FULL PAPER

Decentralized Multi-Robot Allocation of Tasks with Temporal and Precedence Constraints

E. Nunes[a], Mitchell McIntire[b], and M. Gini[a]

[a] Department of Computer Science and Engineering, University of Minnesota, 200 Union St SE, Minneapolis. MN 55455, US.; [b]Computer Science Department, Stanford University, US

ARTICLE HISTORY
Compiled September 19, 2017

ABSTRACT
We present an auction-based method for a team of robots to allocate and execute tasks that have temporal and precedence constraints. Temporal constraints are expressed as time windows, within which a task must be executed. The robots use our priority-based iterated sequential single-item auction algorithm to allocate tasks among themselves and keep track of their individual schedules. A key innovation is in decoupling precedence constraints from temporal constraints and dealing with them separately. We demonstrate the performance of the allocation method and show how it can be extended to handle failures and delays during task execution. We leverage the power of simulation as a tool to analyze the robustness of schedules. Data collected during simulations are used to compute well-known indexes that measure the risk of delay and failure in the robots' schedules. We demonstrate the effectiveness of our method in simulation and with real robot experiments.

KEYWORDS
Task allocation; temporal constraints; precedence constraints

1. Introduction

Service robots that operate in large areas have often tasks that are distributed in space and require to be executed within assigned time intervals while satisfying specific precedence constraints. Examples include robots in warehouses, hospitals, and offices. In a warehouse, robots might need to prioritize packages going to some locations over others, or have to move packages from start to end points within specific time intervals.

In this paper we study the temporal and precedence constrained multi-robot task allocation problem with homogeneous robots. The problem is NP-hard even for a single-robot and without precedence constraints, as we can reduce the elementary shortest path problem with resource constraints [1] to our problem. It is therefore infeasible in general to compute exact solutions to these problems.

This problem falls under the XD[ST-SR-TA = Single-Task robot, Single-Robot task, Time-extended Assignment] category of the Multi-Robot Task Allocation (MRTA) taxonomy given in [2], since precedence constraints impose cross-schedule dependencies, so any delay or execution failure in one robot's schedule will affect other robots' schedules. We assume robots can execute at most one task at a time (ST), tasks require

CONTACT Author. Email: gini@umn.edu

a single robot (SR), and tasks are scheduled over a planning horizon (TA).

We defined the class of multi-robot task allocation problems with temporal and ordering constraints as MRTA/TOC [3], and extended the MRTA taxonomy to include temporal constraints expressed in the form of time windows (TA:TW) and ordering constraints expressed in the form of synchronization and precedence constraints (TA:SP). In this paper we handle both types of temporal constraints. Unlike many other MRTA problems, this problem in its general form, allowing arbitrary precedence constraints and time windows for tasks, has not been thoroughly studied, due in part to its complexity even for approximate solutions.

Our main contribution is an auction-based algorithm, TePSSI (Temporal and Precedence constrained Sequential Single-Item auction), which provides a decentralized method of computing a solution for task allocation problems with temporal and precedence constraints. In our approach each robot owns its schedule for the subset of tasks assigned to it. A schedule is represented as a simple temporal network (STN) [4], which stores the execution times of the tasks. In this paper, we describe the auction algorithm, analyze its complexity, prove its soundness and completeness when only precedence constraints are considered, and demonstrate the algorithm's performance empirically. We also show how to combine the task allocation algorithm with an executive that monitors the execution of the tasks, reallocating tasks via a one-shot greedy auction when needed because of execution delays or failures. The framework herein proposed supports task execution and recovery via a planning-execution-replanning cycle. We present an experimental evaluation of the framework in simulation and through experiments with real robots.

In our previous work we designed TeSSI (Temporal Sequential Single-I tem auction) [5] and pIA (Prioritized Iterated Auction) [6], to handle general temporal and precedence constraints independently, and only at planning time. The auction herein proposed is, to the best of our knowledge, the first auction-based algorithm that is designed to handle general precedence and temporal constraints simultaneously, whilst also considering plan execution aspects.

2. Related Work

Methods for multi-robot task allocation can be broadly categorized into centralized, decentralized, and hybrid; and depending on the optimality of the solution, exact or heuristic. A recent example of a centralized method uses an efficient mixed-integer linear programming approach for multi-robot scheduling with spatial constraints [7]. Centralized methods can achieve optimal results, but are not suitable for field operations where communication can be limited and unreliable, and faults are common. Hence, we choose a decentralized approach.

Distributed Constraint Optimization Problem (DCOP) [8] algorithms provide a viable option for modeling constraint problems in a distributed way. However, solving DCOP exactly is NP-hard and impractical even for unconstrained Multi-Robot Task Allocation (MRTA) problems [9]. Approximate methods such as Max-Sum have been proposed [10, 11], 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 [12, 13]. Auctions move the burden of computation onto individual robots and are robust to local changes or failures, since the auction can proceed with the remaining robots when some robots malfunction [14].

For the simple case of the MRTA problem in which robots are homogeneous and need only to visit tasks in order to complete them, the sequential single-item (SSI) auction from [15] gives a 2-approximation for the total sum-cost of the resulting schedule and a 2m-approximation (where m is the number of robots) for the makespan [16], which is the difference between the overall latest finish and the overall earliest start time. Combinatorial auctions in which robots bid on combinations of tasks can produce good schedules, but are expensive to compute. Zheng et al. [17] show that borrowing combinatorial methods can improve the SSI auction scheme without incurring high computation costs. The SSI auction has been shown both effective and computationally inexpensive compared to other auction schemes [18], but it does not handle precedence and temporal constraints. We have extended SSI to tasks with temporal constraints and proposed the Temporal Sequential Single-Item auction (TeSSI) algorithm [5].

A decentralized solution to tasks with temporal constraints has been proposed in [19], 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 computation and communication costs needed to update the temporal model when the environment changes rapidly.

