Scheduling Operator Assistance for Shared Autonomy in Multi-Robot Teams

Cai, Yifan ; Dahiya, Abhinav ; Wilde, N.; Smith, Stephen L.

DOI
10.1109/CDC51059.2022.9993014

Publication date
2022

Document Version
Final published version

Published in
Proceedings of the IEEE 61st Conference on Decision and Control (CDC 2022)

2022 IEEE 61st Conference on Decision and Control (CDC)
December 6-9, 2022. Cancún, Mexico

Scheduling Operator Assistance for Shared Autonomy in Multi-Robot Teams

Yifan Cai, Abhinav Dahiya, Nils Wilde, Stephen L. Smith

2022 IEEE 61st Conference on Decision and Control (CDC) | 978-1-6654-6761-2/22/$31.00 ©2022 IEEE | DOI: 10.1109/CDC51059.2022.9993014

***Abstract*— In this paper, we consider the problem of allocating human operator assistance in a system with multiple autonomous robots. Each robot is required to complete independent missions, each defined as a sequence of tasks. While executing a task, a robot can either operate autonomously or be teleoperated by the human operator to complete the task at a faster rate. We formulate our problem as a Mixed Integer Linear Program, which can be used to optimally solve small to moderate sized problem instances. We also develop an anytime algorithm that makes use of the problem structure to provide a fast and high-quality solution of the operator scheduling problem, even for larger problem instances. Our key insight is to identify *blocking tasks* in greedily-created schedules and iteratively remove those blocks to improve the quality of the solution. Through numerical simulations, we demonstrate the benefits of the proposed algorithm as an efficient and scalable approach that outperforms other greedy methods.**

I. Introduction

Autonomous mobile robot teams have been widely used in manufacturing and related sectors resulting in improved productivity and reduced risk to human workers. Such robot teams are able to function autonomously on their own, while also bearing the capability of making use of human assistance to further improve their performance [1]–[3]. As it is challenging for human operators to supervise and assist a large number of robots on their own [4], [5], a number of studies in the literature propose effective decision support systems (DSS) to aid the human operator(s) in providing assistance [1], [6], [7].

In this paper, we present such a DSS for a multi-robot system comprising a fleet of autonomous robots with a human operator available to teleoperate the robots to speed up their missions, given their availability. Figure 1 presents an overview of the problem setup, showing K robots navigating in a city-block-like environment and going through a series of tasks. A task in this example may refer to navigating through the robot route, crossing a road, going through a crowded area, and etc. Each task is characterized by different completion times, depending on whether the task is executed autonomously or under teleoperation. There is a human operator available, who can assist/teleoperate at most one robot at a time. All robots are capable of completing their respective tasks on their own, but can be assisted by a human operator to speed up the task completion. The DSS provides the operator with a teleoperation schedule that specifies which task of a robot should be executed via teleoperation, and in what order. If a robot task is scheduled for teleoperation, then the robot and operator must *wait* for each other to be available before starting this task. Thus, a schedule specifies the teleoperation actions for the operator and the wait actions for all robots and the operator.

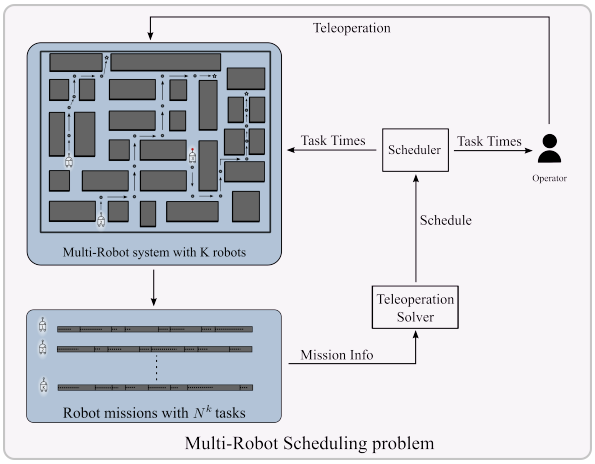

Fig. 1: Information flow in the multi-robot teleoperation scheduling system with K robots. A robot k are assigned with an independent series of tasks. Given mission information, solver computes the schedule, which is then converted to information about task timing for both operator and robots. The operator assists on tasks assigned by the schedule.

The problem objective is to find a teleoperation schedule for both the human operator and robots that minimizes the time taken until all robot missions are complete.

Our work makes the following contributions:

1) We formulate a Mixed Integer Linear Program (MILP) that can be used to generate optimal schedules for the given problem. We also present an extension to a multiple-operator version of the problem.

2) We develop an anytime algorithm that iteratively generates teleoperation schedules for the given problem. The algorithm is capable of solving much larger instances of the given problem than the MILP formulation.

3) We evaluate our proposed algorithm in numerical simulations. The results show that our method provides an efficient and scalable solution compared to other approaches.

A. Background and Related Works

Human-multi-robot teams have found their application in search-and-rescue [8], smart factory operation [9], home care

This research is supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC) and in part by the Innovation for Defence Excellence and Security (IDEaS) Program of the Canadian Department of National Defence through grant CFPMN2-037.

