Iterated Multi-Robot Auctions for Precedence-Constrained Task Scheduling

Mitchell McIntire
University of Minnesota
200 Union St. SE
Minneapolis, MN 55455
mcint286@cs.umn.edu

Ernesto Nunes
University of Minnesota
200 Union St. SE
Minneapolis, MN 55455
enunes@cs.umn.edu

Maria Gini
University of Minnesota
200 Union St. SE
Minneapolis, MN 55455
gini@cs.umn.edu

ABSTRACT

We consider the multi-robot task scheduling problem with precedence constraints, and introduce a general algorithm to approximate a solution to this problem. Our algorithm utilizes an iterated auction scheme, in which a batch of tasks that are pairwise unconstrained is selected in each iteration and scheduled using a modified sequential single-item auction. This algorithm also gains flexibility by allowing the use of task prioritization to order the scheduling process. We demonstrate the effectiveness of this iterated auction scheme empirically using existing 100- and 1000-task data sets that we have modified to include precedence constraints.

1. INTRODUCTION

The basic task scheduling problem has been studied in many variations and in many domains, with applications in operations research, parallel processing, and robot routing. The last of these in particular often requires robots to travel to a specified location in order to complete a task; the resulting problems are typically NP-hard and therefore intractable in general. As a result, scheduling heuristics and approximation algorithms are used.

In robot routing, auction-based algorithms have risen to prominence, providing a strong decentralized approach to task scheduling. Auctions have the advantage of being naturally decentralized (offloading most of the computation into each robot's calculation of its bids), as well as being flexible to different team objectives and problem types.

Here we study the precedence-constrained multi-robot task allocation problem with homogeneous robots (henceforth PC-MRTA), which is NP-hard (it is shown in [16] that a special case of PC-MRTA is NP-hard). This problem falls under the XD[ST-SR-TA] category of the MRTA taxonomy given in [14], since precedence constraints impose cross-schedule dependencies, robots can execute at most one task at a time, tasks are assigned to single robots, and multiple tasks can be assigned to each robot (so the problem is time-extended). Unlike many other MRTA problems, this problem (in the general form, allowing arbitrary precedence constraints) has not been thoroughly studied, perhaps due in part to its complexity even for approximation.

In this paper, we develop an auction-based algorithm which gives a decentralized method of approximate solution for PC-MRTA and generalize this algorithm to make use of methods for task prioritization. We analyze this algorithm's complexity, prove its soundness and completeness, and demonstrate its performance empirically using modified versions of the Solomon [25] and Gehring-Homberger [5] data sets for the vehicle routing problem with time windows.

Appears in: *Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2016), J. Thangarajah, K. Tuyls, C. Jonker, S. Marsella (eds.), May 9–13, 2016, Singapore.*

2. RELATED WORK

Multiprocessor Scheduling

The problem of task scheduling with precedence constraints is well-studied within the parallel processing literature. Graham [7] gives a $(2-\frac{1}{m})$-approximation algorithm for scheduling under precedence constraints with the minimum-makespan objective. However, this problem is substantially simpler than ours, in which robots must travel between tasks.

Multiprocessor scheduling algorithms often make use of linear programming (LP) relaxations; in [9], a 7-approximation is given for precedence-constrained problems with release dates (i.e. earliest start times) using task completion times in the LP-relaxed schedule to order tasks for simple list scheduling. Chekuri et al. [1] improve this bound, and it has been shown that using LP midpoints to order the tasks gives a 4-approximation for the more general constraint of precedence delays [22]. This result in particular is enticing; precedence delays specify an amount of time that must elapse before each task can begin after its predecessors, which evokes a comparison with travel time. Unfortunately, precedence delays are not general enough to incorporate travel time, which varies based on a robot's prior location.

Another important aspect that is lacking (for our purposes) in these methods is a decentralized approach. List scheduling algorithms can often be adapted into an auction framework, but in general multiprocessor scheduling algorithms are not designed with this goal in mind. In MRTA, however, auction-based approaches are desirable due to their flexibility, decentralized nature, and robustness to failure.

Multi-robot Task Allocation and Scheduling

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 for the makespan [16]. The SSI auction has been shown both effective and computationally inexpensive compared to other auction schemes [13].

A more general approach using a mixed integer LP (MILP) model similar to ours was taken by [6], but this model was strictly centralized. In [10] a decentralized method is introduced that performs well, but for a problem variant that is not sufficiently general to extend to PC-MRTA.

Luo, Chakraborty, and Sycara [17] recently introduced an auction-based algorithm for multi-robot assignment of tasks for a particular special case of PC-MRTA, 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 between it and 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.

In [24], a distributed solution to a problem similar to ours is presented. 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, which does not require communication during task execution.

Distributed Constraint Optimization

Distributed Constraint Optimization Problem (DCOP) [19] models and algorithms offer an alternative framework for PC-MRTA. However, solving DCOP exactly is NP-hard and impractical even for unconstrained MRTA problems [11]. Thus, approximate methods such as Max-Sum [4] have been used for task allocation in sensor networks and in RoboCup Rescue [23].

Various other methods have been proposed for solving varieties of DCOP [23, 18, 3, 27]. However, despite the wealth of literature on DCOP models and solutions, we are unaware of any work that has extended DCOP to handle general precedence constraints.

3. PROBLEM DEFINITION

We are given $\mathcal{T}$, a set of n tasks with precedence constraints. The precedence constraints can be encoded with a precedence graph, i.e. a directed graph $G_{\mathcal{P}} = (\mathcal{T}, E)$, where if $t_1 \prec t_2$ (or equivalently, if $(t_1, t_2) \in E$), then the task t_1 is a predecessor of t_2 and must be completed before any robot begins executing t_2. It is straightforward to see that $G_{\mathcal{P}}$ must be acyclic for a valid schedule to exist. We allow any valid precedence constraints on the tasks; i.e. $G_{\mathcal{P}}$ may be any directed acyclic graph (DAG) over the tasks.

Additionally, we have $\mathcal{R}$, a set of m robots, each with an initial location. This initial location is allowed to differ from robot to robot, but $\mathcal{R}$ is otherwise homogeneous. Each task has a location associated with it that a robot must be at to execute the task.

A valid schedule for PC-MRTA consists of a partitioning of $\mathcal{T}$ across $\mathcal{R}$ (allocation) and a schedule for each robot which specifies the time at which each allocated task is executed. These robot schedules must collectively respect the precedence constraints on $\mathcal{T}$. The makespan of a schedule S is defined as the latest finish time of any task in S, i.e. the maximum robot finish time. Our goal is to minimize this value, which is similar to the MiniMax team objective in e.g. [16, 26, 28].