In [20], a distributed solution is presented for tasks that have precedence and resource constraints, but no temporal constraints. However, this solution computes essentially a greedy schedule initially, and then adapts this schedule during task execution. Our goal by contrast is to compute the entire schedule initially, so no communication is required during task execution. In [21] 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. [22] recently introduced an auction-based algorithm for multi-robot assignment of tasks for a particular special case, in which precedence constraints are in a form called *set* precedence constraints (SPCs). The SPC problem partitions tasks into sets with size at most equal to the number of robots. These sets are strictly ordered, giving a precedence relationship. The resulting problem is similar to ours, but heavily constrains the space of allowable precedence graphs. In particular, if we view the precedence graph by layers (described in more detail later), SPCs require that every task in a layer has a precedence constraint with every task in the next layer. This is equivalent to saying that the topological ordering of an SPC precedence graph is unique up to changing the order of tasks within each set. They also require that robots complete at most one task per set. This method is therefore too restrictive to apply to our problem.

3. 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. A robot need to arrive to a task before its

latest start time LS_{t_j}, which, if not specified, can be computed as $LS_{t_j} = LF_{t_j} - DU_{t_j}$.

The objective is to find a route for each agent, such that (1) each task location is visited once by one agent, (2) all locations are eventually visited, and (3) the team optimizes an objective function. We assume a non-negative objective function that represents the cost of moving between two locations. Typical functions used in the literature [23] are: MiniSUM, i.e. minimize the sum of the agent path costs over all the agents; MiniMAX, i.e. minimize the maximum agent path cost over all the agents; MiniAVE: i.e. minimize the average path cost to reach all the tasks. An agent's path cost is the sum of the costs along an agent's entire path, from its initial location to the last task location on its path. Minimizing the makespan (i.e., the total duration of the schedule) is similar to MiniMAX, but costs are measured in terms of time.

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 temporal and precedence constraints.

4. Temporal and Precedence constrained Sequential Single-Item auction

Our Temporal and Precedence constrained Sequential Single-Item auction (TePSSI) combines the Prioritized Iterated Auction (pIA) [6] algorithm, with a modified version of the TeSSI algorithm [5] to enable scheduling tasks with both temporal and precedence constraints.

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 TeSSI. TeSSI uses a variant of the sequential single-item auction algorithm [16], where each robot maintains information on its allocated tasks using a simple temporal network (STN).

In explaining the operation of the iterated auction algorithm, it is helpful to think about the task precedence graph $G_{\mathcal{P}}$ as a DAG with nodes grouped into layers. We use the term 'free' to describe task nodes with no parents, and for a set of tasks A we let free(A) be the set of tasks in A which have no predecessors in A

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 [6] 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 the travel time, tt, 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

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

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 the 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. Tasks in T_F are auctioned in a sequential single-item auction. In each iteration of the auction, a subset of high priority tasks in T_F is auctioned-off.

Upon receiving the 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

$$\text{bid} = \alpha \times m + (1-\alpha) \times \Delta tt \tag{2}$$

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. Notice that adding a task does not necessarily increase the makespan, since the task could fit into an empty slot in the robot schedule. Δtt is included in the bid to penalize bids from robots that are far away from a task compared to robots that are closer to the task.

Each robot sends to the auctioneer only its smallest bid, i.e. the one that minimizes the cost function. 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 continues the auction with the remaining tasks.

Temporal Model and Validity Checking. Temporal constraints are modeled as a simple temporal network [4]. 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 [24]. 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 $F_{t_k} = S_{t_k} + DU_{t_k}$. If any of the start times has the value ∞ it means that the schedule is inconsistent, in which case the robot cannot do the task.

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.

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

Soundness and Completeness. We will now outline an informal proof that our algorithm will always produce a valid schedule if one exists, when only precedence constraints are taken into account. Note that we allow general precedence constraints, and so any set of tasks whose precedence graph is a DAG is valid. Furthermore, a valid schedule will always exist in this case, since we can simply assign every task to a single robot, which can complete the tasks in any topological ordering of the precedence graph. For the rest of this section, we will assume that the problem instance is valid, with a directed acyclic precedence graph. We therefore only need to show that the algorithm produces a valid schedule in this case. We assume that priorities are nonnegative and that the priority of every task is at least as high as that of any of its successors in the precedence graph.

First, we show that the algorithm will successfully schedule all the tasks and terminate. In every iteration the number of tasks auctioned is > 0, and therefore the number of tasks scheduled increases in every iteration until all tasks are scheduled.

Now assume instead that the second layer T_L is empty. If T_L is empty there must be at least one task in T_F, as otherwise every task would already have been scheduled. Since we assume that priorities are nonnegative, every task in T_F will be added to T_{auct}, and so T_{auct} will be nonempty. Thus in every iteration at least one task is scheduled, and so the algorithm terminates.

Next, we must show that the schedule produced by the algorithm is valid. It is only invalid if precedence constraints are violated. Note that the tasks scheduled in any given iteration are pairwise independent, so tasks with precedence constraints must have been scheduled in an earlier iteration.

A similar analysis can be used to show that soundness is guaranteed also for temporal constraints. However, completeness cannot be guaranteed in the presence of

temporal constraints. The task prioritization algorithm does not take time windows into account. For this reason, it is possible for tasks to be auctioned in an order that violates the time window constraints for tasks that appear later in task chains.

5. Execution of Allocated Tasks

When tasks are allocated, robots communicate with each other and the auctioneer via ROS (Robot Operating System) [25] publishers and subscribers. We run an auction topic in which requests for bids and bids are sent back and forth. For simplicity, bidding is done synchronously. The auctioneer waits until all robots send their bids or a timeout is reached, prior to choosing the winner. There is a timeout to account for cases where robots fail to submit their bid.

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

Task Execution by the Robots and Role of Executive. The result of the auction process is a multi-robot schedule, with a set of m sub-schedules, one per robot. Each of them contains a sequence of tasks, ordered according to the time they will be performed. Each task t_j appears only in one sub-schedule and has a planned start time Δ_{t_j}. The number of tasks scheduled could be smaller than n if some tasks could not be scheduled.