Y. Cai, A. Dahiya, and S. L. Smith are with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON N2L 3G1, Canada {yifan.cai, abhinav.dahiya, stephen.smith}@uwaterloo.ca. N. Wilde is with the Cognitive Robotics Department, Delft University of Technology, Netherlands (N.Wilde@tudelft.nl).

for seniors [10], and package delivery [6]. However, such a team composition also brings the risk of increasing operator workload and decreasing in their situational awareness [11], [12]. In [13], the authors show that scheduling the operator's attention can improve the efficiency of control over multi-robot system. Therefore, such systems can benefit from having a DSS that decides how to distribute human assistance among different robots or autonomous systems [1], [14].

The problem of scheduling human assistance among multiple robots has similarities with disciplines of multi-robot supervision, queuing theory, and task scheduling and sequencing. All these studies propose some forms of DSS, where an advising agent guides the human operator(s) on a robot (or task) which they should assist, with specified time. This advice can take form of an online allocation, like in [6], or a pre-determined offline schedule, like in [15]. In human-supervised multi-robot systems, frameworks such as sliding autonomy that considers factors like coordination and situational awareness are shown to improve understanding of such systems [16], [17]. Research on effective interaction interfaces also aims to facilitate human supervision of robot teams [18], [19]. Our work is concerned with providing instructions to human operator on how to allocate their attention among different robots. In the queuing discipline, efficient techniques have been developed to enable a human to service a queue of tasks [20]. However, the model that we study is different from a queuing model as it is possible for the robots to complete their tasks without the help of operators, and there is no pre-defined order in which tasks (of different robots) are required to be processed.

Related studies in scheduling literature present methods to schedule processing of different tasks to minimize performance metrics like makespan, idle time etc. A common way of solving the scheduling problem is through the MILP formulation, which can be used to obtain optimal solutions for scheduling problems. In the literature, we also find scalable methods to approximately solve a MILP for large instances which may take MILP hours to find the minima. For example, the study presented in [21] makes use of a heuristic procedure for a single machine job scheduling. However, in our system not all tasks are required to be scheduled and tasks from different robots are not required to be in any particular order. Methods like rolling-horizon splits problems into smaller pieces based on time and pursue the local optimal [22]. In contrast to the problem considered in [22], our problem is highly-coupled over time, and thus there aren't natural breakpoints in time to decompose the problem.

The most related works to our problem are presented in [23] and [15]. These studies propose solutions to scheduling of operators, and robot planning for multi-robot system having critical configurations where operator attention/input is required to proceed. While sharing a similar goal with these studies (minimizing mission time), our system lacks the presence of any such critical configurations or states, and every task can be completed both autonomously and under teleoperation.

II. Multi-robot Teleoperation Scheduling

We consider a system consisting of a human operator supervising a fleet of K autonomous robots. Each robot $k \in \mathcal{K} := \{1, \ldots, K\}$ is assigned a mission $p^k \in \mathcal{P} := \{p^1, \ldots, p^K\}$, which is a pre-defined sequence of tasks. To complete its mission p^k the robot k is required to complete N_k tasks. The j^{th} task of robot k is denoted as e^k_j. For each task, a robot can either operate autonomously or be teleoperated by the human operator. Executing a task e^k_j takes time α^k_j if the robot operates autonomously and time $\beta^k_j (\leq \alpha^k_j)$ if it is teleoperated[1].

There is a DSS that provides a teleoperation schedule for the operator. A complete teleoperation schedule contains the information of when to start each task of every robot and which of the tasks are teleoperated. This information also tell us if a robot or an operator needs to wait before starting a task. However, since the completion times for each task are known, this teleoperation schedule can be presented in a more compact form as only a sequence of teleoperated tasks. The timing information can be computed in polynomial time from this sequence using the time α^k_j and β^k_j.

For our problem, we consider a schedule $\mathcal{S}$ as a sequence of tasks $\langle s_1, \ldots, s_n \rangle$ where each s_i corresponds to some task e^k_j for $k \in \{1, \ldots, K\}, j \in \{1, \ldots, N^k\}$ that is required to be teleoperated. Once the mission starts, the human operator teleoperates the specific tasks in the order provided by the schedule $\mathcal{S}$, i.e., task s_1 followed by s_2 and so on. If at the end of some task $s_i = e^k_j$, the robot k is not yet ready for the required task (executing its previous tasks), the operator waits for the robot to arrive at the start of e^k_j. Likewise, if the robot is ready for the task s_i, but the operator is still working on a previous task $s_{i'}$ where $i' < i$, then the robot waits for the operator.

The mission ends when all robots complete their respective sequence of tasks. A common metric of measuring performance of such systems is the time elapsed until all robot missions are complete, called the *makespan* [24], denoted as $\mu(\mathcal{S}) \in \mathbb{R}_{>0}$.

A. Problem Statement

We impose the following assumptions on the problem:

- **(A1)** The operator teleoperates at most one robot at a time.
- **(A2)** A task's mode of operation cannot change once the task is started, i.e., an operator must teleoperate a robot throughout a task, and they cannot join a task which already started autonomously.
- **(A3)** All robots may start with the first task in their respective missions at or after the time $t = 0$.

The objective is to solve the following optimization.

[1]In this paper we consider $\beta^k_j \leq \alpha^k_j$, i.e., teleoperation is at least as fast as autonomous operation. However, even for cases when this condition does not hold, the analysis and algorithms presented in this paper apply without any changes, as the tasks where autonomous operation is faster than teleoperation are not considered for scheduling.