The mathematical model for this problem is described in Figure 1. The model adds a set of dummy tasks $\mathcal{T}_d$ at the robots' initial locations to represent their starting positions. The decision variables include the continuous and nonnegative start (S_{t_j}) and finish (F_{t_j}) time variables for each task, the makespan variable Z, the binary assignment variables $A^{r_i}_{t_j}$, $Y^{r_i}_{t_j}$, and the precedence ordering variables $X^{r_i}_{t_j,t_k}$. The variable $A^{r_i}_{t_j}$ is 1 if task t_j is allocated to robot r_i, $Y^{r_i}_{t_j}$ is 1 if t_j is the last task for r_i, and $X^{r_i}_{t_j,t_k}$ is 1 if

$$\text{minimize:} \quad Z \tag{2}$$
$$\text{subject to:} \quad Z \geq F_{t_j},\ \forall t_j \in \mathcal{T} \tag{3}$$
$$\sum_{r_i \in \mathcal{R}} A^{r_i}_{t_j} = 1,\ \forall t_j \in \mathcal{T} \cup \mathcal{T}_d \tag{4}$$
$$\sum_{t_j \in \mathcal{T}_d} A^{r_i}_{t_j} = 1,\ \forall r_i \in \mathcal{R} \tag{5}$$
$$\sum_{t_j \in \mathcal{T}} Y^{r_i}_{t_j} = 1,\ \forall r_i \in \mathcal{R} \tag{6}$$
$$\sum_{t_j \in \mathcal{T} \cup \mathcal{T}_d, t_j \neq t_k} X^{r_i}_{t_j,t_k} = A^{r_i}_{t_j},\ \forall t_k \in \mathcal{T}, r_i \in \mathcal{R} \tag{7}$$
$$\sum_{t_k \in \mathcal{T}, t_j \neq t_k} X^{r_i}_{t_j,t_k} + Y^{r_i}_{t_j} = A^{r_i}_{t_j},\ \forall t_j \in \mathcal{T} \cup \mathcal{T}_d,\ r_i \in \mathcal{R} \tag{8}$$
$$F_{t_j} + d(t_j, t_k) - M(1 - X^{r_i}_{t_j,t_k}) \leq S_{t_k}\ \forall t_j, t_k \in \mathcal{T},\ r_i \in \mathcal{R} \tag{9}$$
$$0 \leq S_{t_j} \leq \infty,\ \forall t_j \in \mathcal{T}_d \tag{10}$$
$$0 \leq F_{t_j} \leq \infty,\ \forall t_j \in \mathcal{T} \cup \mathcal{T}_d \tag{11}$$
$$F_{t_j} - S_{t_j} \geq \text{dur}(t_j), \quad \forall t_j \in \mathcal{T} \tag{12}$$
$$S_{t_k} - F_{t_j} \geq 0,\ \forall (t_j, t_k) \in \mathcal{T}_{prec} \tag{13}$$
$$A^{r_i}_{t_j} \in \{0,1\},\ \forall t_j \in \mathcal{T} \cup \mathcal{T}_d,\ r_i \in \mathcal{R} \tag{14}$$
$$Y^{r_i}_{t_j} \in \{0,1\},\ \forall t_j \in \mathcal{T},\ r_i \in \mathcal{R} \tag{15}$$
$$X^{r_i}_{t_j,t_k} \in \{0,1\},\ \forall t_j, t_k \in \mathcal{T} \cup \mathcal{T}_d,\ r_i \in \mathcal{R} \tag{16}$$

Figure 1: Mixed Integer Program for PC-MRTA

robot r_i performs task t_j followed by t_k. The model accounts for the time robots spend traveling between tasks, $d(t_j, t_k)$, and the duration of tasks, dur(t_j).

The model variable Z is a continuous positive variable that replaces the optimization objective in Equation 1. We linearize the optimization objective by making Z the upper bound for all finish times across all tasks, i.e.

$$Z = \max_{t_j \in \mathcal{T}} F_{t_j} \tag{1}$$

In this model, constraint (2) defines the makespan as the latest finish time of any task. Constraints (3-4) ensure that each task is assigned to exactly one robot, and that each robot has a dummy task; constraints (5-7) limit the number of final tasks for each robot to one and ensure that all tasks that are not scheduled first or last have predecessors or successors respectively. Constraints (8-11) are temporal constraints that enforce that robots have time to travel between and execute their tasks, and constraint (12) establishes a relationship between start times of dependent tasks ($\mathcal{T}_{prec}$), such that a task never starts before its predecessors. In constraint (8) M is a large value, which is part of the big-M formulation for mixed integer program constraints.

4. THE ITERATED AUCTION ALGORITHM

Our algorithm is in the form of an iterated auction, which repeatedly uses a modified version of the sequential single-item (SSI) auction to schedule tasks in batches. In particular, batches of tasks are selected such that the tasks are pairwise independent, so that they can be scheduled without regard for when the other tasks in the batch are scheduled. This task independence is the main assumption of the SSI auction. The batch selection and auctioning process is managed by an auctioneer, while robots serve as bidders.

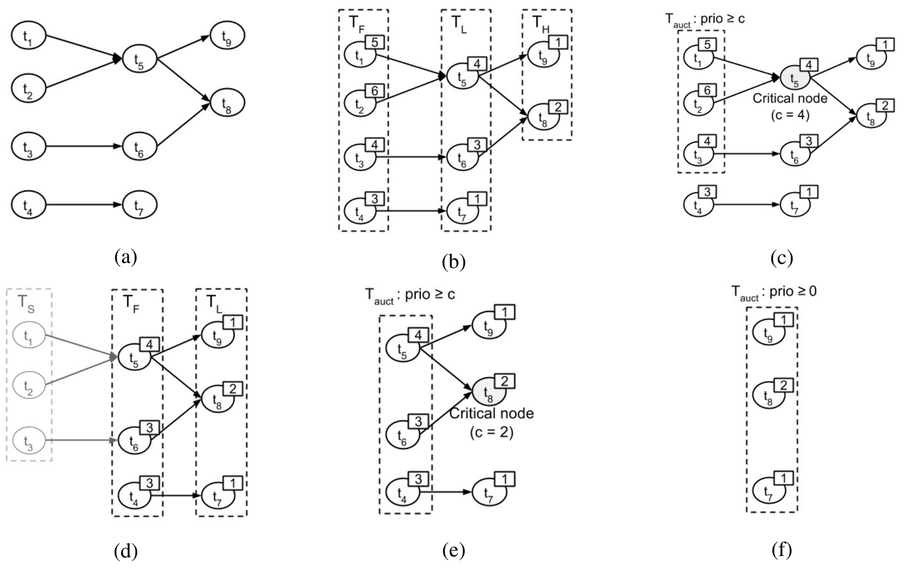


Figure 2: A demonstration of the iterated auction. The number in a square next to each node in (b)-(f) represents that task's priority. Priorities have been assigned arbitrarily to illustrate the batch selection process. (a) The initial task precedence graph. (b) The labeling of the precedence graph by layers. (c) The identification of the critical node, and batch selection for the first iteration of pIA. (d) The new labels for the precedence graph after the first iteration. (e) Batch selection for the second iteration. (f) The remaining tasks are scheduled in the final iteration.

