated, the auctioneer subscribes to ROS topics in which robots post their tasks' execution status. In each ROS cycle it checks if a task has been completed. If a task is marked as aborted, the auctioneer auctions that task to all the robots to see if any of them can add it to its schedule. Each robot computes its bid, as it did during planning, and sends the bid to the auctioneer, which allocates the task to the robot with the smallest bid, if any. If no robot can do the task (i.e., all robots submit ∞ as their bid value), the task and all the tasks in its induced subgraph are marked as *failed* and removed from the scheduled tasks. To keep precedence constraints consistent failed tasks are never executed.

When a robot experiences a delay it attempts to set a new value to its task start time and any task whose execution time depends on the delayed task. The auctioneer needs to check if the new start time causes inconsistencies for other

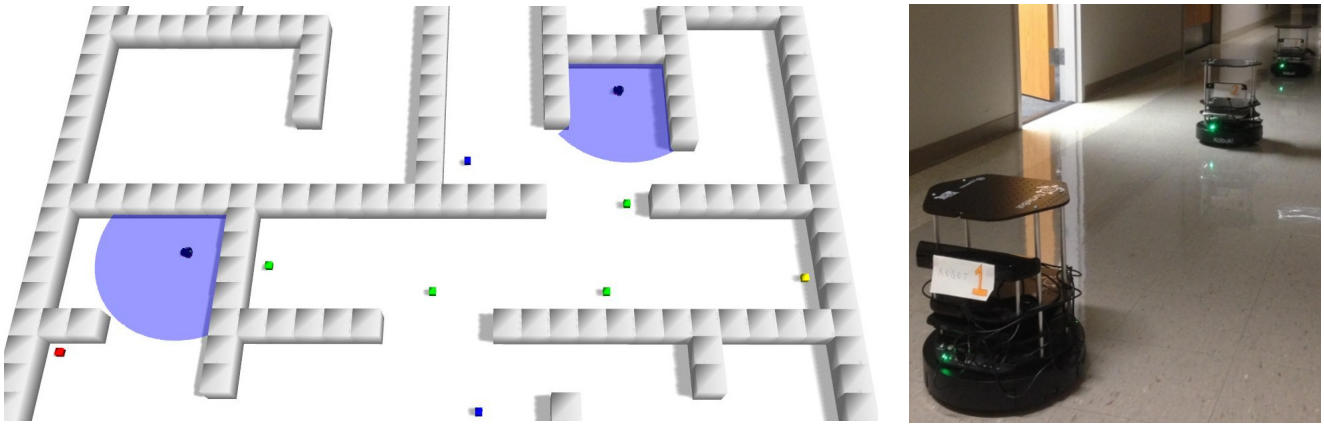

Fig. 1. (Left) Example indoor Gazebo simulation scenario with two robots and eight tasks (cubes). Tasks (colored cubes) can be thought as hazardous materials than can only be cleared at certain times of the day. The colors represent dependencies between tasks: the precedence order is red, blue, green, and yellow. The goal is to minimize the time to clear the last hazard or the distance covered. (Right) Three Turtlebot 2 robots used for our physical robot experiments, which operated in the room and corridor environment shown.

robots. The auctioneer keeps a DAG that encodes precedence constraints and start and finish times of tasks, which are updated during execution. When a robot asks the auctioneer to check for inconsistency of a potential time update, it sends the new start (and finish) times. The auctioneer proceeds by temporarily updating the finish times of all remaining tasks in the DAG. Next, it performs a topological sort on the DAG to compute a linear ordering according to the precedence constraints. The auctioneer uses the sorted graph to compute $\tilde{S}_{t_k}$ for all tasks. It then checks if $\tilde{S}_{t_k} \leq LS_{t_k}, \forall t_k$, if that is the case, the auctioneer accepts the temporal updates and the robot is sent an "OK" message. Otherwise, the auctioneer rejects the time updates, the robot aborts the task, and the auctioneer resets the task start time to its previous value.

