Proc. of IEEE International Conference on Robotics and Automation, Karlsruhe, Germany, 2013.

Multi-Robot Task Scheduling

Yu Zhang and Lynne E. Parker

***Abstract*— The scheduling problem has been studied extensively in the literature. Many algorithms have been developed to operate with different types of processors and tasks. In the robotics domain, when considering each robot as a processor, some of these algorithms can be directly adapted. However, most of the existing algorithms can only handle single-robot tasks, or multi-robot tasks that can be divided into single-robot tasks. As the task requirements may only be partially known, and the available (heterogeneous) robots can dynamically change, robots may be required to cooperate tightly to share different capabilities (i.e., sensors and motors). In such cases, considering scheduling for individual robots is no longer sufficient, since the robots need to work at the *coalition* level. Although there exist a few algorithms that also support these more complex cases, they do not represent efficient solutions that can be adapted by various multi-robot systems in a convenient manner. In this paper, we propose heuristics to address the multi-robot task scheduling problem at the *coalition* level, which hides the details of robot specifications, thus allowing these heuristics to be incorporated straightforwardly. These heuristics are easy to implement and efficient enough to run in real time. We provide formal analyses and simulation results to demonstrate and compare their performances.**

I. INTRODUCTION

The general scheduling problem is often formulated as the problem of arranging a set of tasks on a set of processors. Here, *processor* represents a generic term that can assume any form, such as a CPU, an airplane, or a robot. Along with the objective function, a scheduling problem is often specified as three components separated by |. These three components, written together as $P|T|func$, can be used to specify the type of processors, tasks, and the objective function. For more detailed discussions of the different scheduling problems that have been studied, refer to [1], [3].

To study the scheduling problems in robotic systems, one must first realize that these problems represent special cases of the general scheduling problem. In particular, the scheduling problems in robotic systems are often characterized by the following restrictions: 1) Execution is assumed to be non-preemptive; 2) Robots (and capabilities) are assumed to be non-divisible, such that the same robot can only work on a single task at any time point (thus restricting the robots to be single-task robots [4]). The first restriction holds especially in multi-robot tasks, since it is often costly to regroup robots and bring them into new coalitions [15]; the second restriction is often due to geographical distances between different task execution locations [13]. Consequently, algorithms in this paper are developed under these two restrictions, although preemptive task execution and multi-task robots can sometimes be beneficial. Furthermore, task dependencies (e.g., precedence orders) are not considered. However, note that the algorithms proposed in this paper can be extended when task dependencies are present, for example, using similar approaches as in [10], [16]. The solution bounds in these situations need to be re-established.

Following the above discussion, a simple approach to solve the scheduling problems in robotic systems is to use existing algorithms for the restricted problem instances. There are two cases to consider. The simpler case is scheduling with single-robot tasks, which can be specified as $R||func$. This problem with the objective function $\sum_l e_l$, can be solved in polynomial time using a solution for the assignment problem [1]. Here, R represents *unrelated machines*, since the times required for robots to accomplish the same task can be unrelated (or independent of their capabilities); l is the index for tasks and e_l denotes the finishing time of task t_l. The more complex case is with multi-robot tasks. Sometimes, when the multi-robot tasks can be divided into single-robot tasks, the scheduling problem reduces to $R||\sum_l e_l$. For situations in which this is infeasible, the problem becomes the $MPM\ MPT||\sum_l e_l$ problem (*multi-purpose machine* with *multi-processor task*). As a special case of $MPM\ MPT||\sum_l e_l$, the MPT problem is shown in [6] to be $\mathcal{NP}$-hard, even with only two machines, i.e., $MPT2||\sum_l e_l$.

Unfortunately, very few algorithms are provided to address this very difficult problem. As multi-robot systems develop, we need efficient algorithms to generate acceptable solutions fast in dynamic environments. In this paper, we propose four heuristics to address the $MPM\ MPT||\sum_l e_l$ problem. To the best of our knowledge, this is the first work that provides efficient heuristics for this problem with provable solution bounds. Furthermore, since these heuristics directly operate at the coalition level, they can be easily incorporated in a centralized or distributed manner using a layering technique [12]. The coalition information is provided by the underlying layer, in which robots search for coalitions and compute their costs based on the task requirements and robot capabilities (see [14] for an example). This information is then submitted to the upper layer (e.g., our heuristics) for task scheduling.

This paper is organized as follows. After a brief discussion of the related work in Section II, we formulate the problem in Section III. We introduce and analyze two simple heuristics in Section IV, and two more complex ones in Section V. The complexity analyses are provided in Section VI. Finally, we present results in Section VII and conclude in Section VIII.

This material is based upon work supported by the National Science Foundation under Grant No. 0812117.