Henceforth we will 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. In explaining the operation of the iterated auction algorithm, it is helpful to visualize the task precedence graph $G_{\mathcal{P}}$ as a DAG with nodes grouped into layers, as shown in Figure 2b. The first layer, T_F, is the set of tasks with no predecessors, i.e. $T_F = \text{free}(\mathcal{T})$. The second layer is then $T_L = \text{free}(\mathcal{T} \setminus T_F)$, and so on for successive layers. In our notation, T_H is the set of tasks which are not in $T_F \cup T_L$, i.e. the 'hidden' tasks which are not free or second-layer. We refer to T_H as 'hidden' tasks because in a given iteration the tasks in T_H do not affect the scheduling process.

It is important to note that no precedence constraints exist between tasks in any layer of $G_{\mathcal{P}}$. To see this, just consider that if a constraint did exist within a layer, then the child node would not be free, which contradicts the layer selection process (see Figure 2b). This allows us to use the modified SSI auction, since the tasks in each layer can be scheduled in any order with respect to each other without violating constraints.

4.1 The Simple Iterated Auction

We will refer to the simplest iterated auction algorithm as the Simple Iterated Auction (sIA). In sIA, the batches of tasks to be auctioned are the layers of $G_{\mathcal{P}}$. In each iteration, the auctioneer selects the top layer of tasks, T_F, and sends them to each robot; the robots then compute and send bids to the auctioneer, which selects the robot with the lowest bid and assigns the corresponding task to it (as in an SSI auction; see e.g. [15, 16] for a more detailed description). The pseudocode for sIA is given in Algorithms 1-4 if all tasks are taken to have priority zero. Alternatively, one obtains full pseudocode for sIA by removing line 5 from Algorithm 1 and replacing line 6 with $T_{auct} = T_F$ (so that the entire first layer is auctioned). Since the algorithm presented in the next section is more general than sIA in this sense, we defer the full description of the process to the following sections.

4.2 The Iterated Auction with Prioritization

The simplicity of sIA is appealing, and allows the iterated auction process to be easily visualized as peeling layers from the task precedence graph. By adding a precomputation step in which tasks are assigned a numerical priority (which we discuss later), we can guide the scheduling process. Put simply, in each iteration only the tasks in T_F that are high-priority with respect to the tasks in T_L are scheduled, as shown in Figure 2c. This allows less important first-layer tasks to be delayed so that important tasks not in T_F can be completed sooner. We call this the Iterated Auction with Prioritization (pIA).

The pseudocode for the auctioneer's role in pIA is given in Algorithm 1. In lines 1-2, tasks are partitioned into T_S, T_F, T_L, and T_H based on whether they have been scheduled, are free, are second-layer, or are 'hidden'; this is shown in Figure 2b. In line 3, the arrays F and PC are initialized; F stores the latest finish time of tasks that have been scheduled, while PC stores the earliest valid start time for tasks whose preconditions have been scheduled. Note that the tasks initially in T_F can be started at any time, so the values of zero in PC are appropriate for these tasks.

In each iteration (starting in line 4), the auctioneer finds the 'critical value' c, which is the maximum priority of any second-layer task (shown in line 5 of Algorithm 1; in this paper, we use the convention that $\max_{\emptyset}(x) \equiv 0$ for all x). We will occasionally refer to a task in T_L with priority c as a critical node in the context of the task precedence graph. In line 6, the first-layer tasks with priority at least as high as the critical value (see Figure 2c) are then selected as T_{auct}. The tasks in T_{auct} are then auctioned in line 7 using the same modified SSI auction as in sIA. The robots' algorithm for this auction is shown in Algorithm 3.

In the modified SSI auction, the auctioneer sends T_{auct} to each robot along with the earliest start times for the tasks in PC. In parallel, the robots each find the task in T_{auct} for which their bid is lowest and submit this (task, bid) pair to the auctioneer (lines 2-12 of Algorithm 3). The auctioneer then sends the (robot, task) pair

Algorithm 1 Auctioneer's algorithm for pIA

```
Input: Precedence graph $G_\mathcal{P} = (\mathcal{T}, E)$, robots $\mathcal{R}$
 1: $T_S = \emptyset, T_F = \text{free}(\mathcal{T})$
 2: $T_L = \text{free}(\mathcal{T} \setminus T_F), T_H = \mathcal{T} \setminus (T_F \cup T_L)$
 3: Initialize arrays $F$, $PC$ to zero
 4: while $|T_S| < n$ do
 5:     $c = \max_{t \in T_L}(\text{prio}(t))$
 6:     $T_{auct} = \{t \in T_F :\ \text{prio}(t) \geq c\}$
 7:     ModifiedSSI($T_{auct}$, $PC$)
 8:     ReceiveFinishTimes($\mathcal{R}$,$F$)
 9:     for all $t \in T_{auct}$ do
10:         UpdatePrecGraph($t$, $PC$, $G_\mathcal{P}$)
11:     end for
12: end while
13: return $F$
```

Algorithm 2 UpdatePrecGraph

```
Input: Task $t$, array $PC$, precedence graph $G_\mathcal{P}$
 1: Move $t : T_F \rightarrow T_S$
 2: for all $t' \in T_L \cap$ children($t$) do
 3:     if parents($t'$) $\subset T_S$ then
 4:         Move $t' : T_L \rightarrow T_F$
 5:         $PC[t'] = \max_{t'' \in \text{parents}(t')}(F[t''])$
 6:         for all $t'' \in T_H \cap$ children($t'$) do
 7:             if parents($t''$) $\subset T_S \cup T_F$ then
 8:                 Move $t'' : T_H \rightarrow T_L$
 9:             end if
10:         end for
11:     end if
12: end for
```