VI. Experimental Setup

Simulation Experiments in 3D. Simulation experiments were conducted in ROS [15], using the Gazebo plugin. We built a model for virtual Pioneer robots, and a 3D world in which the robots operate.

To facilitate robot localization and motion planning, the simulator also keeps a 2D map of the world. The maps are discretized by overlaying a graph over them. A node in the graph represents a (x, y) location, the weighted edges represent Manhattan distances between pairs of nodes without obstacles between them. The graph is used for path planning, using Dijkstra's algorithm to compute the distance between graph nodes (or waypoints). The 100 $\times$ 100 meters map corresponding to the world in Fig. 1 contains a total of 40 points, from which we choose task locations.

Real Robot Experiments. We also validated our algorithm with three Turtlebot 2 robots and 12 tasks. Each robot has a Kinect sensor, which is used for obstacle avoidance. These experiments were run for each benchmark algorithm on a map (54 by 51 meters) of a room and corridors (see the right part of Fig. 1). The results for these experiments were averaged over five runs.

Data Generation. We generated a set of tasks, each located at a distinct waypoint. Each task's x-y location is randomly drawn within the map, and a nearest-neighbor search with Manhattan distances is used to assign the task to the nearest waypoint. In our data sets, each task is assigned an earliest start time that is randomly drawn from $\mathcal{U}(25, 400)$, which are the numbers of seconds from the beginning of the simulation. The length of the tasks time windows is uniformly drawn from $\mathcal{U}(100, 1200)$, hence the time window with the latest possible end point closes roughly 26 minutes after the simulation starts. Tasks' durations range from 20 to 40 seconds.

We also generated precedence graphs randomly. To prevent the generation of over-constrained problems, we placed restrictions on the number of edges in the graph. In our experiments, we created precedence graphs with random density. This randomization is important to test the algorithm sensitivity to graph shapes and sizes. Graphs can have at most $3n$ edges, where n is the number of nodes (or tasks) in the graph. The algorithm that generates the precedence graph is described in [4]. Each data set is created by keeping tasks' locations fixed, while the tasks' time windows are allowed to change. We generated 10 data sets for each map and number of tasks.

Benchmark Algorithms. In addition to pIA, tasks are allocated using a greedy auction and two implementations (OPT-M) and (OPT-Duo) of the MILP in [4]. In the greedy auction, the auctioneer allocates up to m tasks, one per robot per round, which is equivalent to each robot greedily choosing the task with the least cost in each round. OPT-M minimizes the makespan, $z(\mathcal{A}, S_{t_j}, F_{t_j})$, where the decision variables are the allocation $\mathcal{A}$, the start S_{t_j} and finish time F_{t_j} for all tasks $t_j \in \mathcal{T}$. OPT-Duo minimizes the average of the makespan and the sum of all the travel times of the individual robots. Both optimization are solved using Gurobi [16].

VII. Results and Discussion

Sensitivity Results. In the right table in Fig. 2 we report statistics for the makespan and distance values for different

Task Id	ES	LF	DU
0	60	600	20
1	90	600	20
2	100	800	20
3	150	800	20
4	70	600	20
5	120	600	20
6	150	800	20
7	100	800	20

		β							
		0.1		0.5		0.7		0.9	
		μ	δ	μ	δ	μ	δ	μ	δ
		Makespan Values (minutes)							
α	0.1	3.86	0.72	**3.64**	0.67	3.85	1.51	3.64	0.81
	0.5	4.23	1.73	3.74	0.98	**3.49**	0.52	**3.61**	0.66
	0.9	**3.53**	0.76	3.91	1.02	4.21	1.29	4.14	0.68
		Distance Values (meters)							
α	0.1	153.33	43.20	**131.07**	36.35	**114.85**	25.81	150.22	48.37
	0.5	**126.43**	34.51	142.69	34.99	142.69	34.99	**144.27**	35.33
	0.9	131.23	46.22	145.47	37.31	150.22	48.37	153.14	45.59

Fig. 2. Simulation experiments with 2 robots and 8 tasks. The left table shows earliest start, latest finish times, and duration of tasks for the case in Fig. 4. The right table shows makespan and total distance for pIA for different values of α and β. Results averaged over 10 random precedence graphs.