Problem 1. Given the set $\mathcal{K}$ of robots, the missions $\{p^1, \ldots, p^K\}$ for each robot, and the autonomous and teleoperation completion times α_j^k and β_j^k for each task, find a schedule $\mathcal{S}$ that minimizes the makespan $\mu(\mathcal{S})$.

B. Hardness of Problem 1

An NP-complete variant of Satisfiability called *2p1n-3SAT* [25] can be reduced to the decision version of Problem 1. Thus, the decision problem is in NP-Complete[2]. Then, the problem of finding the optimal teleoperation schedule in Problem 1 is NP-Hard.

III. MILP Formulation

Since all constraints in our problem are linear time constraints, we formulate our problem as a mixed integer linear program (MILP). In the MILP formulation, our objective is to find a schedule $\mathcal{S}$ that minimizes team makespan $\mu(\mathcal{S})$, subject to conditions on system dynamics and task ordering. We begin by introducing three variables for each task: (1) x_j^k, a binary teleoperation variable for task e_j^k, (2) τ_j^k, the scheduled start time for e_j^k, and (3) ε_j^k, the finish time for e_j^k, which can be expressed as a sum of the τ_j^k and the task completion time under the schedule, i.e.,

$$\varepsilon_j^k = \tau_j^k + (1 - x_j^k)\,\alpha_j^k + x_j^k\,\beta_j^k.$$

A MILP can then be formulated as follows:

$$\begin{aligned}
\text{Minimize:}\quad & \bar{\mu} \\
\text{Subject to:}\quad & \bar{\mu} \geq \varepsilon_{N^k}^k \qquad \forall\, k \in \mathcal{K}, && (1)\\
& \tau_1^k \geq 0, \qquad \forall\, k \in \mathcal{K}, && (2)\\
& \tau_j^k \geq \varepsilon_{j-1}^k,\ \forall\, k \in \mathcal{K},\, j \in \{2, \ldots, N^k\}, && (3)\\
& x_j^k + x_i^l = 2 \implies \tau_j^k \geq \varepsilon_i^l \ \text{ or } \ \tau_i^l \geq \varepsilon_j^k, \\
& \qquad \forall\, k, l \in \mathcal{K};\, k \neq l, \\
& \qquad \forall\, j \in \{1, \ldots, N^k\}, \\
& \qquad \forall\, i \in \{1, \ldots, N^l\}, && (4)\\
& x_j^k \in \{0, 1\}, \forall\, k \in \mathcal{K},\, j \in \{1, \ldots, N^k\}. && (5)
\end{aligned}$$

Constraint (1) restricts the time needed for every robot to complete its mission to be not more than the objective $\bar{\mu}$. Constraint (2) sets the earliest start time for the robots. Constraint (3) ensures that the j^{th} task of a robot mission can only starts after the $j-1^{th}$ task is completed. Constraint (5) restricts the variables x_j^k to be a binary variable. Constraint (4) specifies no two tasks can be teleoperated with an overlapping time interval. The exclusive disjunction (XOR) of two conditions is required due to the undetermined order of teleoperation of the two tasks. Note that constraint (4) above is presented as an implication and is not written as a linear constraint. However, it can be converted to a set of linear constraints (for example, by using the Big-M method), which are supported directly by many mixed integer linear program solvers [26].

[2]Details of this proof can be found in this arXiv preprint of the paper: https://arxiv.org/abs/2209.03458

Note: To implement constraint (4), we can limit the ranges to $k \in \{1, \ldots, K-1\}$, $l \in \{k, \ldots, K\}$, which eliminates the repetitions in constraint checking, thus more efficient.

A. Extension to Multiple Operators

It is worth noting that we can directly extend the MILP to handle the multi-operator-multi-robot setting. In this case we have a set of M operators $\mathcal{M} := \{1, \ldots, M\}$, and use binary variable x_{jm}^k to indicate whether e_j^k is teleoperated by operator $m \in \mathcal{M}$. Whether a task is teleoperated or not is now indicated by $\sum_{m \in \mathcal{M}} x_{jm}^k$, instead of x_j^k. Consequently, changes are made in expressions for ε_j^k and Constraint (4). Constraint (5) is repeated for all x_{jm}^k.

In addition, we need a constraint to bound $\sum_{m \in \mathcal{M}} x_{jm}^k$, since each task can be assigned to at most one operator:

$$\sum_{m \in \mathcal{M}} x_{jm}^k \leq 1,\, \forall\, k \in \{1, \ldots, K\},\, j \in \{1, \ldots, N^k\}. \quad (6)$$

B. Solving the MILP

A globally optimal solution to a MILP can be found using solvers like Gurobi or CPLEX. However, as mentioned earlier, while such solvers are effective for small problem instance (i.e. 2 robots with 8 tasks each, such an instance takes about 8.3 sec for MILP), they do not scale to large instances (i.e. 3 robots with 15 tasks each, such an instance takes about 296 sec for MILP), each with ten or more tasks in its mission. In the next section, we present an efficient algorithm that makes use of the problem structure to provide a fast and high-quality solution of Problem. 1.

IV. Iterative Greedy

In this section, we present a greedy algorithm called `Iterative Greedy`. The algorithm begins by greedily creating a schedule to improve the team's makespan, until no further improvements can be made by adding tasks of a makespan robot to the schedule. Our key insight here is to then identify *blocking tasks* in such greedily-created schedules and iteratively remove those blockages to improve the solution. The algorithm cycles between two routines: Greedy Insertion and Block Removal.