Algorithm 3 Robot r's algorithm for each auction

```
Input: Set of tasks $T_{auct}$, earliest start times $PC$
Output: Vector of finishing times $F$
 1: while $|T_{auct}| > 0$ do
 2:     $b = \infty$
 3:     $t_{bid} = \varnothing$
 4:     for all $t \in T_{auct}$ do
 5:         for all positions $p$ in current schedule $S$ do
 6:             Insert $t$ in $S$ at $p$ to get $S'$
 7:             if isValid($S'$) and bid($S'$) $< b$ then
 8:                 $b$ = bid($S'$), $t_{bid} = t$
 9:             end if
10:         end for
11:     end for
12:     sendBid($b, t_{bid}$)
13:     $r_{win}, t_{win}$ = receiveWinner()
14:     if $r = r_{win}$ then
15:         Insert $t_{win}$ in $S$ at best valid position
16:     end if
17:     $T_{auct} = T_{auct} \setminus t_{win}$
18: end while
19: TightenSchedule($F$)
20: return $F$
```

Algorithm 4 TightenSchedule

```
Input: Vector of finishing times $F$,
       Vector of robot's internal constraints on latest finish times, $LF$
 1: for all tasks $t$ in current schedule $S$ do
 2:     Compute earliest finish time $f$ for task $t$
 3:     Set task finish time to return to auctioneer: $F[t] = f$
 4:     Introduce temporal constraint on STN: $LF[t] = f$
 5: end for
```

with the lowest bid to each robot, shown in line 13 of Algorithm 3. In lines 14-16, the winning robot updates its schedule to include the new task. The auction continues in this way until all of T_{auct} has been scheduled. In line 19 of Algorithm 3, each robot calls 'TightenSchedule' (Algorithm 4), which updates the finish times in F and imposes constraints on the tasks the robot has scheduled so that they cannot be delayed. This is done to ensure precedence constraints are satisfied, and is discussed in more detail below.

After the modified SSI auction finishes, the auctioneer collects the finish times for T_{auct} and updates F, shown in line 8 of Algorithm 1. The last step in each iteration is the call to 'UpdatePrecGraph', in which the auctioneer maintains the labeling of tasks as e.g. second-layer. The value $PC[t]$ for each task t that has been freed in the current iteration is also set during this step to the latest finish time of any parent of t. This allows robots to enforce that a task is not begun before its predecessors finish.

4.3 Robot Scheduling and Bidding

We have until now glossed over the mechanisms by which robots maintain their schedules and compute bids, mainly because it is more complicated than the rest of the discussion. In the basic SSI auction, the robots simply receive tasks from the auctioneer, compute the cheapest insertion of each task into their existing schedules (this is called the insertion heuristic), and place a bid (usually either the cost of insertion or the makespan after insertion). The winning robot then inserts the corresponding task into its schedule.

Since our goal is to minimize the makespan, our robots bid the makespan of the new schedule (which prevents overencumbered robots from being assigned more tasks if other robots can help). Note that a robot's bid is a function only of its current schedule and the additional task under consideration; robots are not aware of task prioritization. In the iterated auction scheme, we must introduce additional complexity to ensure that precedence constraints are not violated. This yields the modified SSI auction described above, written in Algorithm 1 as 'ModifiedSSI(T_{auct}, PC)'. The auctioneer's role in this auction is essentially the same as in an SSI auction. However, instead of inserting tasks arbitrarily into their schedules, robots instead check for the validity of schedules, and 'tighten' their schedule between iterations. Pseudocode for the robots' scheduling process is given in Algorithm 3, and for 'TightenSchedule' in Algorithm 4. We will discuss the 'TightenSchedule' procedure shortly.

In a robot's schedule, to represent timing constraints and check whether a task insertion is valid, we use a Simple Temporal Network [2]. The STN is a graph-like data structure in which events (task start and finish times in our case) are nodes. Timing constraints are represented as edges; an edge between nodes constrains their relative time of occurrence (e.g. an edge between task start and task finish enforcing that the duration of the task elapses between them). Constraints can also be added with respect to absolute time, so that events occur in a specific time window.

In 'TightenSchedule' (Algorithm 4), the robot adds constraints so that its currently scheduled tasks cannot be delayed by tasks in a later iteration. This ensures that precedence constraints are not violated, while maintaining the robots' privacy, i.e. robots are only aware of their tasks and the timing constraints on those tasks. The STN also allows us to check that a schedule is feasible, or can be completed by the robot without violating any constraints; this is done via 'isValid' in Algorithm 3, which utilizes the tasks' earliest start times in PC. See [2] for a more detailed description of managing temporal constraints using STNs, and [21] for an example of their application in a similar setting.

The similarity of 'ModifiedSSI' to the SSI auction is possible due to the lack of precedence constraints between tasks in T_{auct}. This allows us to, despite the constraints in the overall problem, approach each iteration with a independent-task scheduling method, and gain some of the benefit of the insertion heuristic (as discussed in [25, 16]) in each iteration based on how large T_{auct} is. In particular, the finish times in F are not set until the end of the iteration, since these are malleable until the end of the iteration. This allows tasks to be inserted into schedules in the most efficient manner possible. When 'TightenSchedule' is called additional constraints are added to each robot's internal STN to ensure that no currently scheduled task can be delayed.

5. PRIORITY ASSIGNMENT

The question of how to prioritize tasks is critical to the performance of pIA; with the flexibility of controlling the order in which tasks are scheduled comes the potential to either worsen or improve the resulting schedule. In this paper, we use a simple heuristic to assign priorities 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

$$L(t) = \text{dur}(t) + \max_{t' \in \text{children}(t)} (L(t')) ,$$

$$U(t) = \text{dur}(t) + \max_{t' \in \text{children}(t)} (d(t,t') + U(t')) ,$$

where we use the convention that $\max_{\emptyset}(x) \equiv 0$, so that $L(t) = U(t) = \text{dur}(t)$ if the task t has no children. Then we let

$$\text{prio}_{\alpha}(t) = (1-\alpha)L(t) + \alpha U(t) , \; 0 \leq \alpha \leq 1.$$

In this way our priority assignment is a heuristic for the total length of the precedence graph, or the total time to completion for the maximum-duration chain of tasks rooted at the given task. The value $U(t)$ represents the total time this chain would take to be executed by a single robot (in the absence of other constraints), while $L(t)$ is the least possible time required to execute these tasks (since it is the sum of their durations, and the tasks are constrained and must be completed sequentially).

The parameter α can be thought of roughly as the proportion of travel time that is accounted for in task priorities. If α is high, then the priority of a task t is approximately $U(t)$, or the longest path to completion including travel time. Similarly, $\alpha \approx 0$ yields $\text{prio}_{\alpha}(t) \approx L(t)$, which is the longest path to completion including only the duration of the tasks. Since the task nodes along the longest path to completion may change as α changes, our intuition for α as the proportion of travel time accounted for should be taken more as a rough intuition and not as a definition.