When the execution starts, a robot retrieves the first task in its schedule. In each ROS cycle, the robot computes the estimated arrival time to its upcoming task by adding to the current time the estimated remaining travel time to the task's location. If the estimated arrival time is smaller than the start time for the task in its schedule, the robot continues execution. If not, there are different cases. If the estimated arrival time is smaller than the task's latest start time, the task can still be executed, but the delay could cause inconsistencies with other tasks for which this task is a precedence constraint. Since those other tasks might have been assigned to other robots, the delayed robot needs to check with the executive, for any potential violation. If there are no violations, the robot assigns the estimated arrival time as the task's start time and continues executing. Otherwise, the robot marks the task as *aborted* since it is unable to execute it within its temporal constraints. At that point, the executive runs an auction to try to reallocate that task to another robot, if possible, otherwise marks the task as *failed*. When a robot has completed the execution of a task, it marks it as *succeeded*, notifies the executive, and proceeds to its next task. This way the executive can keep track of the overall progress.

Monitoring by the Executive. As execution unfolds the executive monitors the execution of the tasks by subscribing to ROS topics in which robots post their tasks' execution status. In each ROS cycle it checks if a task has been completed. The auctioneer keeps a DAG that encodes precedence constraints and start and finish times of tasks, which are updated during execution. When a robot has a potential time update, the executive proceeds by temporarily updating the finish times of all remaining tasks in the DAG. Next, it does a topological sort on the DAG to compute a linear ordering according to the precedence constraints. It then uses the sorted graph to compute a new start time $\tilde{S}_{t_k}$ for all tasks. If $\tilde{S}_{t_k} \leq LS_{t_k}, \forall t_k$, the executive accepts the temporal updates and the robot is sent an "OK" message. Otherwise, the executive

rejects the time updates, the robot aborts the task, and the executive resets the task start time to its previous value.

When a task is marked as aborted, the executive starts an auction for that task to see if any other robot can add it to its schedule. Each robot computes a bid, as during planning, and sends it to the executive, 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, there is no attempt to execute failed tasks.

We found it convenient to use the ROS subscribing mechanism to share information among robots and executive. Having the executive keep track of the overall progress of all the tasks keep things consistent, but can create a bottleneck when the number of tasks and robots is very large, Individual robots could be given more control in monitoring the execution, but this will make it harder to deal with potential catastrophic failures of the robots.

6. Experimental Setup

Simulation Experiments. Simulation experiments were conducted in ROS [26], using the Gazebo plugin. We used a Gazebomodel for virtual Pioneer 3-DX robots, and a 2D world in which the robots operate.

To facilitate robot localization and motion planning, the simulator keeps a 2D map of the world, but the sensing is done in the 3D model. 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). For simplicity, the travel times used by robots in their bids are computed using the distance between nodes in the graph. The ROS navigation package handles the actual motion of the robots.

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.

Data Generation for Real and Simulated Experiments. 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 is the range of 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 [6]. Each data set is created by keeping the tasks' locations fixed, while the time windows can change.

Benchmark Algorithms. In addition to TePSSI, tasks are allocated using a greedy auction and two modifications, OPT-M and OPT-Duo, of the MILP in [6]. 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 [27].

7. Results and Discussion

7.1. *Simulation Results*

Simulation experiments were conducted in ROS [26], using the Gazebo plugin. The 100×100 meters map shown in part in Fig. 1 contains a total of 40 points, from which we choose task locations and initial positions for the robots.

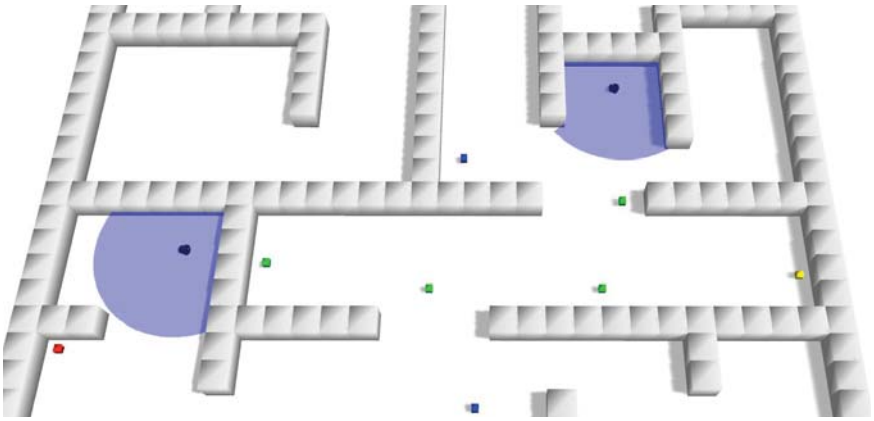

Figure 1. Example indoor Gazebo simulation scenario with two robots and eight tasks. Tasks are shown colored small cubes. The colors represent dependencies between tasks: the precedence order is red, blue, green, and yellow. The goal is to minimize the time to clear the last hazard or the distance covered.

Table 1. A specific example of time windows with eight tasks.

Task Id	EarlyStart	LateFinish	Duration
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

For the experiments, we generated 10 data sets for different number of tasks (8 and 16 tasks used), keeping the task locations fixed but changing the time windows. We present now a sensitivity analysis to see how results are affected by the choice of the parameters α and β. Then we compare the performance of TEPSSI against other algorithms.

Table 2. Simulation experiments with 2 robots and 8 tasks. The table shows makespan and total distance for TePSSI for different values of α and β. Results averaged over 10 random precedence graphs for a total of 120 simulation runs.

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

7.1.1. Sensitivity Analysis Results

In Table 2 we report statistics for the makespan and distance values for different values of α and β. α is used in the optimization function, β is used when prioritizing tasks with precedence constraints. Higher values of α increase the importance of makespan, lower values increase the importance of travel time in the value of the bid (see Eq. 2). Higher values of β place more weight on combined duration and travel times of task chains, lower values place more weight on durations of task chains alone ignoring travel times (see Eq. 1).

The TEPSSI 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.

7.1.2. Comparison of Methods

Table 3 shows that TEPSSI 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. TEPSSI'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.

Figure 2 shows examples of the solutions found by TePSSI, OPT-DUO, and a greedy algorithm. The colors of the tasks in the figure reflect their precedence constraints. The allocations produces by TePSSI and OPT-DUO are very similar, while the greedy algorithm produces much worse allocations.