Yu Zhang and Lynne Parker are with the Distributed Intelligence Laboratory in the Department of Electrical Engineering and Computer Science, University of Tennessee, Knoxville, TN 37996, USA, {yzhang51,parker}@eecs.utk.edu.

II. Related Work

It is first noted in [4] that the multi-robot task scheduling problem is equivalent to the *MPM MPT*$|| \sum_l e_l$ problem, although [4] does not assume single-task robots. This problem is often addressed by considering it as composed of two problems: an assignment problem and a *MPT* scheduling problem. The assignment problem determines the assignment of tasks to coalitions; the *MPT* scheduling problem then determines the task execution schedules. While the assignment problem can be solved optimally using the Hungarian algorithm [7], as we have discussed, the *MPT* problem is $\mathcal{NP}$-hard. To reduce the complexity, simplifying assumptions are often made, such as requiring the same processing times for tasks, restricting the number of processors, or allowing preemptive scheduling. However, the solution quality is often difficult to analyze after combining solutions of the assignment and the *MPT* problems.

On the other hand, the *MPT* problem is often formulated more generally as the *RCPSP* (resource-constrained project scheduling problem) problem [3]. The *RCPSP* problem reduces to the *MPT* problem when each robot is considered as one unique resource, such that neither the robots nor their capabilities are divisible (i.e., single-task robots). The corresponding *MPM MPT*$|| \sum_l e_l$ problem is then referred to as the multi-mode *RCPSP* problem. Many algorithms have been provided to address the *RCPSP* problem. Reviews of different methods can be found in [2], [5]. For example, mixed integer linear programming [8], as well as branch and bound algorithms [9], have been applied. Some of these algorithms have been extended to the multi-mode case [11]. However, none of these extensions represent efficient solutions with provable solution bounds.

Although the number of coalitions can be exponential in the number of robots, in multi-robot systems, the number of executable coalitions [15] is often limited. These coalitions are associated with assignments that do not have any unsatisfied preconditions. Consequently, the number of coalitions that need to be considered for any given task is also limited. This observation motivates the design of some of the heuristics in this paper. Furthermore, all desirable heuristics must be convenient to be incorporated. One solution is to use a layering technique [12], which requires the algorithms to directly operate at the coalition level.

III. Problem Formulation

We now present the formulation of the multi-robot task scheduling problem, or *MPM MPT*$|| \sum_l e_l$. Given:

- A set of tasks $T = \{t_1, t_2, ...\}$, and a set of robots R;
- A set of distinct coalitions $C = \{c_1, c_2, ...\}$, $c_j \subseteq R$;
- A function $f : C \times T \to \Re^+$, which returns the time, p_{jl}, required for a coalition c_j to accomplish a task t_l;

The scheduling problem we consider in this paper is:
Minimize:

$$\sum_l e_l \tag{1}$$

subject to the following restrictions:

1) Non-preemptive scheduling;
2) Non-divisible robot resources and capabilities;

Note that in this formulation, a coalition may be able to accomplish multiple tasks and a task may be accomplished by multiple coalitions. Again, e_l denotes the finishing time of task t_l. The reason that we are interested in $\sum_l e_l$ is because we often want to reduce the overall system operating time, including robot waiting or idle times, as they too consume resources. The second restriction adds *interference* between different coalitions, which prohibits coalitions sharing the same robots to work on different tasks at the same time.

IV. Simple Heuristics

A. MinProcTime

MinProcTime chooses at every step the assignment (i.e., coalition-task pair) that has the shortest processing time. This process creates an ordering of task assignments; we assume that ties are broken in a deterministic manner. For scheduling assignments, *MinProcTime* always assigns the earliest starting times. The intuition behind *MinProcTime* is to process tasks with shorter processing times earlier.

Theorem 4.1: The *MinProcTime* heuristic yields a solution quality bounded by $\frac{|T|+1}{2}$.

Proof: *MinProcTime* chooses an assignment for one remaining task in T at each step until all tasks are assigned. Given the greedy choice, when choosing a task t_l, the heuristic necessarily chooses the coalition that can accomplish t_l in the shortest time. Denote this processing time as p_l^{min}. We further suppose that tasks are ordered non-decreasingly based on p_l^{min}, which is also the order for *MinProcTime* to choose tasks. Then, the solution produced by *MinProcTime* must satisfy:

$$S^g(C,T) \leq \sum_l (|T| - l + 1) \cdot p_l^{min} \tag{2}$$

in which S^g is used throughout to represent the solutions of our heuristics (g stands for *greedy*, since there are always some greedy criteria involved). This inequality holds because, in the worst case, tasks would be accomplished in a sequential manner by their respective coalitions. On the other hand, the solution of the optimal solution clearly satisfies:

$$S^*(C,T) \geq \sum_l p_l^{min} \tag{3}$$

which happens only when all tasks are processed in parallel. Given the non-decreasing ordering, we know that $p_l^{min} \leq p_{l'}^{min}$ when $l \leq l'$. Combining Equations (2) and (3) yields:

$$\begin{aligned} S^g(C,T) &\leq \sum_l (|T| - l + 1) \cdot p_l^{min} \\ &\leq \frac{|T|+1}{2} \cdot \sum_l p_l^{min} \leq \frac{|T|+1}{2} \cdot S^*(C,T) \end{aligned} \tag{4}$$

Hence, the conclusion holds. ■

Although the relaxations in the proof seem to be pessimistic, the bound is actually tight; it is not difficult to create problem instances with the worst case solution quality.