Formulating the priorities in this way allows the possibility of adaptively choosing α for each task based on the structure of the graph or other factors, or changing the value of α during the iterated auction based on robot locations, etc. Though we do not use adaptive choices for α in this paper, it is possible that practical use cases for our algorithm will find this flexibility useful. We will use α to refer specifically to our method for priority assignment throughout this paper. In most cases, intuiting it as a proportion of travel time accounted for is appropriate.

Note that we can compute these priorities in time linear in the size of the precedence graph $G_{\mathcal{P}}$ by using dynamic programming. This can be achieved by computing priorities bottom-up, beginning with the tasks that have no children. Thus each task node is iterated over once when its priority is computed and each edge is iterated over once when a parent node checks the priority of each of its children.

This system of prioritization is inspired by critical path scheduling methods, introduced in [12]. These methods identify the longest task chain in a DAG and schedule tasks so as to minimize delays on that chain. Adapting this critical path approach to the PC-MRTA problem is difficult, however. We use the priority system above to account for the ability of other robots to continue a task chain and thereby lessen the cost of travel time.

Our method of prioritization is a somewhat naive heuristic, since it does not always correctly account for travel time and fails to consider the location of future tasks; for example, a robot may ignore a conveniently located, easy-to-complete task in favor of higher-priority tasks, and thereby worsen its schedule when it must return to complete this task. Heuristics which explicitly consider task location (and not just the length of a task chain) may be more successful.

As an open question, we conjecture that a non-heuristic prioritization scheme based on an LP relaxation may be very successful in pIA. It was shown in [22] that using task midpoints from a schedule produced via LP relaxation to order tasks for a list scheduling algorithm gives a 4-approximation for the general problem of multiprocessor scheduling with precedence delays; we suspect that a similar approach for prioritization could outperform many heuristic methods in the pIA scheme for PC-MRTA.

6. ANALYSIS

In this section, we provide arguments for the complexity, soundness, and completeness of the iterated auction.

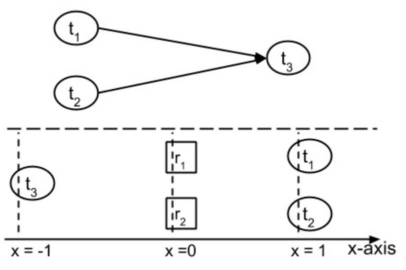
6.1 Complexity Analysis

Here we provide a sketch for the complexity of the complete iterated auction procedure. We will assume the average case, in which tasks are assigned in roughly equal number to the different robots. This assumption seems strong, but is actually well-founded; a robot's bid is the total time taken to complete its current schedule, so robots with more tasks will tend to make higher bids and be assigned fewer tasks.

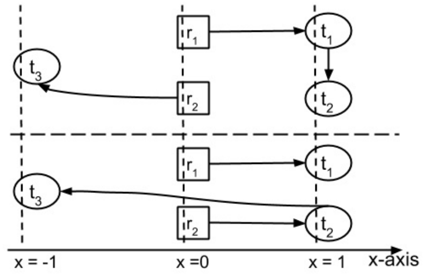
First consider the 'ModifiedSSI' procedure called in line 11 of Algorithm 1. Let $n_{auct} = |T_{auct}|$ be the number of tasks to be auctioned. The auctioneer sends T_{auct} and PC to each robot for m total messages of size $\mathcal{O}(n_{auct})$. Let n_{pre} be the total number of tasks scheduled in previous iterations. Then in iteration i of Algorithm 3's while loop, any given robot will have on average $\frac{n_{pre}+i}{m}$ positions for insertion of a task in its current schedule. On average there will therefore be $\mathcal{O}((n_{auct}-i)(\frac{n_{pre}+i}{m}))$ attempted insertions in any robot's schedule (line 6 of Algorithm 3). Validity checking using the robot's STN has average time complexity of $\mathcal{O}((\frac{n_{pre}+i}{m})^3)$ [2], and thus the total time taken in a scheduling iteration is $\mathcal{O}((n_{auct}-i)(\frac{n_{pre}+i}{m})^4)$, along with $2m$ messages (one sent and one received by each robot) of size $\mathcal{O}(1)$. Since this procedure is repeated n_{auct} times, this gives us a total complexity

$$\sum_{i=1}^{n_{auct}} (n_{auct}-i)(\frac{n_{pre}+i}{m})^4 = \mathcal{O}(\frac{n_{auct}^2}{m^4}(n_{auct}^4 + n_{pre}^4)) .$$

Note that this complexity for the 'ModifiedSSI' procedure is (typically) asymptotically dominant in each iteration; the only other consideration is the n_{auct} calls to 'UpdatePrecGraph', which will on average be relatively inexpensive. To see this, observe that the total work done during the entire iterated auction by 'UpdatePrecGraph' is to move all n tasks into T_S, layer by layer. This requires



(a)



(b)

Figure 3: A simple worst-case example demonstrating the vulnerability of the iterated auction to certain mistakes. In (a), the example is defined by three tasks with the precedence graph shown on the top of the figure, and with task locations and initial robot locations shown below. In (b), the optimal allocation is shown on top, while the iterated auction allocation (for any prioritization) is shown on the bottom.

at most three movements per task (from T_H to T_L, then to T_F, then to T_S), and thus the total work done by 'UpdatePrecGraph' is $\mathcal{O}(n)$. The complexity of the modified SSI auction is therefore asymptotically dominant.

Now consider the complexity of the overall algorithm. Roughly, we have two extremes - either for each iteration $n_{auct} = \mathcal{O}(1)$, or $n_{auct} = \Omega(n)$. In the former case, the above complexity for each iteration evaluates to $\mathcal{O}((\frac{i}{m})^4)$ for iteration i, which gives us a total complexity of

$$\sum_{i=1}^{\mathcal{O}(n)} (\frac{\mathcal{O}(i)}{m})^4 = \mathcal{O}(\frac{n^5}{m^4}) .$$

In the latter case, we have $\mathcal{O}(1)$ iterations, with total complexity $\mathcal{O}(\frac{n^6}{m^4})$. Thus if we assume that tasks are evenly distributed among robots, the iterated auction algorithm has complexity

$$\mathcal{O}(\frac{n^5}{m^4}) \leq T(n,m) \leq \mathcal{O}(\frac{n^6}{m^4}) .$$

It is interesting to note that this result implies that pIA is more efficient than sIA, since prioritization reduces the average value of n_{auct}. Note also that this is often a fairly loose upper bound, since in particular many of the checks made by robots for their schedules' validity will be resolved more quickly than the asymptotic bound.