A. Greedy Insertion

This routine creates a teleoperation schedule by greedily selecting tasks from the mission of a robot whose total time currently equals the makespan (called a makespan robot).

Definition 1 (Greedy Insertion)**.** For a given schedule $\mathcal{S}$, let robot k be a robot achieving the makespan (i.e., last task's finish time $\varepsilon_{N^k}^k = \mu(\mathcal{S})$). We call the addition of a task e_i^k to schedule $\mathcal{S}$ a Greedy Insertion if the addition of e_i^k directly reduces $\varepsilon_{N^k}^k$, without increasing the team makespan.

Pseudo-code for the `Greedy Insertion` algorithm is presented in Algorithm 1. In the algorithm, given a schedule $\mathcal{S}$, we first identify the set of all makespan robots, denoted as $\overline{\mathcal{K}}$. We then determine the *best* task e^k, defined as the task that reduces $\varepsilon_{N^k}^k$, the mission time of any robot $k \in \overline{\mathcal{K}}$ by

Algorithm 1 `Greedy Insertion`

Input: $\mathcal{P}$, $\mathcal{S}$
Output: $\mathcal{S}'$

Initialize $\Delta\varepsilon^* = 0$, $\mathcal{S}' = \mathcal{S}$
Calculate mission time $\varepsilon^k_{N^k}$ for $k \in \{1, \ldots, K\}$ given $\mathcal{P}$, $\mathcal{S}$
$\overline{\mathcal{K}} \leftarrow \arg\max_k \{\varepsilon^1_{N^1}, \ldots, \varepsilon^K_{N^K}\}$
for $k \in \overline{\mathcal{K}}$ **do**
    Find the `Best` task e^k, with time reduction $\Delta\varepsilon^k_{N^k}$ and corresponding schedule $\mathcal{S}^k$ given $\mathcal{P}$, $\mathcal{S}$
    if $\Delta\varepsilon^k > \Delta\varepsilon^*$ **then**
        $\Delta\varepsilon^* \leftarrow \Delta\varepsilon^k_{N^k}$; $\mathcal{S}' \leftarrow \mathcal{S}^k$
return $\mathcal{S}'$

the most, while not increasing the makespan $\mu(\mathcal{S})$. This task is then inserted in the schedule.

Note: The `best` task in the Greedy Insertion algorithm is defined as the one which results in the most reduction in mission time of any makespan robot.

An example is shown in Fig. 2 to illustrate its operation. Robot 3 is the makespan robot, and has two tasks currently not in the schedule. Select the one with more time reduction, and this addition to the schedule will reduce $\mu(\mathcal{S})$.

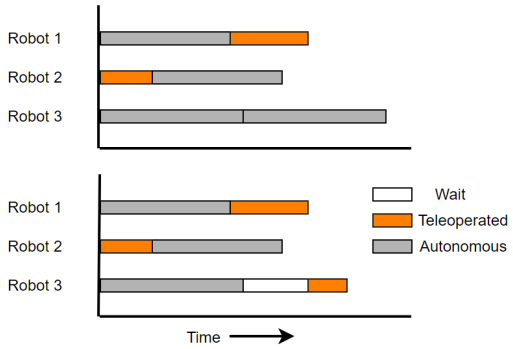

Fig. 2: Example of Greedy Insertion. Robot 3 is the makespan robot, and by teleoperating its last task, we reduce its total mission time.

B. Block Removal

A schedule created using `Greedy Insertion` gives a feasible (locally optimal) solution but such a schedule often results in considerable and frequent idle times for the operator. However, even when the system's makespan cannot be improved further by adding more tasks of makespan robots to the schedule, it may be possible to improve the makespan by adding tasks from other robots. This is the idea behind the Block Removal technique, which works by finding and eliminating *blocking tasks* and reduces the team makespan by reducing waiting times in the schedule. We begin to introduce the details of this technique with the following definitions.

Definition 2 (Idle Time)**.** For any two adjacent tasks s_j and s_{j+1} in schedule $\mathcal{S} = \langle s_1, \ldots, s_n \rangle$, the idle time is defined as the time between task s_j finish time and s_{j+1} start time. For s_1, if its start time > 0, idle time is simply the start time of itself.

Definition 3 (Blocking Task and Blocking Robot)**.** A task s_{j+1} in schedule $\mathcal{S}$ is called a blocking task if the idle time between s_j and s_{j+1} is greater than zero[3]. The robot to which task s_{j+1} belongs to is called a blocking robot.

A blocking task is called so because it prevents a task in the makespan robot's plan from getting teleoperated or being teleoperated at an earlier time. Reducing the starting time of the blocking task indirectly results in a smaller makespan or allows for further makespan decrease in future iterations. With above, the Block Removal operation can be defined.

Definition 4 (Block Removal)**.** Given a schedule $\mathcal{S}$, let robot k be a robot achieving the makespan (i.e., $\varepsilon^k_{N^k} = \mu$). We call the addition of a task $e^{k'}_i$ from a non-makespan robot k' to the schedule $\mathcal{S}$ a Block Removal if the addition of $e^{k'}_i$ reduces or allow futher reduction on $\varepsilon^k_{N^k}$, without increasing the team makespan. Such addition results in removal of blockage (idle time removed or reduced) by the robot k' in the schedule.