B. *MinStepSum*

Similar to *MinProcTime*, *MinStepSum* always assigns the earliest starting time when considering an assignment. It then chooses at every step the assignment that contributes the least to the objective function. We again assume that ties are broken in a deterministic manner. Note that while the chosen assignments and their ordering are pre-determined in *MinProcTime* given the problem instance, they are determined incrementally (i.e., one assignment at each greedy step) in *MinStepSum*. Although the two greedy heuristics operate differently, interestingly, they share the same solution bound. We introduce the following lemma before proving this result, in which we still assume that the tasks are ordered non-decreasingly based on p_l^{min}.

Lemma 4.2: Denoting the finishing time of the l'th task chosen by *MinStepSum* as $e_{l'}$, we have that:

$$\max_{l'=1}^{k} e_{l'} \leq \sum_{l=1}^{k} p_l^{min} \tag{5}$$

Proof: We prove this result by an induction on k. Suppose that the inequality holds up to k. At step $k+1$, *MinStepSum* chooses an assignment that contributes the least to the objective function. There are two cases. Case 1, when all tasks in $\{t_1, ..., t_k\}$ are already assigned: since t_{k+1} is not assigned in this case, the least we can do is to have it accomplished by the coalition with processing time p_{k+1}^{min}. In such a case, we have:

$$\max_{l'=1}^{k+1} e_{l'} \leq \max_{l'=1}^{k} e_{l'} + p_{k+1}^{min} \leq \sum_{l=1}^{k+1} p_l^{min} \tag{6}$$

Case 2, when not all tasks in $\{t_1, ..., t_k\}$ are assigned: since tasks are ordered non-decreasingly based on p_l^{min}, the least we can do is to have any unassigned task $t_h (h \leq k)$ accomplished. In such a case, we have:

$$\begin{aligned}\max_{l'=1}^{k+1} e_{l'} &\leq \max_{l'=1}^{k} e_{l'} + p_h^{min} \\ &\leq \max_{l'=1}^{k} e_{l'} + p_{k+1}^{min} \leq \sum_{l=1}^{k+1} p_l^{min}\end{aligned} \tag{7}$$

Finally, since the induction clearly holds for $k = 1$, we have that the conclusion holds. ■

Theorem 4.3: The *MinStepSum* heuristic yields a solution quality bounded by $\frac{|T|+1}{2}$.

Proof: Since we know that $S^g(C, T) = \sum_{l'} e_{l'}$, the result follows directly from Lemma 4.2 and Equation (4). ■

Similarly, it can be shown that the bound in Theorem 4.3 is also tight. Although the number of tasks can be large, as we see in the simulation results section, both *MinProcTime* and *MinStepSum* produce solutions with quality much better than this bound on average.

V. COMPLEX HEURISTICS

One limitation of *MinProcTime* and *MinStepSum* is that they do not consider the *interference* between coalitions. Hence, they cannot make an informed decision when two assignments have similar processing times but different influences on other coalitions. However, a coalition can directly or indirectly influence other coalitions in complex ways. The heuristics in this section, nevertheless, assume that a coalition can only influence a limited number of coalitions, given the observations in [13], [15].

A. *InterfereAssign*

The idea that motivates *InterfereAssign* is the fact that the scheduling problem with single-robot tasks can be recast as an assignment problem, and hence can be solved optimally. To add in the effects of the interference, we first define the notion of interference:

Definition 5.1: Coalition Interference – For any two coalitions c_j and $c_{j'}$ ($j \neq j'$), c_j interferes (or conflicts) with $c_{j'}$ if and only if $c_j \cap c_{j'} \neq \emptyset$.

This definition applies similarly to assignments. Given this definition, we denote F_j as the set of coalitions that interfere with c_j. Observe that an assignment $c_j \to t_l$ can increase the objective function by the following three factors:

1) The assignment's processing time p_{jl};
2) The tasks that are scheduled on c_j after t_l;
3) The tasks that are scheduled on any coalitions in F_j.[1]

Denoting the scheduling position for t_l on c_j as I_{jl},[2] the first and second factors above together contribute $I_{jl} \cdot p_{jl}$.