Table 3. Simulation results comparing the makespan, total distance traveled, total idle time, and completion percentage for TEPSSI, greedy auction, OPT-M, which finds optimal solutions using makespan only, and OPT-Due, which finds optimal solutions using and makespan and distance combined. μ is the mean and σ the standard deviation. Eight and 16 tasks are allocated to two robots in the simulated environment in Fig. 1 (left). Minimum values are bold.

	Configuration	Makespan (minutes)		Distance (meters)		Idle Time (minutes)		%Tasks completed	
		μ	σ	μ	σ	μ	σ	μ	σ
TePSSI	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

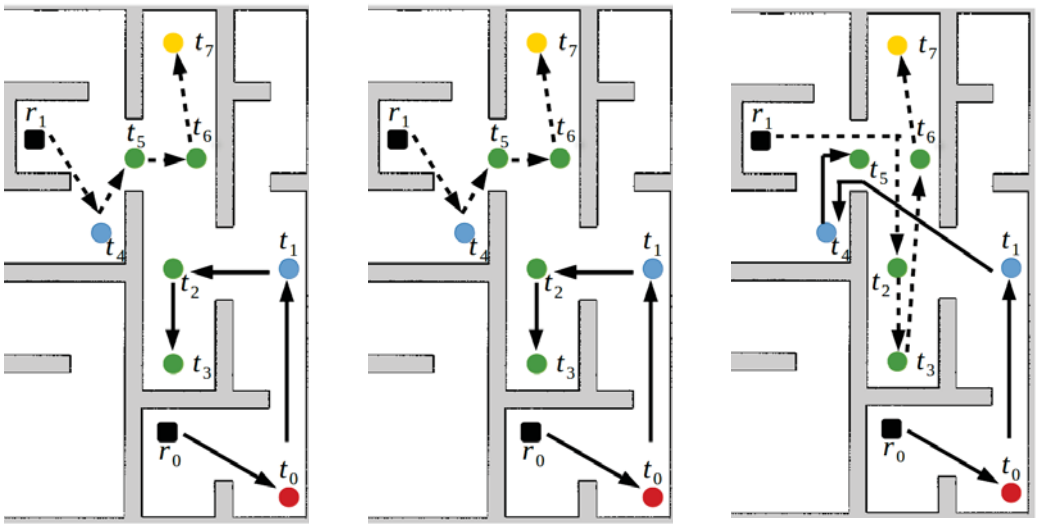

Figure 2. Routes produced by OPT-Duo (left; makespan 2.87, distance 96.88), TePSSI (middle; makespan 3.06, distance 96.88) and Greedy method (right; makespan 4.03, distance 154.26) for an instance with 8 tasks and two robots. Makespans are measured in minutes and distances in meters. The colors represent tasks' precedence where red precedes blue, blue precedes green, and green precedes yellow. The temporal constraints on the tasks are in Table 1.

7.2. *Results with Real Robots*

The experiments with real robots were run on a map (54 by 51 meters) of a room and corridors (see the left part of Fig. 3). Three TurtleBot 2 had to do 12 tasks distributed around the space. The results for these experiments are averaged over five runs.

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

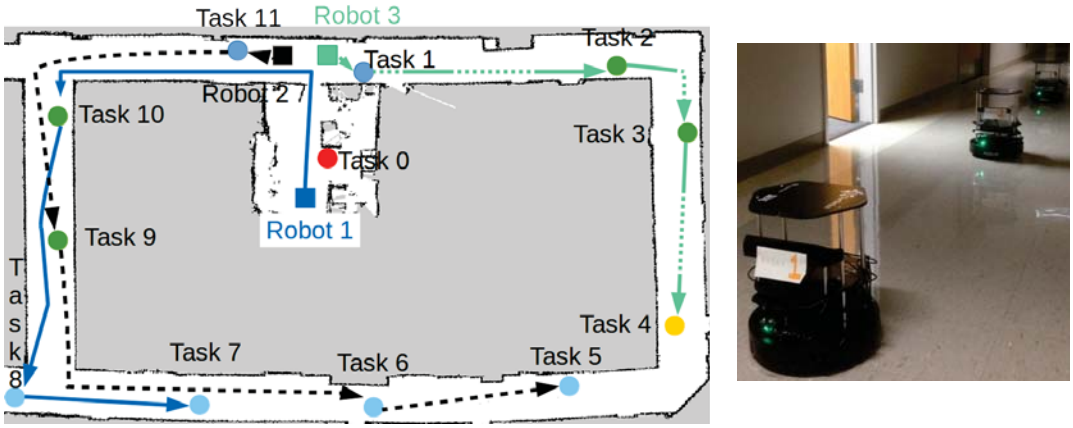

Figure 3. (Left) Scenario used with real robots. The colors represent dependencies between tasks: the precedence order is red, blue, green, and yellow. The goal is to minimize the time to complete the last task or the distance covered. (Right) Three Turtlebot 2 robots used for our physical robot experiments, which operated in the room and corridor environment shown.

Table 4. Real robot results comparing makespan and total distance traveled for TePSSI, greedy auction, and OPT-M solutions. All the tasks are allocated and there is no idle time. Results averaged over five runs.

	Configuration	Makespan (minutes)		Distance (meters)	
		μ	σ	μ	σ
TePSSI	3 robots, 12 tasks	**4.20**	0.52	132.50	1.50
Greedy	3 robots, 12 tasks	6.21	1.20	152.41	1.80
OPT-M	3 robots, 12 tasks	6.40	0.48	**99.10**	1.35

robots, which explains in part why the algorithm's makespan is larger than TePSSI's.

7.3. *Analysis*

Our parametric analysis shows that placing more emphasis on the distance objective yields 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 TePSSI 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.