Pseudo-code for the `BlockRemoval` algorithm is presented in Algorithm 2. In the algorithm, we start by finding the blocking task in the schedule with the largest start time. This is because for most of time, there is no idle time between blocking task with the latest start time and the makespan robot's last teleoperated edge, and blocking can be resolved efficiently. We then try to add a task from the blocking robot's mission to the schedule such that it reduces the start time of the blocking task. If such an addition is possible, we return the updated schedule, else we discard this task and move to the blocking task with next largest starting time, until we reach the beginning of the schedule.

Algorithm 2 `BlockRemoval`

Input: $\mathcal{P}$, $\mathcal{S}$
Output: $\mathcal{S}'$

1: Initialize: $\mathcal{S}' = \mathcal{S}$
2: $\overline{s} \leftarrow$ blocking task with largest starting time
3: Find task e' that reduces the start time of $\overline{s}$
4: **if** e' exists **then**
5: **return** Updated schedule $\mathcal{S}'$
6: **else**
7: Go to line 2 and repeat for blocking task with next largest start time until no more blocking tasks are present
8: **return** $\mathcal{S}'$

An example is shown in Fig. 3 to illustrate this operation. Given the schedule generated in Fig. 2, further Greedy Insertion is not possible. Adding task of e^3_1 to the schedule does not reduce makespan because the task final task of Robot3, e^3_2, will have to wait until the operator finishes the

[3]Depending on the application, it may be useful to set a threshold $\epsilon \in \mathbb{R}_{>0}$ on the idle time between s_j and s_{j+1} to consider s_{j+1} as a blocking task. For example, we can set $\epsilon = min\{\beta^k_j\}$. In this case, if there is an idle time less than the minimum teleoperation time, inserting any task here only delay's the execution of later tasks in the schedule. Thus, such an idle time cannot help improve the makespan and we should skip it

task e_2^1 (the blocking task). Instead, if we add e_1^1 to $\mathcal{S}$, it reduces the makespan by reducing the start time of the blocking task e_2^1.

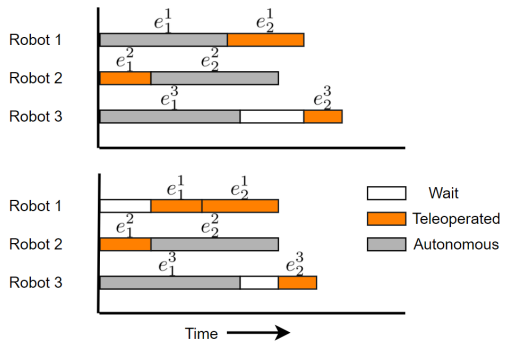

Fig. 3: Example for Block Removal. Makespan Robot 3's mission time is reduced indirectly by teleoperating e_1^1.

C. Iterative Greedy

Starting with an empty teleoperation schedule, the Iterative Greedy algorithm first generates an intermediate schedule using Alg. 1. Using this schedule, we try the Block Removal routine using Alg. 2. The schedule is iteratively improved by applying Alg. 1 and Alg. 2 one after the other, until both of these algorithms stop to make improvements in a given schedule $\mathcal{S}$, which is then selected as the final output.

Algorithm 3 `IterativeGreedy`

Input: $\mathcal{P}$
Output: $\mathcal{S}$
Initialization : $\mathcal{S} = [\ ]$, $done = 0$.
while not $done$ **do**
  $\mathcal{S}' \leftarrow$ Greedy Insertion($\mathcal{P}$, $\mathcal{S}$)
  if $\mathcal{S}' = \mathcal{S}$ **then**
    $\mathcal{S}' \leftarrow$ Block Removal($\mathcal{P}$, $\mathcal{S}$)
    if $\mathcal{S}' = \mathcal{S}$ **then**
      $done = 1$
  $\mathcal{S} \leftarrow \mathcal{S}'$
return $\mathcal{S}$

Runtime of `IterativeGreedy`: Letting $\bar{N} := \sum_{k=1}^{K} N^k$, each iteration of `GreedyInsertion` can be implemented to run in $O(\bar{N})$ time. Similarly, each iteration of `BlockRemoval` runs in $O(\bar{N})$ time. Since at most $\bar{N}$ tasks can be added to the schedule, the overall runtime of `IterativeGreedy` is bounded by $O(\bar{N}^2)$.

V. Evaluation

In this section, we present performance results for a simulated multi-robot scheduling problem under the following methods (described in Section V-A): 1) Optimal schedule (solution of the MILP formulation), 2) Iterative Greedy, 3) Greedy Insertion, 4) Comparison Greedy, and 5) Naïve Greedy. The problem and the solution frameworks for all algorithms were implemented using Python. The Gurobi Python API is used for the MILP solution.

To generate an instance, for each task of each robot, two numbers are sampled from a uniform random distribution and are rounded to 2 decimal places. One is used as the task working time under teleoperation β_j^k, and the sum of two is used as the autonomous time α_j^k:

$$\beta_j^k \sim U[10, 20],\ \ \Delta\tau_j^k \sim U[0, 10],$$
$$\alpha_j^j \leftarrow \beta_j^k + \Delta\tau_j^k. \tag{7}$$

A. Baseline Algorithms

We consider the following baseline solution methods to assess the performance of the Iterative Greedy algorithm.

MILP Solution: The MILP formulation in Section III is implemented and solved with Python Gurobi API. Solving the formulation directly gives us x_j^k and τ_j^k for each task.