The third factor can be estimated by the number of tasks that are scheduled on F_j. Although this number is not known a priori, we know that for each coalition c, the number of tasks that are scheduled on c must be no greater than $|N_c|$, in which N_c represents the set of tasks that c can accomplish. Hence, the third factor must be no greater than $|\cup_{c \in F_j} N_c|$.

Then, we can formulate the new assignment problem:

- Create a task node for each task t_l;
- Create a coalition-position node for each coalition c_j and position pair, with positions ranging from 1 to N_{c_j} for coalition c_j;
- If a coalition c_j can accomplish a task t_l, connect t_l with all coalition-position nodes for c_j, and set the weights to be $(|\cup_{c \in F_j} N_c| + I_{jl}) \cdot p_{jl}$, respectively, based on I_{jl}.

The above assignment problem can be solved optimally, which gives the positions of tasks to be executed on the coalitions. This solution satisfies the following property:

Lemma 5.1: There exists a schedule that is no worse than the solution of the assignment problem.

Proof: We prove this result by constructing a schedule from the solution of the assignment problem. This solution specifies the coalitions to execute the tasks and the sequences of these executions. The basic idea for the construction is to move from the beginning to the end of the entire execution and resolve any conflicts along the path. The resolution ordering guarantees that the process does not introduce new conflicts before where the current conflict

[1]Although these tasks (and the corresponding assignments) can recursively influence the objective function, such influences are too complicated to model and hence are not considered.

[2]Scheduling positions are numbered from the latest to the earliest task, such that the last task to be executed on a coalition is at position 1.

occurs. This process ends when no conflicts exist and the resulting schedule is valid.

All we need to show now is how to resolve a conflict. When there are conflicts with an assignment $c_j \to t_l$, given the resolution ordering, we know that no conflicts exist before the starting of its execution. Since conflicts can only occur when a coalition conflicts with c_j, we only need to adjust task executions scheduled on F_j. In the worst case, the number of assignments that we need to move to start time p_{jl} later is $|\cup_{c \in F_j} N_c|$. In such a way, the conflict is resolved and the process can move forward to the next assignment in the entire execution. ■

Theorem 5.2: The schedule that is constructed in Lemma 5.1 from the solution of the assignment problem yields a solution quality bounded by $\max_j |\cup_{c \in F_j} N_c| + 1$.

Proof: In the assignment problem, setting the connection weight to be $I_{jl} \cdot p_{jl}$ clearly establishes a lower bound on the objective function for the scheduling problem, since it assumes that there are no conflicts between the coalitions. Based on this observation and Lemma 5.1, the solution for the schedule that is constructed in Lemma 5.1 satisfies:

$$\begin{aligned} S^g(C,T) &\leq S^{assign}(T, \{C, I\}) \\ &\leq \max_j \frac{|\cup_{c \in F_j} N_c| + I_{jl}}{I_{jl}} \cdot S^*(C,T) \\ &\leq (\max_j |\cup_{c \in F_j} N_c| + 1) \cdot S^*(C,T) \end{aligned} \quad (8)$$

Hence, the conclusion holds. ■

Breaking Ties: Note that from the construction in Lemma 5.1, we know that when there are ties in resolving the conflicts (i.e., when multiple conflicting assignments are scheduled to start at the same time), we can choose any one of these assignments to perform the resolution process. In *InterfereAssign*, we choose the one that minimizes its contribution to $\sum_l e_l$ with consideration of the interference (i.e., computed as e_l in addition to the necessary moves for resolving the conflicts).

Note that the solution quality of *InterfereAssign* is dependent on the problem instance. For cases when the number and size of coalitions are limited (e.g., due to communication and coordination costs, or restrictions from location constraints), such that the coalitions interfere with only a limited number of other coalitions, *InterfereAssign* is likely to produce good quality solutions. In particular, when only single-robot coalitions are allowed, the problem reduces to an assignment problem and *InterfereAssign* produces the optimal solution.

B. *MinInterfere*

In *InterfereAssign*, $|\cup_{c \in F_j} N_c|$ is often too much of an overestimate of the number of tasks that are scheduled on F_j. However, one cannot make a more precise estimation before some assignments are made. *MinInterfere* is a greedy heuristic introduced to address this issue. The idea is to find an upper bound on the number of tasks scheduled on F_j that is as tight as possible, so that the estimation of the interference becomes more accurate. At step k during the greedy process, *MinInterfere* has already made $k-1$ assignments. Then, for every remaining task t_l, for any coalition c_j that can execute t_l, *MinInterfere* schedules the assignment at the earliest time possible, i.e., the earliest time following the last chosen assignment on c_j that incurs no conflicts. *MinInterfere* computes the following value as the greedy criterion to minimize:

$$\beta_{jl} = e_{jl} + |\cup_{c \in F_j} N_c \setminus M_{jl}| \cdot p_{jl} \quad (9)$$

in which M_{jl} includes t_l, and the set of tasks that are scheduled before $c_j \to t_l$ in the greedy process. *MinInterfere* then schedules an assignment according to $\arg\min_{j,l} \beta_{jl}$.