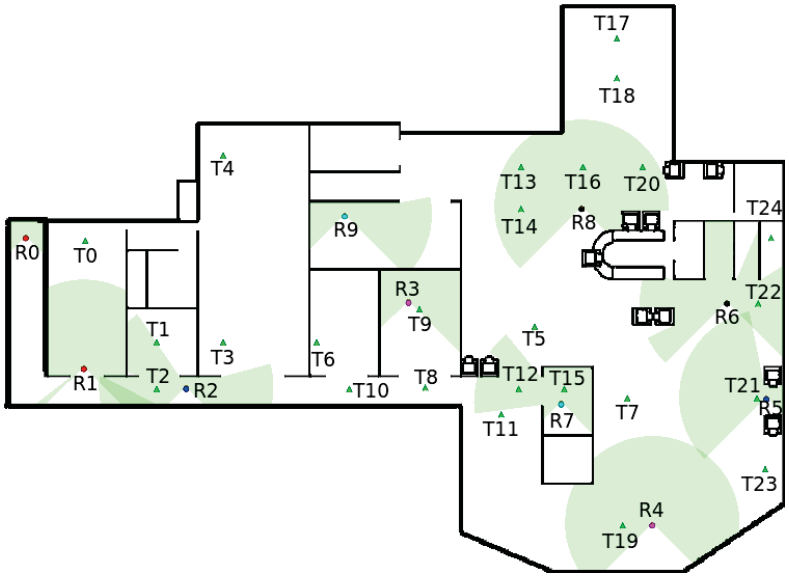

Figure 4. The map used for risk analysis. Robots are the colored circles labeled $R_{0,\ldots,9}$ and tasks are the green triangles labeled $T_{0,\ldots,24}$

8. Schedule Execution and Uncertainty

The goal in this section is to propose the use of simulation as a tool to identify and estimate some of the sources of schedule execution failures. This is a first step in incorporating uncertainty during task allocation and execution. Uncertainty is modeled via probabilities that ultimately can be used with probabilistic STN models [28]. We also discuss risk measures, which can be used as part of the computation of TePSSI. The inclusion of probabilities and risk measures into the STN model and the auction are left for future work.

Specifically, in this paper we are interested in using a large number of simulations to compute:

(1) $P(st_{t_j} \leq \Delta_{t_j})$, the probability that the start time of task t_j at execution time is within its planning time;
(2) $P(st_{t_j} \leq ls_{t_j})$, the probability that the start execution time of t_j is smaller than the task's latest planned start time.

For these experiments we simulated robots using ROS/Stage robot and world models [29] to allow us to scale the number of robots used. Each robot is equipped with a simulated Hokuyo ranger sensor for sensing. We used a 52×37 meter map (see Figure 4), with 10 robots and 25 tasks. Tasks temporal constraints are drawn from distributions similar to those used in Section 7. For the constraint graph generation we limited the number of immediate children for any node to be at most 4 (found experimentally), however we do not limit the length of any task chain.

We generate a discretization of the map by overlaying a grid over the map (see Figure 5). We tried grids of different sizes and found that a 4×4 meter cell provided enough resolution for our analysis. We use the grid to gather statistics for tasks' start times and for the time robots spent on each cell of a grid. The start times statistics in the right part of Figure 5 were collected over 100 runs on the same robots' schedules.

Using the global plan returned by the ROS navigation we generate timed path traces

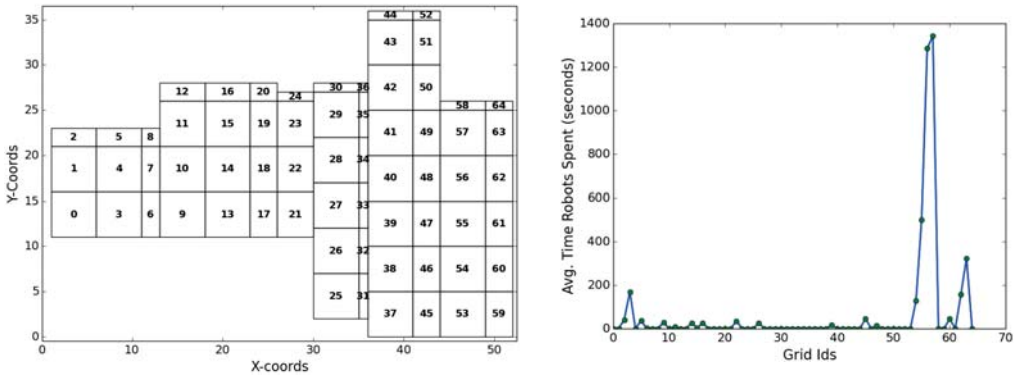

Figure 5. (Left) 4×4 meter grid decomposition of the map. (Right) Distribution of traversal times over the grid. Results collected over 100 simulated execution runs.

for the robots, where to each (x, y) location we attach a timestamp indicating when the robot arrived to the location. For each set of locations in the robot path that are in the same cell we compute the time the robot spends in the cell by calculating the difference between timestamps. The differences are computed across all robot runs. The time distribution is averaged over all robots that have paths that visit the same cells. We also run additional simulations where we vary tasks' locations but keep time windows, precedence constraints, and robots' locations the same. By varying tasks' locations we create traversal time profiles for more cells in the map.

The traversal time distribution (right part in Figure 5) per cell shows that for most cells the robot traversal time is under 7 minutes, except for the regions between 56-58 where robots spend much longer. This is partly explained by the fact that when operating in that area robots sometimes cannot properly localize themselves and they get stuck in the same place for the rest of the simulation.

8.1. *Toward Risk Models for Scheduling Analysis*

We expect the start times of tasks to vary across runs due to several sources of uncertainty. Here we focus on the delays caused by congestion (too many robots working in the same area), number and shape of fixed obstacles (space is too tight), and sensor and localization errors as the main sources of uncertainty.

Congestion causes the robots to reduce their speeds to avoid collisions, which in turn causes fluctuations in their travel times, and ultimately affects their arrival times to tasks. The number and shape of fixed obstacles (especially in unstructured environments) require robots to slow down to avoid collision with the obstacles, also causing arrival time variations. When robots experience localization and sensor failures they might need to replan, which introduces a time overhead. In this paper we only account for transient failures such as the ones already discussed, we do not directly address permanent failures (e.g., destruction of robots).