6.2 Soundness and Completeness

We will now show that the iterated auction algorithm will always produce a valid schedule if one exists. 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 iterated auction algorithm produces a valid schedule in this case. We will prove this for pIA, since sIA is equivalent to pIA for a given priority assignment. 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 pIA will successfully schedule all the tasks and terminate. The relevant pseudocode is the while loop beginning on line 4 of Algorithm 1. We will argue that in every iteration $|T_{auct}| > 0$, and therefore that the number of tasks scheduled, $|T_S|$, increases in every iteration until all tasks are scheduled.

To see this, first suppose for a given iteration that T_L, the set of second-layer tasks, is nonempty. Then the critical value c will be set to the priority of a task in T_L, and by definition of T_L that task must have a parent in the set of free tasks T_F. Since the priority of a task is at least that of its successors, this task in T_F has priority at least c and will be included in T_{auct}.

Now assume instead that T_L is empty. Then c will be set to zero. 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. Consider tasks t_1, t_2 with $t_1 \prec t_2$. Note that the tasks scheduled in any given iteration are pairwise independent, so since $t_1 \prec t_2$, t_1 must have been scheduled in an earlier iteration than t_2.

When t_1 was scheduled, the robot to which it was assigned returned t_1's finish time in F (Algorithm 1, line 8 and Algorithm 3, line 20). This followed the robot's call to 'TightenSchedule', which set $F[t_1]$ as the latest valid finish time for t_1. When t_2 was moved into T_F by 'UpdatePrecGraph' (Algorithm 2) after all of its predecessors were scheduled, $PC[t_2]$ was set to the maximum completion time of any of its predecessors; since $t_1 \prec t_2$, we therefore know that $PC[t_2]$ is at least the finish time of t_1. But then when t_2 is auctioned, $PC[t_2]$ is passed to each robot as the earliest start time for t_2 (Algorithm 1, line 7). The robot that wins t_2 in the auction adds it to its schedule in line 15 of Algorithm 3; in particular, t_2 is inserted into the robot's schedule in the best *valid* position, i.e. the position which respects all of the earliest start times in PC and yields the cheapest schedule of all such insertion positions. Validity is checked using robots' STNs, the data structure in which timing constraints are managed (discussed above). Since t_2 is inserted into only valid positions, and no other task insertion is allowed if it results in t_2 beginning before $PC[t_2]$, t_2 will start no earlier than $PC[t_2]$. Since the finish time of t_1 is at most $F[t_1] \leq PC[t_2]$, this implies that the precedence constraint $t_1 \prec t_2$ is satisfied.

6.3 Weaknesses of the Iterated Auction

In Figure 3, an example problem is shown that the iterated auction performs poorly for. Refer to Figure 3b. If each task has duration δ, the optimal allocation sends the robots in different directions, so task t_1 is completed at time $1 + \delta$, task t_2 at time $1 + 2\delta$, and task t_3 at time $1+3\delta$. However, the iterated auction attempts to free t_3 for scheduling as soon as possible, resulting in both robots being sent to $x = 1$. In this case, both $t1$ and $t2$ are completed at time $1 + \delta$, but t_3 is not completed until time $3 + 2\delta$.

This example exploits the primary weakness of the iterated auction. Since pIA does not explicitly consider future task locations, robots may be mispositioned with respect to future iterations if task locations oscillate between extremes from iteration to iteration. In such cases, the entire robot team may be enlisted to help in each location in consecutive iterations, depending on the lengths of their partial schedules. This can result in relatively poor performance with respect to the optimal schedule.

To improve performance in such cases, a modified bidding scheme could be used in which robots place bids that are based in part on distance traveled. This would result in an optimal solution of the example in Figure 3: when the robots are bidding for task t_2 in the first iteration, the first robot's bid (using our bidding scheme) is its new makespan, $1 + 2\delta$, while the other robot bids $1 + \delta$. If the fact that the latter robot would travel an entire unit is taken into account, however, its bid would be higher, and the first robot would complete both t_1 and t_2. See e.g. [21] for more details on such a bidding scheme.

7. EXPERIMENTS

Our experiments were conducted via simulation, using an implementation of the pIA algorithm in C++. We utilized existing data sets from the vehicle routing problem (VRP) literature [25, 5], using the time windows for task completion in these problems to loosely order generation of precedence constraints. This was necessary due to the lack of current research on PC-MRTA; we were unable to find more relevant problem instances.

Each of the above data sets is split into six parts: C1, C2, R1, R2, RC1, and RC2. The C instances have tasks which are spatially clustered, while the R instances have tasks which are randomly distributed. The RC instances are a mixture of R and C, with some tasks clustered and others randomly placed. Each Solomon instance has 100 tasks, while each Gehring-Homberger (G-H) instance has 1000 tasks.

Since these data sets are for research on the VRP with time windows, we must modify them in order to use them for PC-MRTA. To this end, we use the method of [20] to generate a precedence graph uniformly at random on the tasks in each instance. In this we respect the time windows of the original data; if a task's time window closes before another's in the original data, we do not allow the latter task to precede the former in our precedence structure.

In addition to imposing a loose ordering on the precedence graph, we also restrict the density of the graph. Without density limits, a random DAG will have on average $\frac{n^2}{4}$ edges [20], which is overconstrained for a scheduling problem: a Solomon instance with 100 tasks would have roughly 2500 constraints. To remedy this, we generate for each original data instance eight random precedence graphs, four of which are sparse (limited to at most $\frac{n}{2}$ edges) and four of which are denser (limited to at most $2n$ edges). In all cases, we average across all four of the random graphs for each instance and level of density to reduce the effects of chance.

For the clustered (C) and partially clustered (RC) data sets, we reasoned that random constraints on a clustered set of tasks would de-emphasize the clustered nature of the tasks. We therefore imposed additional restrictions on the precedence structure in these sets. The generation process selects random pairs of nodes and inserts an edge between them if it is valid (or removes the edge if it already exists); for the C and RC data instances, we randomly decline edges that pass between clusters with a probability of 0.8. This yields a precedence graph in which the majority of constraints respect the clustered structure of the original data, and task clusters are sparsely connected by constraints.

7.1 Results and Discussion

Our results were gathered by running our implementation of pIA on each of our data sets to get the makespan of the iterated auction schedule on each instance, using ten robots for the sparse Solomon data and five for the dense Solomon data, with ten times this number for the G-H data. We gathered results for sIA and for pIA with $\alpha \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$. For comparison, we also used a simple greedy algorithm that selects a task at random from T_F and allocates it to the robot that can finish it soonest. As mentioned above, each instance's result (e.g. Solomon instance C101 with sparse precedence constraints) is averaged across four random graphs.