The formal analysis of *MinInterfere* is left as future work. To provide an example of how these heuristics work, we provide the algorithm for *MinInterfere* as follows.

Algorithm 1 *MinInterfere* for multi-robot task scheduling

```
 1: while there are tasks remaining do
 2:     for all $t_l \in$ tasks remaining do
 3:         for all $c_j$ that can accomplish $t_l$ do
 4:             Compute the earliest starting time $s_{jl}$.
 5:             Compute $e_{jl} = s_{jl} + p_{jl}$.
 6:             Compute $\beta_{jl}$ in Equation (9).
 7:         end for
 8:     end for
 9:     Select $j, l$ according to $\arg\min_{j,l} \beta_{jl}$.
10:     Assign task $t_l$ to $c_j$.
11: end while
```

VI. COMPLEXITY ANALYSIS

To solve the *MPM MPT*$||\sum_l e_l$ problem optimally, one needs to search all orderings of tasks. For each ordering, we must iterate through all possible coalition assignments for each task. Since finding the earliest starting time takes $O(|T|)$, the complexity is $O((|C||T|)^{|T|}|T|!)$.[3] Clearly, computing the optimal solution is intractable.

MinProcTime requires us to first sort the tasks based on their minimum processing times. This process takes $O(|C||T| + |T|\log|T|)$ using a set structure. For scheduling each task, since finding the starting time takes $O(|T|)$, the complexity of *MinProcTime* is $O(|C||T| + |T|^2)$.

At every greedy step, *MinStepTime* checks every assignment's contribution to the objective function. This process takes $O(|C||T|^2)$. Hence, the computational complexity of *MinStepTime* is $O(|C||T|^3)$.

InterfereAssign is a direct application of the Hungarian algorithm. Hence, the complexity is $O(|C|^3|T|^3)$, given that the number of coalition-position pairs is $O(|C||T|)$. The complexity for breaking ties is absorbed.

Similarly, from Algorithm 1, it can be inferred that the computational complexity of *MinInterfere* is $O(|C||T|^3)$. Table I provides a summary of the discussed heuristics, including solution bounds, and computational complexities.

[3]Although $|C|$ can be exponential in $|R|$, the number of coalitions that need to be considered is often much smaller in practice [10], [13], [15], and thus the problem can be solved quickly.

TABLE I
SUMMARY OF DISCUSSED HEURISTICS

Name	Solution Bound	Complexity
Optimal	1	$O((\lvert C\rvert\lvert T\rvert)^{\lvert T\rvert}\lvert T\rvert!)$
MinProcTime	$\frac{\lvert T\rvert+1}{2}$	$O(\lvert C\rvert\lvert T\rvert^3)$
MinStepTime	$\frac{\lvert T\rvert+1}{2}$	$O(\lvert C\rvert\lvert T\rvert^3)$
InterfereAssign	$\max_j \lvert \cup_{c\in F_j} N_c\rvert + 1$	$O(\lvert C\rvert^3\lvert T\rvert^3)$
MinInterfere	Not Determined	$O(\lvert C\rvert\lvert T\rvert^3)$

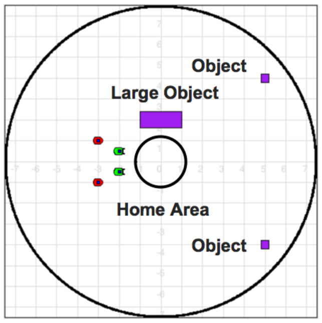


Fig. 1. A simple scenario of the scheduling problem.

TABLE II
TASKS FOR THE SCENARIO IN FIGURE 1

Task	Robots Required	Process Time
1) Object 1	One gripper, one localizer	6
2) Object 2	One gripper, one localizer	6
3) Large Object	Two grippers	5

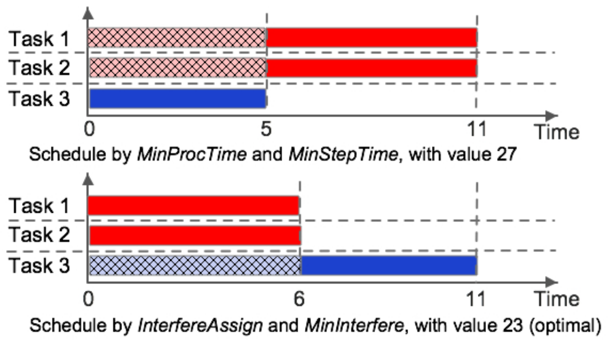


Fig. 2. Schedules created by the heuristics for the scenario in Table II.

VII. SIMULATION RESULTS

Simulations are run on a 2.4GHz laptop with 2GB memory; wall-clock times are reported. Statistics are collected over 100 runs; problem instances are randomly generated.