Naïve Greedy: Under this algorithm, the operator is simply scheduled to teleoperate the next available task of the makespan robot. If the makespan robot is still executing a task, the operator waits for the robot.

Comparison Greedy: We have also developed the Comparison Greedy algorithm, which compares between alternatives given an intermediate schedule. We compute the finish time of the last task in the current schedule, and determine the task e_j^k that the makespan robot will be executing at that time. We then pick the better of the two alternatives: 1) Adding e_j^k to the schedule, and have the makespan robot wait for the operator at start of e_j^k, or 2) Adding e_{j+1}^k to the schedule and have the operator wait for the makespan robot to complete e_j^k.

Greedy Insertion: To assess the improvement brought by the Block Removal step, we compare the schedule generated by only Greedy Insertion defined in Algorithm 1.

B. Scalability Test

We begin the evaluations by looking at the computation time of MILP and Iterative Greedy on different problem sizes (number of robots and tasks in their missions), as specified in Table I. The computation times shown in the table are the average of 100 instances for each case. Along both dimensions of the problem size, the number of robots and number of tasks, the computation time of MILP increases at a very high rate. Computation time of Iterative Greedy algorithm remains below 0.01 seconds for all test cases in Table I. Even for larger instances, where MILP solution is unavailable, the computation time of Iterative Greedy algorithm grows at a much slower rate. For example, for a problem instance with 4 robots and 40 tasks each, its average computation time is 5.22 seconds. For reference, the simulations were run on a laptop computer with 4 core, 2.1 GHz processor and 16 GB RAM.

C. Comparison with the Optimal Schedule

The Iterative Greedy algorithm is compared against the optimal schedule to validate its applicability for our problem. The optimal schedule using MILP formulation cannot be computed for larger problem instances, due to its poor scalability, therefore this test is limited to small-sized problems. The relative performance (ratio of the makespan under

TABLE I: CPU Time of MILP and Iterative Greedy (in seconds)

	$K=2$ $N^k=11$	$K=3$ $N^k=5$	$K=3$ $N^k=8$	$K=3$ $N^k=11$	$K=4$ $N^k=11$
MILP	0.30	0.4	0.80	10.16	109.22
Iterative Greedy	<0.01	<0.01	<0.01	<0.01	<0.01

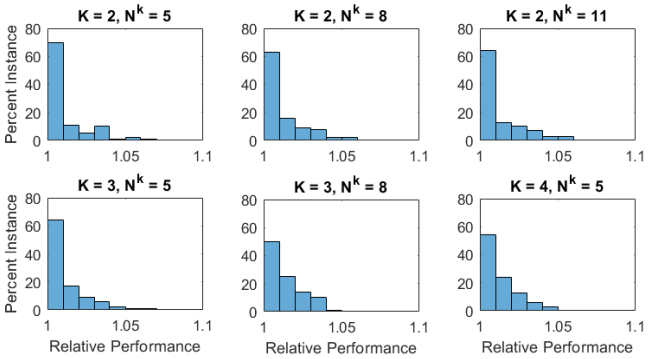

Fig. 4: Relative performance of the Iterative Greedy methods compared to the optimal solution for number of robots $K \in \{2,3,4\}$, and number of tasks $N^k \in \{5,8,11\}$ for all robots. Each plot shows the distribution of 100 instances based on their relative performance (ratio of makespan under Iterative Greedy method to the optimal makespan).

Iterative Greedy algorithm to the optimal schedule) is shown in Fig. 4. For each size, 100 instances were generated using the random instance generation mentioned earlier.

We observe that the performance of the Iterative Greedy algorithm is comparable to that of optimal schedule. As the number of robots increases, the distribution of relative performance slowly shifts away from 1. However, the makespan under the Iterative Greedy algorithm is still within 5% of the optimal schedule for over 90% of the instances under all test cases. For reference, the team makespan without teleoperation is, on average, 20.73% more than the optimal for these test cases.

D. Comparison with other Greedy Algorithms

Next, we compare the performance of the Iterative Greedy algorithm with the Greedy Insertion, Comparison Greedy and Naïve Greedy algorithms on larger problem instances. Note that it is also possible to combine the Iterative Greedy algorithm with any of these greedy algorithms. We include performance results from such combinations to demonstrate its effects on greedily-generated schedules. For the comparison, under each test condition (given number of robots and tasks in their missions), 100 problem instances are created in a similar way as before. Fig. 5 shows performance comparison of the different algorithms. Iterative Greedy has the best performance among all algorithms, and we observe 6 to 10% improvement over the baseline Naïve Greedy algorithm for small to moderate problem size. We observe that, as the number of robots increases, the difference between the performance of all algorithms start to diminish. This supports