Additionally, we used Gurobi [8] with the model in Figure 1 to compute optimal results for three small instances based on Solomon data. For all three, pIA ($\alpha = 0.5$) performed within about 10% of the optimal, and sIA somewhat worse. The paths for a 16 task, 4 robot problem are shown in Figure 4. For this problem, the optimal makespan was 109.4, while the pIA and sIA makespans were 121 and 149 respectively.

Our results are shown in Table 1. Each entry is averaged over approximately 80 trials: there are about 20 instances in the Solomon and G-H data sets for each problem type, and we generate four random precedence graphs for each instance. A distributed implementation of each algorithm runs in ≈ 0.25 seconds on the Solomon problems and less than 10 seconds on the 1000-task G-H problems.

First, we see that the iterated auctions significantly outperform the greedy auction in almost every case. For the Solomon data, sIA produces schedules that beat the greedy schedule by about 20% or more for every type of problem and level of sparsity, with pIA performing slightly worse than sIA in most cases.

For the G-H data, the results are less uniform; on the dense precedence graphs, either sIA or pIA achieve about a 20% improvement on the greedy schedule. However, on the sparse graphs, this improvement is less substantial, with sIA achieving a 12% improvement on the R data, about a 5% improvement on the RC data, and actually performing slightly worse than the greedy algorithm on the C data. In each of these cases, there is a value of α such that pIA is about 12% better than the greedy algorithm, however ($\alpha = 0.1$ for the C and RC data, and $\alpha = 0.9$ for the R data).

We can see that in addition to outperforming the greedy algorithm, sIA also performs as well as or slightly better than pIA in most cases. The only cases in which pIA offers significant improvement over sIA are on the C and RC sets from the G-H data. This is due to two separate effects: the increasing effectiveness of prioritization for larger numbers of tasks, and the improving performance of our priority assignment scheme for more clustered data. We will discuss the former effect first.

With any reasonable priority assignment scheme, we expect pIA to become more effective on larger problems. This is because sIA puts the maximum number of tasks into each auction, making extensive use of the insertion heuristic to order the tasks and exploit pathing synergies between them. Prioritization attempts to ensure that high-priority tasks are completed sooner, but sacrifices this full-sized auction and reduces use of the insertion heuristic. For the 1000-task G-H problems, there are still enough tasks in the auction to benefit from the insertion heuristic, and using the task information encoded in the priorities becomes relatively more valuable.

Second, we see that our method of priority assignment tends to perform much better for clustered data. For the Solomon problems, pIA performs marginally better than sIA only on the C instances, while on the G-H problems, pIA performs much better than sIA on the C instances and somewhat better on the RC instances. This reflects on our particular priority assignment scheme; on the R data, priorities are assigned based on the random positions of the tasks,

Data Set	Problem Type	Sparsity		Greedy	sIA	pIA				
						$\alpha = 0.1$	$\alpha = 0.3$	$\alpha = 0.5$	$\alpha = 0.7$	$\alpha = 0.9$
Solomon	R	Sparse	μ	376.80	**295.23**	306.66	303.39	302.34	302.38	301.35
			σ	9.72	6.82	8.24	6.36	7.33	5.50	6.89
		Dense	μ	812.47	**645.85**	666.70	655.49	652.45	652.77	651.68
			σ	13.89	12.01	10.49	13.59	11.40	13.78	15.37
	C	Sparse	μ	440.12	349.76	341.84	**339.24**	345.96	344.09	342.79
			σ	12.80	8.41	8.67	9.02	13.88	10.34	14.26
		Dense	μ	946.56	743.04	741.84	**739.31**	739.79	743.91	749.10
			σ	21.57	19.13	25.54	21.03	26.96	23.36	20.87
	RC	Sparse	μ	455.44	**346.70**	358.06	355.80	356.72	355.16	358.23
			σ	9.49	8.51	6.07	8.02	11.24	9.12	9.24
		Dense	μ	1001.6	**770.19**	798.95	789.41	791.75	786.75	792.64
			σ	17.17	23.16	17.06	20.95	16.75	14.08	16.83
G-H	R	Sparse	μ	868.13	**761.15**	784.51	782.10	776.05	771.19	765.85
			σ	14.68	10.01	32.31	39.38	27.90	29.39	22.66
		Dense	μ	1777.0	**1407.6**	1463.6	1468.2	1473.2	1490.7	1488.7
			σ	17.99	26.84	24.55	31.49	28.17	27.70	26.02
	C	Sparse	μ	763.04	789.66	**674.84**	688.01	686.43	687.31	689.08
			σ	22.83	19.91	41.95	40.13	37.06	44.44	46.30
		Dense	μ	1498.5	1305.9	**1208.1**	1236.7	1237.7	1252.5	1239.3
			σ	54.85	31.24	43.17	46.74	40.12	38.92	39.70
	RC	Sparse	μ	828.24	788.23	**722.53**	726.16	726.09	729.70	725.48
			σ	11.69	13.67	36.27	37.48	34.92	38.58	38.53
		Dense	μ	1628.1	1354.8	**1297.4**	1311.3	1315.9	1333.7	1332.8
			σ	35.87	22.54	40.96	45.84	38.62	49.55	45.50

Table 1: Makespan results for sparse and dense precedence graphs comparing the pIA and sIA auctions, and a greedy method.

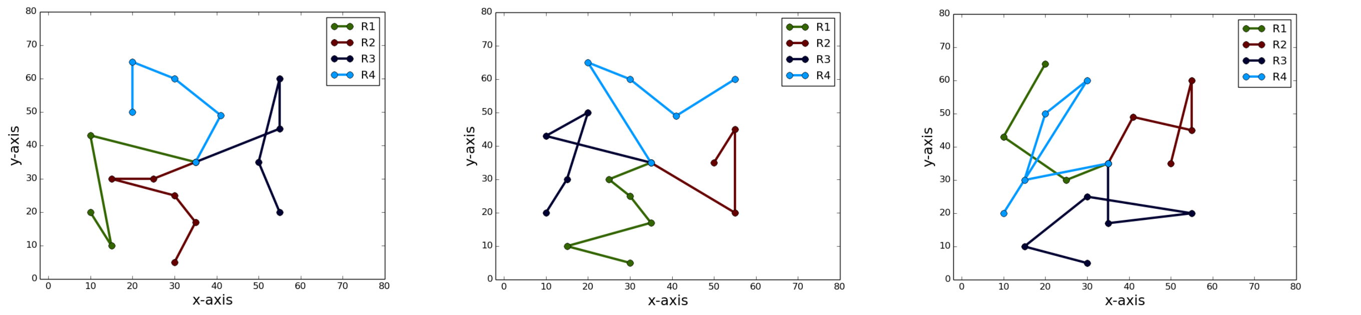


Figure 4: Routes produced by the optimal (left; makespan 109.4), pIA (middle; makespan 121) and sIA (right; makespan 149) methods for a Solomon instance with 16 tasks and four robots.

and may not accurately reflect the proper ordering of the tasks, as discussed above. For clustered data, low priority tasks that are available for scheduling are likely to be located in clusters that have lower average priority; since the task is free, it has no parents within the cluster and is therefore topologically highest, limiting the priority of its children. Due to the higher density of constraints within clusters, this provides a strong indication that the entire cluster is of lower priority. This shows that heuristic priority assignments can exploit synergies within data sets, which is desirable; in general we expect precedence-constrained data to be nonrandom, and to have some type of relationship between task constraints and locations. We expect that in specific applications, prioritization methods may be tailored to exploit the relationships between task locations and constraints for further improvement.