	Configuration	Makespan (minutes)		Distance (meters)		Idle Time (minutes)		%Tasks completed	
		μ	δ	μ	δ	μ	δ	μ	δ
pIA	2 robots, 8 tasks	3.85	1.51	114.85	25.81	2.35	0.97	100	0.00
	2 robots, 16 tasks	9.52	2.10	418.24	35.52	0.00	0.00	100	0.00
Greedy	2 robots, 8 tasks	4.02	0.44	159.47	13.71	0.00	0.00	100	0.00
	2 robots, 16 tasks	9.44	1.64	494.99	93.36	0.00	0.00	100	0.00
OPT-M	2 robots, 8 tasks	3.47	0.48	139.58	39.85	0.00	0.00	100	0.00
	2 robots, 16 tasks	**7.95**	0.70	410.60	21.47	0.00	0.00	100	0.00
OPT-Duo	2 robots, 8 tasks	**2.94**	0.17	**99.76**	6.08	0.00	0.00	100	0.00
	2 robots, 16 tasks	9.00	2.12	**307.21**	13.48	0.00	0.00	100	0.00

Fig. 3. Simulation results comparing the makespan, total distance traveled, total idle time, and completion percentage for pIA, greedy auction, and optimal solutions with makespan only (OPT-M) and makespan and distance combined (OPT-Duo). Eight and 16 tasks are allocated to two robots in the simulated environment in Fig. 1 (left). Minimum values are bold.

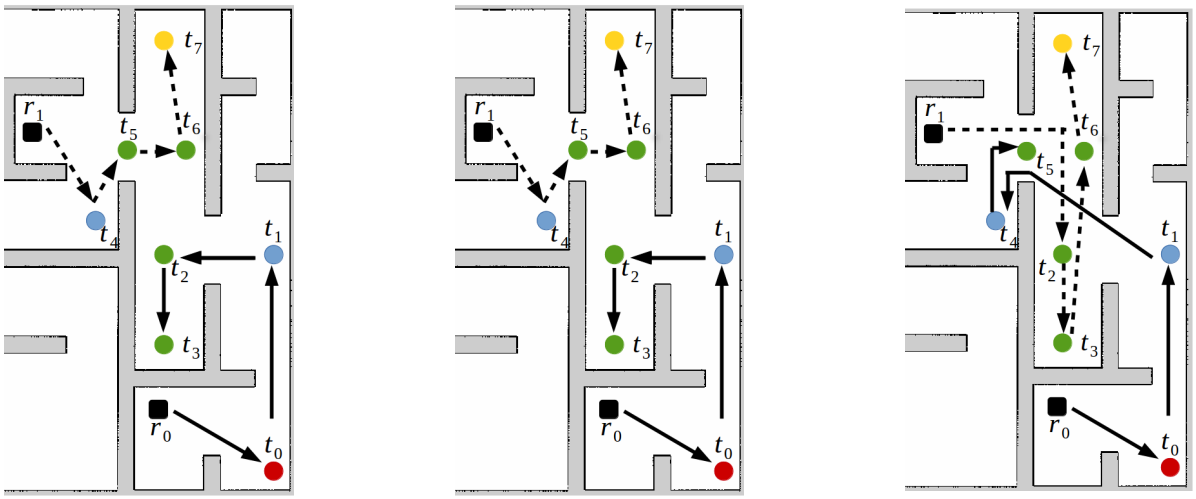

Fig. 4. Routes produced by OPT-Duo (left; makespan 2.87, distance 96.88), pIA (middle; makespan 3.06, distance 96.88) and Greedy method (right; makespan 4.03, distance 154.26) for an instance with 8 tasks and two robots. Makespans are measured in minutes and distances in meters. The colors represent tasks' precedence where red precedes blue, blue precedes green, and green precedes yellow.