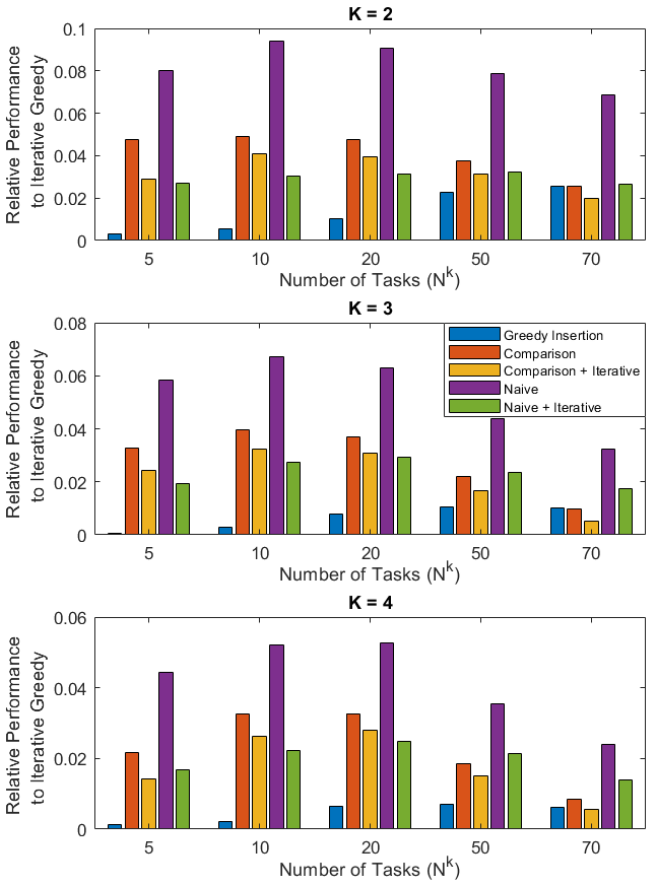

Fig. 5: Performance comparison of baseline greedy solution techniques relative to the proposed Iterative Greedy algorithm. The plots show relative performance of different techniques for up to 4 robots and 70 tasks each.

the intuition that as number of robots increases, the human operator is required to distribute their time to more and more robots, thus decreasing their effectiveness. From the plots, we also observe the effectiveness of the Iterative Greedy in improving relative performance when applied in combination with Naïve Greedy and Comparison Greedy. This indicates that the Iterative Greedy technique can be used to further improve any greedily-generated schedule.

E. Example Problem Instance

Fig. 6 shows an example instance of the scheduling problem with three robots. First, the Greedy Insertion algorithm generates a schedule that reduces makespan but contains long idle time in operator's schedule. Then the Block Removal algorithm removes these gaps and results in a schedule with very little idle time. The MILP solution shows that a better performing schedule is possible even with a greater idle time.

VI. Conclusions and Discussions

In this paper, we present a problem of scheduling a human operator to a team of multiple robots, such that the team makespan is minimized. We show that this problem is NP-Hard and develop the Iterative Greedy algorithm that cycles through two sub-routines: Greedy Insertion and Block

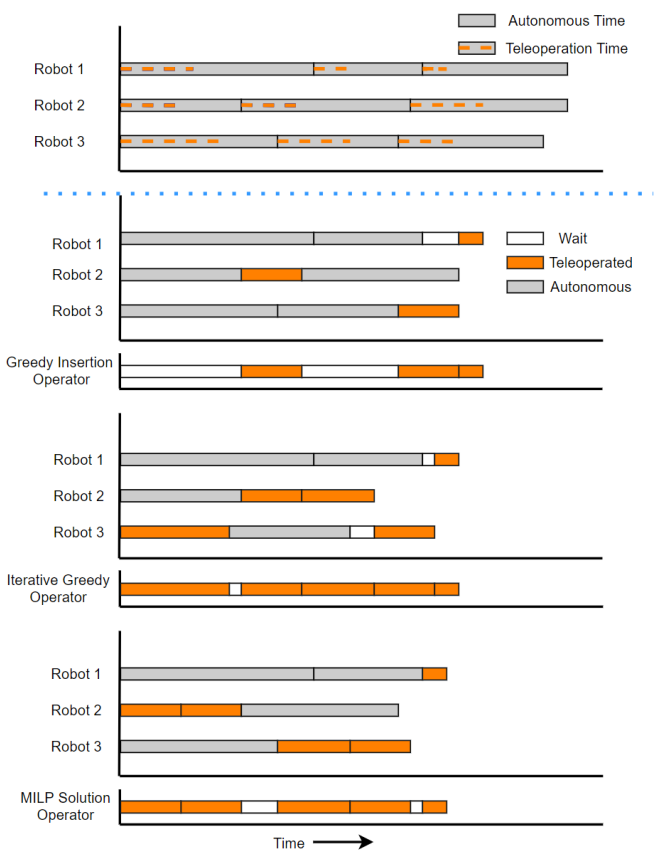

Fig. 6: Scheduling of Iterative Greedy, Greedy Insertion and MILP Solution on a Multi-robot Mission Instance.

Removal. This algorithm generates a greedy schedule in each iteration, and improves it by removing *blockages* when needed. The algorithm scales well with problem size and produce smaller makespan than other greedy solution techniques. It is also shown that the Iterative Greedy algorithm can be applied to any greedily-generated schedule to further improve the performance. For future research, our goal is to further develop the model by allowing imperfect information and possibility of mission re-planning for the robots. The solution technique will also benefit from the ability to adapt the current schedule online based on new observations.

REFERENCES

[1] A. Rosenfeld, N. Agmon, O. Maksimov, and S. Kraus, "Intelligent agent supporting human–multi-robot team collaboration," *Artificial Intelligence*, vol. 252, pp. 211–231, 2017.

[2] A. Khasawneh, H. Rogers, J. Bertrand, K. C. Madathil, and A. Gramopadhye, "Human adaptation to latency in teleoperated multi-robot human-agent search and rescue teams," *Automation in Construction*, vol. 99, pp. 265–277, 2019.