Let $\mathcal{B}_{\mathcal{T}} = \{\mathcal{B}_{t_1}, ..., \mathcal{B}_{t_j}, ..., \mathcal{B}_{t_n}\}$ be the set of random variables for the tasks' start times, one per task. We treat $\mathcal{B}_{t_j}$ as a continuous random variable. We assume hard temporal constraints, for this reason the probability mass for each $\mathcal{B}_{t_j}$ is non-zero only within the task's time windows ($[es_{t_j}, ls_{t_j}]$).

We also consider dependencies among $\mathcal{B}_{t_j}$ variables. The structure of these dependencies is due to precedence constraints over the set of tasks and task sequencing in the robots schedules. The dependencies are also encoded in uncertainties of the tasks:

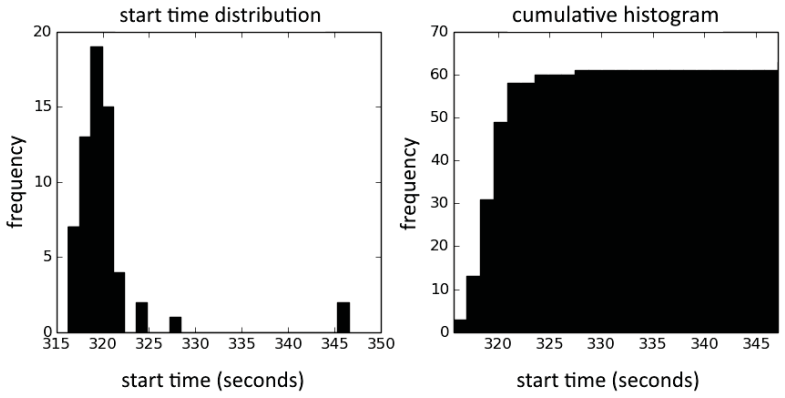

Figure 6. Distribution of start times for task T2 in experiments with 10 robots and 25 tasks.

a late arrival of a robot to a predecessor of a task might delay the arrival to that task too. We use a Bayesian network (BN) to model the dependencies.

A BN is a directed acyclic graph in which random variables that share an edge are dependent. The directed edges define parent-child relationships among random variables, the same way that they describe precedence in our problem. In a BN each variable is conditionally independent of all its non-descendants given all of the variables' parents. This is useful because it allows us to model tasks in the precedence graph that are not dependent on each other (e.g tasks in different subgraphs).

In the BN each $\mathcal{B}_{t_j}$ is parameterized with a conditional probability table that encodes the probabilities of the random variable, given its parents. Given a BN, we can answer queries such as "what is the probability that robot 1 will arrive to task t_3 on time, given that the robot started t_3's predecessors within their planned start times?" To answer the query we compute the following probability:

$$p(\mathcal{B}_{t_3} \leq \Delta_{t_3} | \mathcal{B}_{t_2} \leq \Delta_{t_2}, \mathcal{B}_{t_5} \leq \Delta_{t_5}) = \frac{\mathcal{P}}{\mathcal{Q}}$$

where

$$\mathcal{P} = p(\mathcal{B}_{t_2} \leq \Delta_{t_2}, \mathcal{B}_{t_3} \leq \Delta_{t_3}, \mathcal{B}_{t_5} \leq \Delta_{t_5}) \quad and \quad \mathcal{Q} = p(\mathcal{B}_{t_3} \leq \Delta_{t_3})$$

The computation of $\mathcal{P}$ and $\mathcal{Q}$ is intractable, especially for a BN that encodes a large and complex precedence graph. Approximate inference could be employed using methods such as Gibbs sampling, but this would assume that the BN parameters (the conditional probability tables for each $\mathcal{B}_{t_j}$) and structure have already been learned. Fortunately, in this case we would not need to learn the BN structure (which is a harder task), given that these are derived from the precedence and sequencing constraints. However, we still need to learn the BN parameters.

Training data to learn the BN parameters can be obtained in simulation as historical data for tasks' start times (see Figure 6 for an example). To learn the BN parameters we will use a Bayesian update approach: assign a prior probability density function to each $\mathcal{B}_{t_j}$ and use the training data (e.g., Figure 6) to compute a posterior parameter distribution and the Bayes estimates. We roughly estimate the BN parameters experimentally by running many simulations of the robots executing schedules, and collecting statistics about tasks' start times. Probabilities are computed by counting

the number of times tasks started at given times. The use of more advanced parameter estimation techniques is left for future research. This BN can be used as more advanced probabilistic model for probabilistic STN approaches such as the one proposed by Fang et al [28].

The simulation data can also be used to compute risk measures for individual tasks. This level of risk analysis gives the system user granular information about individual task's influence on the overall execution outcome. The designer can then adjust the tasks' temporal constraints to increase the probability of a task being completed on time.

8.2. *Risk measures*

To measure risk we adapt two well-known risk measures, the *schedule sensitivity index* (SSI) and the *critical delay contribution index* (CDC) to our problem. Both indices have been identified as good risk measures in probabilistic project scheduling [30]. SSI measures risk as a task's variance contribution compared to the overall schedule variance. CDC measures risk as individual task's contributions to the overall lateness (difference in makespan between the planned makespan and the execution makespan).

We simplify the computation of SSI and CDC as follows: the variance contribution in SSI (Eq, 3) is weighted by the number of times a robot arrived to a task after the task's planned start time (start time delay). We compute it in each simulation run and sum over all the simulations (see Eq. 5). The same weight is also used to compute CDC values (see Eq. 4).