A. An Object Collection Scenario

We start with an object collection scenario, as shown in Figure 1, to demonstrate interference between assignments. There are two types of robot, and two robots for each type. The first type has a gripper but cannot localize; the second type can localize but does not have a gripper. The goal is to move all three objects to the home area. While the smaller objects can be moved by one gripper robot, they are relatively far from the home area so that a localization capability is required. The larger object can only be moved by two gripper robots cooperatively but is closer to the home area. Note that although the task execution times are only dependent on the distances between the home area and the objects in this scenario, they can also be dependent on the current robot locations and other dynamic factors. This is related to the issue of task dependency and is not specifically considered in this work. See [16] for some discussions on this issue.

Table II shows the robots required for each object and the time required to accomplish the move. Figure 2 illustrates the schedules created by the four proposed heuristics. We can see that only the heuristics that consider the interference between the assignments (i.e., *InterfereAssign* and *MinInterfere*) produce the optimal solution. While this simulation presents only a small example, the impact can be large as the problem size increases, which is shown next.

B. Performance Comparison

First, the parameters used in the following simulations are listed in Table III. In the first simulation, we vary n_c from 3 to 6 while keeping the other parameters fixed. Due to the complexity for computing the optimal solution, we set $n_t = 9$. Other parameters are: $n_f = (0.9 - 1.8)$ (based on n_c), $n_e = 2.7$, $n_{min} = 5$, and $n_{max} = 10$. The result is shown in Figure 3. We can see that the performance generally decreases as n_c increases for all heuristics. This may seem to be counter-intuitive. However, note that as n_c increases, n_f also increases, which complicates the scheduling. We can also see that *MinStepSum*, *InterfereAssign* and *MinInterfere* perform similarly while *InterfereAssign* is slightly better (at the 5% significance level) than *MinStepSum* and *MinInterfere* for smaller n_c (i.e., $3 - 4$). Another observation is that the average solution quality is better than the proven bounds.

In the next simulation, we vary n_t from 8 to 11. The other parameters are: $n_c = 4$, $n_f = 1.2$, $n_e = (2.4 - 3.3)$ (based on n_t), $n_{min} = 5$, and $n_{max} = 10$. The result is shown in Figure 4. We can see similar conclusions as in the previous simulation (i.e., *InterfereAssign* performs significantly better than *MinStepSum* and *MinInterfere* when $n_t = (8-9)$). Since n_f stays as a constant this time, we can see that the curve formed by the solution quality is smoother.

Next, we show how the performance changes while we vary n_f from 0.5 to 2.5. Other parameters are: $n_c = 5$, $n_t = 10$, $n_e = 3$, $n_{min} = 5$, and $n_{max} = 10$. Figure 5 gives the result. Again, we can see that the performance decreases as the interference becomes more complex (or as n_f increases).

We are also interested in how n_{max} influences the solution. In this simulation, we vary n_{max} from 5 to 10. Other

TABLE III
PARAMETERS USED IN THE SIMULATIONS

Parameter	Description
n_c	No. of coalitions
n_t	No. of tasks
n_f	Average no. of conflicting coalitions per coalition
n_e	Average no. of executable tasks per coalition
n_{min}, n_{max}	Minimum and maximum processing time

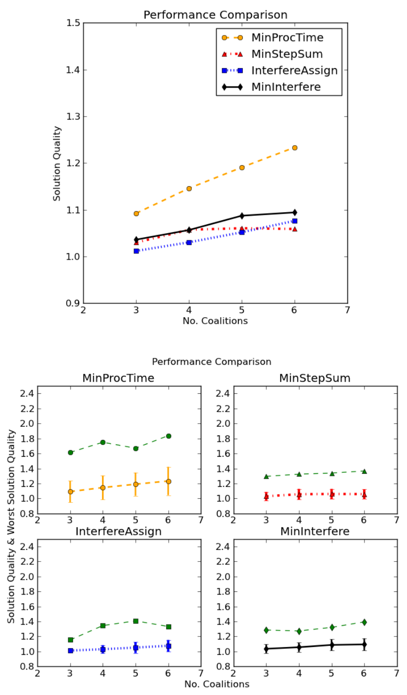


Fig. 3. Scheduling with varying n_c. Top: Average solution quality (the lower the better). Bottom: Separate detailed results with standard deviations. The green data points in each subgraph represent the worst solution quality.