[3] K. Zheng, D. F. Glas, T. Kanda, H. Ishiguro, and N. Hagita, "Supervisory control of multiple social robots for navigation," in *ACM/IEEE Int. Conf. on Human-Robot Interaction*, 2013, pp. 17–24.

[4] J. Y. Chen and M. J. Barnes, "Supervisory control of multiple robots: Effects of imperfect automation and individual differences," *Human Factors*, vol. 54, no. 2, pp. 157–174, 2012.

[5] S. Y. Chien, M. Lewis, S. Mehrotra, and K. Sycara, "Imperfect automation in scheduling operator attention on control of multi-robots," in *Human Factors and Ergonomics Society Annual Meeting*, vol. 57, no. 1, 2013, pp. 1169–1173.

[6] A. Dahiya, N. Akbarzadeh, A. Mahajan, and S. L. Smith, "Scalable operator allocation for multi-robot assistance: A restless bandit approach," *IEEE Transactions on Control of Network Systems*, 2022, To Appear.

[7] G. Swamy, S. Reddy, S. Levine, and A. D. Dragan, "Scaled autonomy: Enabling human operators to control robot fleets," in *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2020, pp. 5942–5948.

[8] Q. Ren, K. L. Man, E. G. Lim, J. Lee, and K. K. Kim, "Cooperation of multi robots for disaster rescue," in *IEEE International SoC Design Conference (ISOCC)*, 2017, pp. 133–134.

[9] Y. Huang, Y. Zhang, and H. Xiao, "Multi-robot system task allocation mechanism for smart factory," in *IEEE International Information Technology and Artificial Intelligence Conference*, 2019, pp. 587–591.

[10] P. Benavidez, M. Kumar, S. Agaian, and M. Jamshidi, "Design of a home multi-robot system for the elderly and disabled," in *System of Systems Engineering Conference*, 2015, pp. 392–397.

[11] C. Y. Wong and G. Seet, "Workload, awareness and automation in multiple-robot supervision," *International Journal of Advanced Robotic Systems*, vol. 14, no. 3, 2017.

[12] J. M. Riley and M. R. Endsley, "Situation awareness in HRI with collaborating remotely piloted vehicles," in *Human Factors and Ergonomics Society Annual Meeting*, vol. 49, no. 3, 2005, pp. 407–411.

[13] S.-Y. Chien, M. Lewis, S. Mehrotra, N. Brooks, and K. Sycara, "Scheduling operator attention for multi-robot control," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2012, pp. 473–479.

[14] J. Y. Chen and M. J. Barnes, "Human–agent teaming for multirobot control: A review of human factors issues," *IEEE Transactions on Human-Machine Systems*, vol. 44, no. 1, pp. 13–29, 2014.

[15] S. K. K. Hari, A. Nayak, and S. Rathinam, "An approximation algorithm for a task allocation, sequencing and scheduling problem involving a human-robot team," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 2146–2153, 2020.

[16] S. Musić and S. Hirche, "Control sharing in human-robot team interaction," *Annual Reviews in Control*, vol. 44, pp. 342–354, 2017.

[17] M. B. Dias, B. Kannan, B. Browning, E. Jones, B. Argall, M. F. Dias, M. Zinck, M. Veloso, and A. Stentz, "Sliding autonomy for peer-to-peer human-robot teams," in *International Conference on Intelligent Autonomous Systems*, 2008, pp. 332–341.

[18] D. Szafir, B. Mutlu, and T. Fong, "Designing planning and control interfaces to support user collaboration with flying robots," *The International Journal of Robotics Research*, vol. 36, no. 5-7, pp. 514–542, 2017.

[19] E. A. Kirchner, S. K. Kim, M. Tabie, H. Wöhrle, M. Maurus, and F. Kirchner, "An intelligent man-machine interface—multi-robot control adapted for task engagement based on single-trial detectability of p300," *Frontiers in Human Neuroscience*, vol. 10, p. 291, 2016.

[20] P. Gupta and V. Srivastava, "Optimal fidelity selection for human-in-the-loop queues using semi-markov decision processes," in *American Control Conference*, 2019, pp. 5266–5271.

[21] J. Roslöf, I. Harjunkoski, T. Westerlund, and J. Isaksson, "Solving a large-scale industrial scheduling problem using milp combined with a heuristic procedure," *European Journal of Operational Research*, vol. 138, no. 1, pp. 29–42, 2002.

[22] A. Bischi, L. Taccari, E. Martelli, E. Amaldi, G. Manzolini, P. Silva, S. Campanari, and E. Macchi, "A rolling-horizon optimization algorithm for the long term operational scheduling of cogeneration systems," *Energy*, vol. 184, pp. 73–90, 2019.

[23] S. A. Zanlongo, P. Dirksmeier, P. Long, T. Padir, and L. Bobadilla, "Scheduling and path-planning for operator oversight of multiple robots," *Robotics*, vol. 10, no. 2, p. 57, 2021.

[24] S. Mau and J. M. Dolan, "Scheduling to minimize downtime in human-multirobot supervisory control," 2006.

[25] R. Yoshinaka, "Higher-order matching in the linear lambda calculus in the absence of constants is np-complete," in *International Conference on Rewriting Techniques and Applications*. Springer, 2005, pp. 235–249.

[26] G. G. Brown and R. F. Dell, "Formulating integer linear programs: A rogues' gallery," *INFORMS Transactions on Education*, vol. 7, no. 2, pp. 153–159, 2007.