When computing SSI we normalize individual task's variances by dividing them by the maximum variance over all tasks' start times. This is a departure from the original index, which used the variance of the last task's start time. This distinction is important because the variance of the last overall task's start time depends on the task's predecessors' variances. If the task has very large time window (and temporal flexibility) the time window length may attenuate or even dissipate the effects that previous tasks' variances have on the last task's variance.

$$SSI_{t_j} = \mathrm{Z} \cdot \sqrt{\frac{\mathrm{Var}(\vec{s}_{t_j})}{\max_{t_k \in \mathcal{T}} \mathrm{Var}(\vec{s}_{t_k})}} \tag{3}$$

$$CDC_{t_j} = \mathrm{Z} \cdot \sum_{q=1}^{\mathcal{N}} (m^{plan} - m^q) \tag{4}$$

where

$$\mathrm{Z} = \frac{\sum_{q=1}^{\mathcal{N}} \delta_{t_j}^q}{\mathcal{N} \sum_{t_j \in \mathcal{T}} \sum_{q=1}^{\mathcal{N}} \delta_{t_j}^q} \tag{5}$$

In Equation 5, $\delta_{t_j}^p$ is an indicator function that assumes the value of 1 if the robot assigned to execute task t_j starts the task past its planned start time Δ_{t_j}, and 0 otherwise. This value is collected in each simulation run p over all $\mathcal{N}$ simulation runs. $\vec{s}_{t_j}$ is the vector of start times for task t_j, and $\vec{s}_{max}$ is the vector of tasks' start times

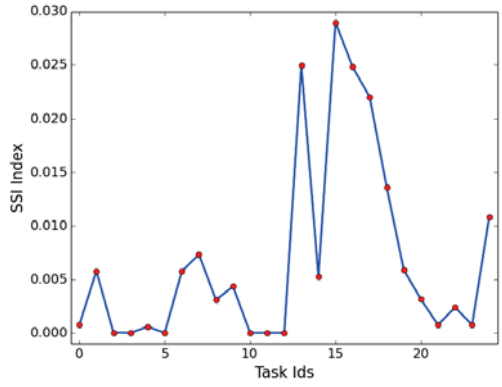

Figure 7. Schedule sensitivity index for 25 tasks executed by 10 robots.

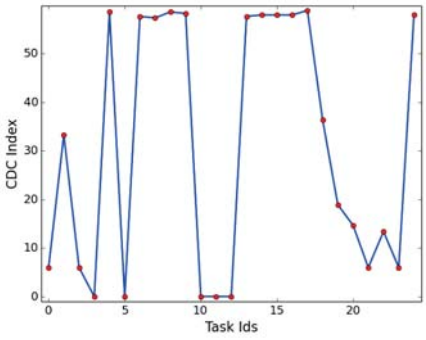

Figure 8. Critical delay contribution for 25 tasks executed by 10 robots.

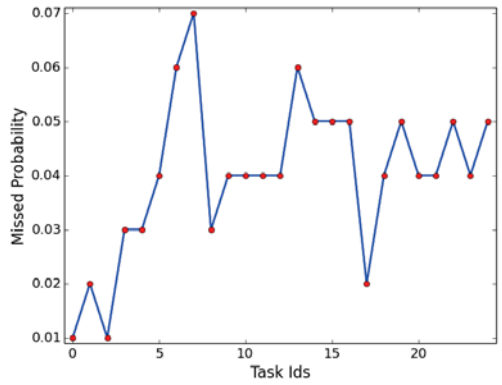

Figure 9. Probability of task failure measured as frequencies over the number of simulations for 25 tasks and 10 robots.

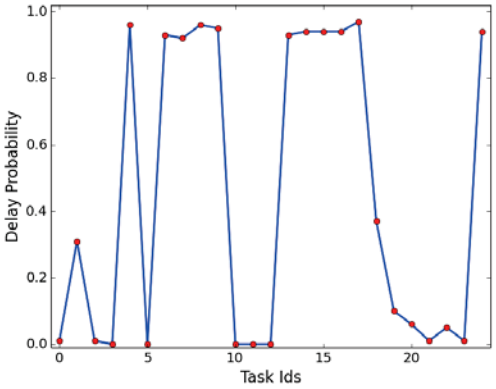

Figure 10. Probability of task delay measured as frequencies over the number of simulations for 25 tasks and 10 robots.

with the maximum variance. These vectors have dimensions $\mathcal{N} \times 1$. In Equation 4, m^{plan} is the makespan of the robots' schedules prior to execution, and m^p is the makespan for simulation number p. $(m^{plan} - m^p < 0)$ indicate lateness in finishing the overall schedule.

8.3. *Preliminary Risk and Probability Results*

We show here preliminary results on tasks' risks and probabilities for the map and tasks illustrated in Figure 4. Figure 7 and Figure 8 show the two measures SSI and CDC, while Figure 9 and Figure 10 show the probabilities of failures and of delay, respectively. The risk measures are computed according to Eqs. 3 and 4. The two risks highlight different information about the tasks' stochastic start times: SSI highlights the importance of tasks' variances, while CDC highlights the importance of delays of individual tasks. In our experiments the tasks with the highest SSI risk are located in the cells with largest traversal times. This supports the hypothesis that tasks with high variance in their travel times, and consequently start times, are more risky. However SSI by itself might not be enough because it does not measure the effects of a tasks' variance on the overall risk of a schedule. For that we need the CDC risk index.

The CDC risk index shows that more tasks are at risk. Part of that is explained by the fact that CDC combines the individual (task level) and overall (schedule-level) de-

lays when computing risk. Tasks for which the start execution time exceeds the planned start time (indicating a delay) are considered more risky. An interesting property is that tasks at the end of long task chains (e.g. tasks with ids 7) have large CDC risks; this is in part due to the fact that delays in earlier tasks are propagated to the later ones in the chain, even if these later tasks do not have high variance in their start times.

Our experiments also show that tasks have low probabilities of failing. Failure happens when a time window constraints are not met during execution. Failure probabilities of tasks in Figure 9 are consistent with the hypothesis that more constrained tasks (e.g., task 7) are more likely to fail. This is also consistent with the delay risk reported in Figure 8. The delay probabilities are consistent with the CDC results, because CDC values use delay frequencies in their computation. The reported low probabilities support our claim on the robustness of our allocation and execution method. Robustness is improved by schedule adjustments and task re-auctioning.

9. Conclusions and Future Work

We extended the pIA and TeSSI algorithms to allocate to multiple robots tasks that have both with temporal and 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. TePSSI is used by the auctioneer to allocate tasks to robots, forming a schedule for each robot. When robots execute their schedules, they send to the executive information about the execution status of their tasks. In case of failure or delays that cannot be dealt with locally by a robot, the executive tries to reallocate tasks to the other robots.