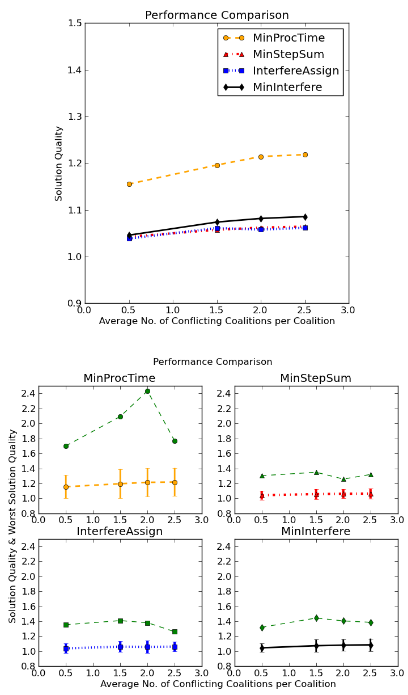


Fig. 5. Scheduling with varying n_f. Top: Average solution quality. Bottom: Separate and more detailed results with standard deviations.

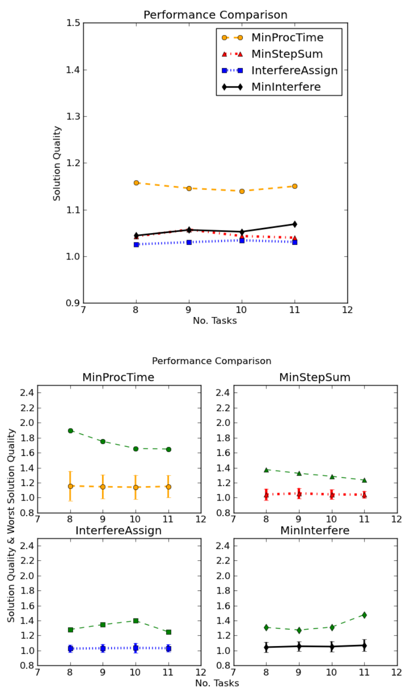


Fig. 4. Scheduling with varying n_t. Top: Average solution quality. Bottom: Separate and more detailed results with standard deviations.

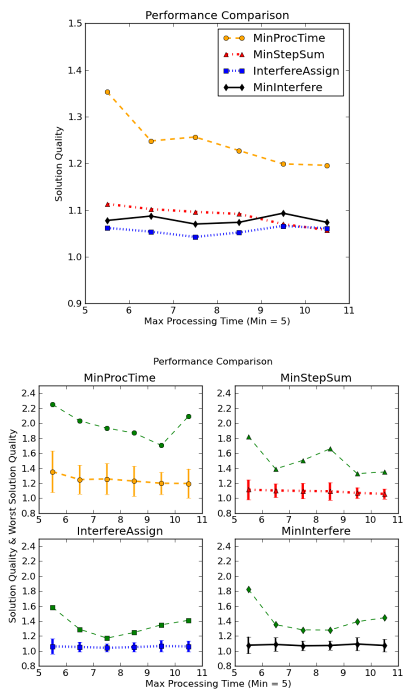
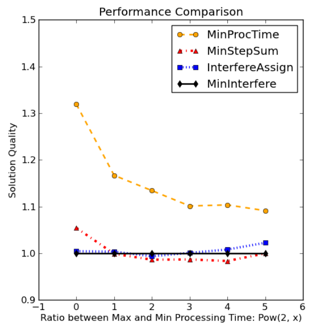


Fig. 6. Scheduling with varying n_{max}. Top: Average solution quality. Bottom: Separate and more detailed results with standard deviations.

parameters are: $n_c = 5$, $n_t = 10$, $n_f = 1.5$, $n_e = 3$, and $n_{min} = 5$. The result is shown in Figure 6. We can see from the figure that the performance decreases as n_{max} increases, which is different from the previous simulations. This is understandable because it is more difficult to schedule assignments when their processing times are similar but their interferences on other assignments differ.

Now, we want to see how the heuristics perform with large



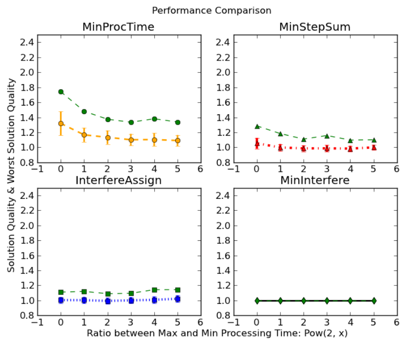


Fig. 7. Scheduling with varying n_{max}, with large n_c and n_t. Top: Average solution quality. Bottom: Separate detailed results with standard deviations.

n_c and n_t. Due to the complexity for computing the optimal solutions, we use *MinInterfere* as the standard to compare against. This time, we vary n_{max} from 5 to 160. We set: $n_c = 25$, $n_t = 50$, $n_f = 1.5$, $n_e = 3$, and $n_{min} = 5$. Figure 7 shows the result. We can see the same trend in the solution quality as in Figure 6. Another observation is that the change of n_{max} does not seem to have a significant influence on the comparative performances of these heuristics.