8. CONCLUSIONS AND FUTURE WORK

We introduced the iterated auction scheme for PC-MRTA and demonstrated substantial improvements over simple greedy scheduling. Our results indicate that at least for some problem types, priority assignment can improve on the performance of sIA. In particular, pIA performs well for large problem sizes with clustered or semi-clustered task groups. We also found that pIA performed within 10% of optimal for several small problems.

Since we used simple methods of prioritization, we expect that more sophisticated methods might yield further improvements. As mentioned previously, in practice precedence constraints are often related somehow to task location, so for a specific application certain priority schemes might provide strong performance by exploiting the specific structure of the problem. For example, if constrained tasks are always east of their parent tasks, a priority scheme that assigns more priority to western tasks may be successful by preventing robots from backtracking.

As we discussed in the section on priority assignment, we believe that using priorities derived from an LP relaxation as in [22] may provide excellent performance for a wide variety of problem types. This method of prioritization may work well in more general settings, when relationships between task locations and precedence constraints are not well understood. It may be also be that low-priority tasks are easier to identify than high-priority tasks; another approach to approximating PC-MRTA might use the sIA method and prune low-priority tasks in each iteration.

REFERENCES

[1] C. Chekuri, R. Motwani, B. Natarajan, and C. Stein. Approximation techniques for average completion time scheduling. *SIAM Journal on Computing*, 31(1):146–166, 2001.

[2] R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49, 1991.

[3] A. Farinelli, L. Iocchi, D. Nardi, and V. Ziparo. Assignment of dynamically perceived tasks by token passing in multirobot systems. *Proceedings of the IEEE*, 94(7):1271 –1288, 2006.

[4] A. Farinelli, A. Rogers, A. Petcu, and N. R. Jennings. Decentralised coordination of low-power embedded devices using the Max-Sum algorithm. In *Int'l Conf. on Autonomous Agents and Multi-Agent Systems*, pages 639–646, 2008.

[5] H. Gehring and J. Homberger. A parallel hybrid evolutionary metaheuristic for the vehicle routing problem with time windows. In *Proceedings of EUROGEN99*, 1999.

[6] M. Gombolay, R. Wilcox, and J. Shah. Fast scheduling of multi-robot teams with temporospatial constraints. In *Robotics: Science and Systems (RSS)*, 2013.

[7] R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17(2), 1969.

[8] I. Gurobi Optimization. Gurobi optimizer reference manual, 2014.

[9] L. A. Hall, A. S. Schulz, D. B. Shmoys, and J. Wein. Scheduling to minimize average completion time: Offline and online approximation algorithms. *Mathematics of Operations Research*, 22(3), 1997.

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

[11] 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*, pages 599–606, 2008.

[12] J. E. Kelley Jr. and M. R. Walker. Critical-path planning and scheduling. In *Proceedings of the Eastern Joint Computer Conference*, 1959.

[13] S. Koenig, C. Tovey, M. Lagoudakis, V. Markakis, D. Kempe, P. Keskinocak, A. Kleywegt, A. Meyerson, and S. Jain. The power of sequential single-item auctions for agent coordination. In *Proc. AAAI Conf. on Artificial Intelligence*, 2006.

[14] G. A. Korsah, A. Stentz, and M. B. Dias. A comprehensive taxonomy for multi-robot task allocation. *The International Journal of Robotics Research*, 32(12):1495–1512, 2013.

[15] 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.

[16] M. G. Lagoudakis, E. Markakis, D. Kempe, P. Keskinocak, A. Kleywegt, S. Koenig, C. Tovey, A. Meyerson, and S. Jain. Auction-based multi-robot routing. In *Robotics: Science and Systems (RSS)*, 2005.

[17] L. Luo, N. Chakraborty, and K. Sycara. Multi-robot assignment algorithm for tasks with set precedence constraints. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, 2011.

[18] K. S. Macarthur, R. Stranders, S. D. Ramchurn, and N. R. Jennings. A distributed anytime algorithm for dynamic task allocation in multi-agent systems. In *Proc. AAAI Conf. on Artificial Intelligence*, pages 701–706, 2011.

[19] 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*, pages 310–317, 2004.

[20] G. Melançon, I. Dutour, and M. Bousquet-Mélou. Random generation of directed acyclic graphs. *Electronic Notes in Discrete Mathematics*, 10, 2001.

[21] E. Nunes and M. Gini. Multi-robot auctions for allocation of tasks with temporal constraints. In *Proc. AAAI Conf. on Artificial Intelligence*, 2015.

[22] M. Queyranne and A. S. Schulz. Approximation bounds for a general class of precedence constrained parallel machine scheduling problems. *SIAM Journal on Computing*, 35(5), 2006.

[23] S. Ramchurn, A. Farinelli, K. Macarthur, M. Polukarov, and N. Jennings. Decentralised coordination in RoboCup Rescue. *The Computer Journal*, 53(9):1–15, 2010.

[24] 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*, pages 2314–2319, Oct 2007.

[25] M. M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35(2), 1987.

[26] C. Tovey, M. Lagoudakis, S. Jain, and S. Koenig. The generation of bidding rules for auction-based robot coordination. In L. Parker, F. Schneider, and A. Schultz, editors, *Multi-Robot Systems. From Swarms to Intelligent Automata Volume III*, pages 3–14. Springer Netherlands, 2005.

[27] Y. Xu, P. Scerri, B. Yu, S. Okamoto, M. Lewis, and K. Sycara. An integrated token-based algorithm for scalable coordination. In *Int'l Conf. on Autonomous Agents and Multi-Agent Systems*, pages 407–414, 2005.

[28] X. Zheng, S. Koenig, and C. Tovey. Improving sequential single-item auctions. In *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2006.