values of α and β. Higher values of α increase the importance of makespan, lower values increase the importance of travel time. Higher values of β place more weight on combined duration and travel values of task chains, lower values place more weight on durations of task chains alone. Our pIA auction registers up to 33% change in distances (38.5 meters) and 17% in makespan values (less than a minute) as the α and β parameters change. No parameter combinations result in non-dominated solutions. For lower α values (0.1-0.5) the best makespan values are obtained by using $\beta \geq 0.5$, this is also partly true for distances. We have not observed much advantage in setting α values very high. Roughly, the parametric analysis shows that the algorithm performs better when more weight is placed on distance-based measures.

Comparing the Methods. In Fig. 3 pIA finds solutions with

paths that are nearly 28% and 16% shorter than the paths returned by the Greedy algorithm for the eight and 16-task cases, respectively. The makespan of the schedules returned by the methods are not statistically different. pIA's solutions are also competitive compared to optimal solutions that only consider makespan as objective, and are less than twice the distance and makespan values returned by OPT-Duo. The performance differences are more evident in the 16 task case.

Results with Real Robots.

	Configuration	Makespan (minutes)		Distance (meters)	
		μ	δ	μ	δ
pIA	3 robots, 12 tasks	**4.20**	0.52	132.5	1.5
Greedy	3 robots, 12 tasks	6.21	1.20	152.41	1.80
OPT-M	3 robots, 12 tasks	6.40	0.48	**99.1**	1.35

Fig. 5. Real robot results comparing makespan and total distance traveled for pIA, greedy auction, and OPM-M solutions. All the tasks are allocated and there is no idle time.

With the real robots the initial allocations were computed with OPT-M, pIA, and Greedy. Results are shown in Fig. 5. The allocation returned by pIA yields a Manhattan distance (132.5) that is nearly 34% longer than OPT-M (99.1) and nearly 13% shorter than the one returned by Greedy (152.41). The differences in makespan values are more modest, pIA schedules are best with a makespan of about 4 minutes, while both OPT-M and Greedy yield schedules of about 6 minutes. Unlike pIA and the Greedy auction, OPT-M only uses two of the three available robots, which explains in part why the algorithm's makespan is larger than pIA's.

Analysis. Our parametric analysis shows that placing more emphasis on the distance objective yield schedules with shorter distances. The same is not true for makespan values. This is partly due to the large distances robots (both real and virtual) have to travel to get to the tasks, and the delays that occur due to re-planning. The results could differ for datasets with very small distances and far apart time windows, because the time windows would dominate the allocation decisions.

Comparison with other methods shows that pIA has a clear advantage over the greedy method when both distance and makespan are considered. Its schedules yield distance and makespan values that are not larger than twice the optimal allocations. Part of the success depends on our careful selection of tasks to auction and balancing of spatial and temporal objectives. However, we do not guarantee that our method will always yield results less than twice larger than optimal; data instances can be designed that produce results similar to the Greedy algorithm. Lastly, all the tasks in our experiments were completed without re-auctioning.

VIII. Conclusions and Future Work

We extended the pIA algorithm to allocate to multiple robots tasks with temporal constraints in addition to precedence constraints, and we presented an executor that monitors the execution of the tasks and reallocates tasks when failures or delays will cause constraint violations. pIA is used by the auctioneer to initially allocate tasks to robots, forming a schedule for each robot. Robots execute their initial schedules, and send to the auctioneer information about their tasks start and finish times and execution status. In case of failure or delays, a single-item auction is run to try to reallocate tasks. Our experimental results show that our method outperforms a Greedy method and yield schedules with distances that are within two away from the optimal. Future work will focus on designing more constrained data sets that will further show the robustness of our method.