Finally, we perform time analyses on these heuristics. For the first result, we vary n_c from 15 to 30; for the second, we vary n_t from 40 to 55. Other parameters remain the same as in the previous simulation. Figure 8 shows the result, which aligns with Table I. All simulations together show that: when the coalitions interfere with only a limited number of other coalitions (i.e., $0-1$), *InterfereAssign* can be used to produce better solutions (e.g., see Figure 4, or Figure 3 when $n_c = (3-4)$), since they explicitly consider this interference in their formulations; otherwise, when the interference becomes difficult to be modeled accurately by our approach, *InterfereAssign*, *MinInterfere*, and *MinStepSum* perform similarly. In such cases, we can run these three heuristics and choose the best schedule produced.

VIII. CONCLUSIONS AND FUTURE WORK

In this paper, we have introduced four efficient heuristics to address the *MPM MPT*$||\sum_l e_l$ problem. We have formally analyzed them and provided simulations to demonstrate and compare their performances. Furthermore, it is convenient to incorporate them using a layering technique.

For future work, it is interesting to study the solution quality of *MinInterfere* and other heuristics, to see if the solution bounds can be further improved. It is also useful to establish bounds on the approximation ratios that any polynomial-time algorithms can achieve. Finally, we also look forward to implementing these heuristics on multi-robot systems to address real-world scheduling problems.

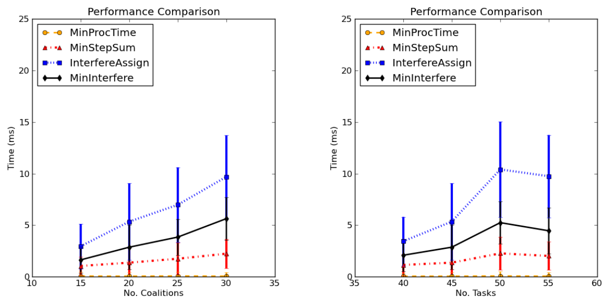


Fig. 8. Time analysis. Left: Varying n_c. Right: Varying n_t.

REFERENCES

[1] P. Brucker. *Scheduling Algorithms*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2001.
[2] P. Brucker, A. Drexl, R.H. Mohring, K. Neumann, and E. Pesch. Resource-constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operational Research*, 112(1):3–41, 1999.
[3] P. Brucker and S. Knust. *Complex Scheduling*. Gor-Publications. Springer, 2006.
[4] B.P. Gerkey and M.J. Mataric. A formal analysis and taxonomy of task allocation in multi-robot systems. *The International Journal of Robotics Research*, 23(9):939–954, September 2004.
[5] S. Hartmann and R. Kolisch. Experimental evaluation of state-of-the-art heuristics for the resourc-constrained project scheduling problem. *European Journal of Operational Research*, 127:394–407, 1998.
[6] J.A. Hoogeveen, S.L. van de Velde, and B. Veltman. Complexity of scheduling multiprocessor tasks with prespecified processor allocations. *Discrete Applied Mathematics*, 55(3):259–272, 1994.
[7] H. W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistic Quarterly*, 2:83–97, 1955.
[8] E. Neron, C. Artigues, P. Baptiste, J. Carlier, J. Damay, S. Demassey, and P. Laborie. Lower bounds for resource constrained project scheduling problem. In *Perspectives in Modern Project Scheduling*, volume 92, pages 167–204. Springer US, 2006.
[9] J.H. Patterson, F. Brian Talbot, R. Slowinski, and J. Weglarz. Computational experience with a backtracking algorithm for solving a general class of precedence and resource-constrained scheduling problems. *European Journal of Operational Research*, 49(1):68–79, Nov. 1990.
[10] O. Shehory and S. Kraus. Methods for task allocation via agent coalition formation. *Artificial Intelligence*, 101(1-2):165 – 200, 1998.
[11] A. Sprecher and A. Drexl. Multi-mode resource-constrained project scheduling by a simple, general and powerful sequencing algorithm. *European Journal of Operational Research*, 107(2):431–450, 1998.
[12] F. Tang and L.E. Parker. A complete methodology for generating multi-robot task solutions using ASyMTRe-D and market-based task allocation. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 3351–3358, Apr. 2007.
[13] L. Vig and J.A. Adams. Multi-robot coalition formation. *IEEE Transactions on Robotics*, 22(4):637–649, 2006.
[14] Y. Zhang and L.E. Parker. IQ-ASyMTRe: Forming executable coalitions for tightly coupled multirobot tasks. *IEEE Transactions on Robotics*, PP(99):1–17, 2012.
[15] Y. Zhang and L.E. Parker. Task allocation with executable coalitions in multirobot tasks. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 3307–3314, May 2012.
[16] Y. Zhang and L.E. Parker. Considering inter-task resource constraints in task allocation. *Autonomous Agents and Multi-Agent Systems*, 26(3):389–419, 2013.