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.

The preliminary analysis of potential execution delays proposed so far is for a fixed schedule and fixed starting positions of robots and tasks. A more general analysis should include simulations over different schedules and locations of robots and tasks. However, learning over the space of all possible schedules is intractable due to the exponential number of schedules. In future work we will incorporate the risk and probabilistic analysis directly in the allocation system, to produce directly more robust schedules.

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] Feillet D, Dejax P, Gendreau M, et al. An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems. Networks. 2004;44(3):216–229.

[2] Korsah GA, Stentz A, Dias MB. A comprehensive taxonomy for multi-robot task allocation. The International Journal of Robotics Research. 2013;32(12):1495–1512.

[3] Nunes E, Manner M, Mitiche H, et al. A taxonomy for task allocation problems with temporal and ordering constraints. Robotics and Autonomous Systems. 2017; 90:55–70.
[4] Dechter R, Meiri I, Pearl J. Temporal constraint networks. Artificial Intelligence. 1991;49(1-3):61–95.
[5] Nunes E, Gini M. Multi-robot auctions for allocation of tasks with temporal constraints. In: Proc. AAAI Conf. on Artificial Intelligence; 2015. p. 2110–2116.
[6] McIntire M, Nunes E, Gini M. Iterated multi-robot auctions for precedence-constrained task scheduling. In: Int'l Conf. on Autonomous Agents and Multi-Agent Systems; 2016. p. 1078–1086.
[7] Gombolay M, Wilcox R, Shah J. Fast scheduling of multi-robot teams with temporospatial constraints. In: Robotics: Science and Systems (RSS); Berlin, Germany; 2013. p. 49–56.
[8] Maheswaran RT, Tambe M, Bowring E, et al. 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. p. 310–317.
[9] Junges R, Bazzan ALC. Evaluating the performance of DCOP algorithms in a real world, dynamic problem. In: Int'l Conf. on Autonomous Agents and Multi-Agent Systems; 2008. p. 599–606.
[10] Farinelli A, Rogers A, Petcu A, et al. Decentralised coordination of low-power embedded devices using the Max-Sum algorithm. In: Int'l Conf. on Autonomous Agents and Multi-Agent Systems; 2008. p. 639–646.
[11] Ramchurn S, Farinelli A, Macarthur K, et al. Decentralised coordination in RoboCup Rescue. The Computer Journal. 2010;53(9):1–15.
[12] Dias MB, Zlot R, Kalra N, et al. Market-based multirobot coordination: A survey and analysis. Proceedings of the IEEE. 2006;94(7):1257–1270.
[13] Korsah GA, Kannan B, Fanaswala IA, et al. Enhancing market-based task allocation with optimal seed schedules. In: Proc. of the Int'l Conf. on Intelligent Autonomous Systems; August; 2010. p. 249–258.
[14] Nanjanath M, Gini M. Repeated auctions for robust task execution by a robot team. Robotics and Autonomous Systems. 2010;58(7):900–909.
[15] Lagoudakis MG, Berhault M, Koenig S, et al. Auctions with performance guarantees for multi-robot task allocation. In: Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems; 2004. p. 1957–1962.
[16] Lagoudakis MG, Markakis E, Kempe D, et al. Auction-based multi-robot routing. In: Robotics: Science and Systems (RSS); 2005. p. 343–350.
[17] Zheng X, Koenig S, Tovey C. Improving sequential single-item auctions. In: Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems; 2006.
[18] Koenig S, Tovey C, Lagoudakis M, et al. The power of sequential single-item auctions for agent coordination. In: Proc. AAAI Conf. on Artificial Intelligence; 2006. p. 1625–1629.
[19] Barbulescu L, Rubinstein ZB, Smith SF, et al. Distributed coordination of mobile agent teams: the advantage of planning ahead. In: Int'l Conf. on Autonomous Agents and Multi-Agent Systems; 2010. p. 1331–1338.
[20] Sariel S, Balch T, Erdogan N. Incremental multi-robot task selection for resource constrained and interrelated tasks. In: Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems; Oct; 2007. p. 2314–2319.
[21] Jones E, Dias MB, Stentz AT. Time-extended multi-robot coordination for domains with intra-path constraints. In: Robotics: Science and Systems (RSS); July; 2009.

[22] Luo L, Chakraborty N, Sycara K. Multi-robot algorithm for tasks with set precedence constraints. In: Proc. IEEE Int'l Conf. on Robotics and Automation; 2011. p. 2526–2533.
[23] Tovey C, Lagoudakis M, Jain S, et al. The generation of bidding rules for auction-based robot coordination. In: Parker LE, Schneider FE, Schultz AC, editors. Multi-Robot Systems: From Swarms to Intelligent Automata, Volume III; Mar. Springer Netherlands; 2005. p. 3–14.
[24] Wilson M, Roos N, Huisman B, et al. Efficient workplan management in maintenance tasks. In: Proc. 23rd Benelux Conference on Artificial Intelligence; nov; 2011. p. 344–351.
[25] Quigley M, Conley K, Gerkey BP, et al. Ros: an open-source robot operating system. In: ICRA Workshop on Open Source Software; 2009.
[26] Gerkey B, Vaughan R, Howard A. The Player/Stage project: Tools for multi-robot and distributed sensor systems. In: 11th Int'l Conf. on Advanced Robotics; 2003.
[27] Gurobi Optimization I. Gurobi optimizer reference manual ; 2014. Available from: http://www.gurobi.com.
[28] Fang C, Yu P, Williams BC. Chance-constrained probabilistic simple temporal problems. In: Proc. AAAI Conf. on Artificial Intelligence; 2014. p. 2264–2270.
[29] Portugal D, Rocha RP. On the performance and scalability of multi-robot patrolling algorithms. In: IEEE Int'l Symp. on Safety, Security, and Rescue Robotics; Nov.; 2011. p. 50–55.
[30] Creemers S, Demeulemeester E, Van de Vonder S. A new approach for quantitative risk analysis. Annals of Operations Research. 2014;213(1):27–65.