Acknowledgments: Partial support provided by the National Science Foundation (under grants NSF IIP-1439728, NSF CNF-1531330) and the Doctoral Dissertation Fellowship program from the University of Minnesota.

References

[1] D. Feillet, P. Dejax, M. Gendreau, and C. Gueguen, "An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems," *Networks*, vol. 44, no. 3, pp. 216–229, 2004.

[2] G. A. Korsah, A. Stentz, and M. B. Dias, "A comprehensive taxonomy for multi-robot task allocation," *The International Journal of Robotics Research*, vol. 32, no. 12, pp. 1495–1512, 2013.

[3] R. Dechter, I. Meiri, and J. Pearl, "Temporal constraint networks," *Artificial Intelligence*, vol. 49, no. 1-3, pp. 61–95, 1991.

[4] M. McIntire, E. Nunes, and M. Gini, "Iterated multi-robot auctions for precedence-constrained task scheduling," in *Int'l Conf. on Autonomous Agents and Multi-Agent Systems*, 2016, pp. 1078–1086.

[5] E. Nunes and M. Gini, "Multi-robot auctions for allocation of tasks with temporal constraints," in *Proc. AAAI Conf. on Artificial Intelligence*, 2015, pp. 2110–2116.

[6] R. T. Maheswaran, M. Tambe, E. Bowring, J. P. Pearce, and P. Varakantham, "Taking DCOP to the real world: Efficient complete solutions for distributed multi-event scheduling," in *Int'l Conf. on Autonomous Agents and Multi-Agent Systems*, 2004, pp. 310–317.

[7] R. Junges and A. L. C. Bazzan, "Evaluating the performance of DCOP algorithms in a real world, dynamic problem," in *Int'l Conf. on Autonomous Agents and Multi-Agent Systems*, 2008, pp. 599–606.

[8] S. Ramchurn, A. Farinelli, K. Macarthur, M. Polukarov, and N. Jennings, "Decentralised coordination in RoboCup Rescue," *The Computer Journal*, vol. 53, no. 9, pp. 1–15, 2010.

[9] M. G. Lagoudakis, M. Berhault, S. Koenig, P. Keskinocak, and A. J. Kleywegt, "Simple auctions with performance guarantees for multi-robot task allocation," in *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2004.

[10] L. Barbulescu, Z. B. Rubinstein, S. F. Smith, and T. L. Zimmerman, "Distributed coordination of mobile agent teams: the advantage of planning ahead," in *Int'l Conf. on Autonomous Agents and Multi-Agent Systems*, 2010, pp. 1331–1338.

[11] S. Sariel, T. Balch, and N. Erdogan, "Incremental multi-robot task selection for resource constrained and interrelated tasks," in *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, Oct 2007, pp. 2314–2319.

[12] E. Jones, M. B. Dias, and A. T. Stentz, "Time-extended multi-robot coordination for domains with intra-path constraints," in *Robotics: Science and Systems (RSS)*, July 2009.

[13] L. Luo, N. Chakraborty, and K. Sycara, "Multi-robot algorithm for tasks with set precedence constraints," in *Proc. IEEE Int'l Conf. on Robotics and Automation*, 2011, pp. 2526–2533.

[14] M. Wilson, N. Roos, B. Huisman, and C. Witteveen, "Efficient workplan management in maintenance tasks," in *Proc. 23rd Benelux Conference on Artificial Intelligence*, nov 2011, pp. 344–351.

[15] B. Gerkey, R. Vaughan, and A. Howard, "The Player/Stage project: Tools for multi-robot and distributed sensor systems," in *11th Int'l Conf. on Advanced Robotics*, 2003.

[16] I. Gurobi Optimization, "Gurobi optimizer reference manual," 2014. [Online]. Available: http://www.gurobi.com