Online Scheduling of Operator Assistance for Multi-Robot Teams with Uncertain Robot Capabilities and Environments

by

Yifan Cai

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2022

Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contributions

I initialized the study presented in the thesis and developed the solution for the deterministic question presented in Chapter 2. Abhinav Dahiya joined for the simulation part in Section 2.7. The work in Chapter 3 was down collaboratively by both of us. Content shown in Chapter 2 except Section2.2 and 2.3 were submitted and accepted by IEEE Conference on Design and Control (CDC) 2022 as a paper titled *"Scheduling Operator Assistance for Shared Autonomy in Multi-Robot Teams"* which is co-authored by Yifan Cai, Abhinav Dahiya, Nils Wilde and Stephen L. Smith (not published yet).

Abstract

In this study, we consider the problem of allocating human operator assistance in a system with multiple autonomous robots. Each robot is assigned with an independent mission, 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 are interested in finding a schedule of teleoperated tasks in order to minimize the system makespan. Makespan is the time elapse from the initial time to the time that all robots finish their missions. Both deterministic and stochastic models of robot task completion times are considered in this study. We first show that the deterministic problem of finding the optimal teleoperation schedule is NP-Hard. We then formulate the problem as a Mixed Integer Linear Program, which can be used to optimally solve small to moderate-sized instances. We also develop an anytime algorithm that makes use of the system structure to provide a fast and high-quality solution of the operator scheduling problem, even for larger instances. Our key insight in this algorithm 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 common solution techniques. Expanding research to the stochastic setting, where task duration is random variable to represent uncertainty of robot capabilities and environments, we developed a parameterized replanning policy. This policy selectively chooses to update the schedule based on task observations. The parameter can be used to control the trade-off between performance and efficiency. The resulting policy demonstrates good planning competence in both average and worst cases. Results also show significant reduction in resource requirements for replanning, with little to no compromise in performance, when compared to the policy that replans on every task completion.

Acknowledgements

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.

Table of Contents

List of Figures

List of Tables

Chapter 1

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 [37], [24], [50], [13]. As it is challenging for human operators to supervise and assist a large number of robots on their own [6], [10], a number of studies in the literature propose effective decision support systems (DSS) to aid the human operator(s) in providing assistance [14], [41], [37], and operation management to optimally allocate tasks in human robot teams [22].

In this thesis, 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 1.1 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. Our model may also apply to manufacturing assembly line and a task can be considered as a step of the assembly line, such as tightening the screw or mounting a part. Robots in the team can be either homogeneous or heterogeneous. 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. In the deterministic case, task completion time is fixed value. However, in the stochastic case, to represent robot capability to handle uncertain environment, task completion times are characterized using random variables. 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. For the stochastic system, in addition to find such a teleoperation schedule, the DSS also needs to identify deviation of task execution from the schedule and provides schedule update accordingly. The problem objective is to find a teleoperation schedule and schedule update policy (in stochastic case) for both the human operator and robots that minimizes the time taken until all robot missions are complete.

This thesis documents the following contributions:

1. We show that the operator scheduling problem for multiple robots with deterministic task duration is NP-Hard using a reduction from a variant of the Satisfiability problem called *2p1n-3SAT* problem in Section 2.2.

2. We formulate a Mixed Integer Linear Program (MILP) that can be used to generate optimal schedules for the given deterministic problem in Section 2.4. An extension to a multiple-operator version of the problem is also developed and documented in Subsection 2.4.1.

3. We present an anytime algorithm that iteratively generates teleoperation schedules for the given problem in Section 2.5. The algorithm is capable of solving much larger instances of the given problem than the MILP formulation.

4. The algorithm is evaluate using numerical simulations in Section 2.7. The results show that our method provides an efficient and scalable solution compared to other approaches.

5. We develop an online parameterized policy that identify the need and suitable method for schedule update in Section 3.2. The policy guides the team in stochastic environment with efficient replanning method and relatively less times of replanning.

6. The online policy is evluated using numerical simulations in Section 3.4. It improves the team efficiency in both average and worse case scenarios.

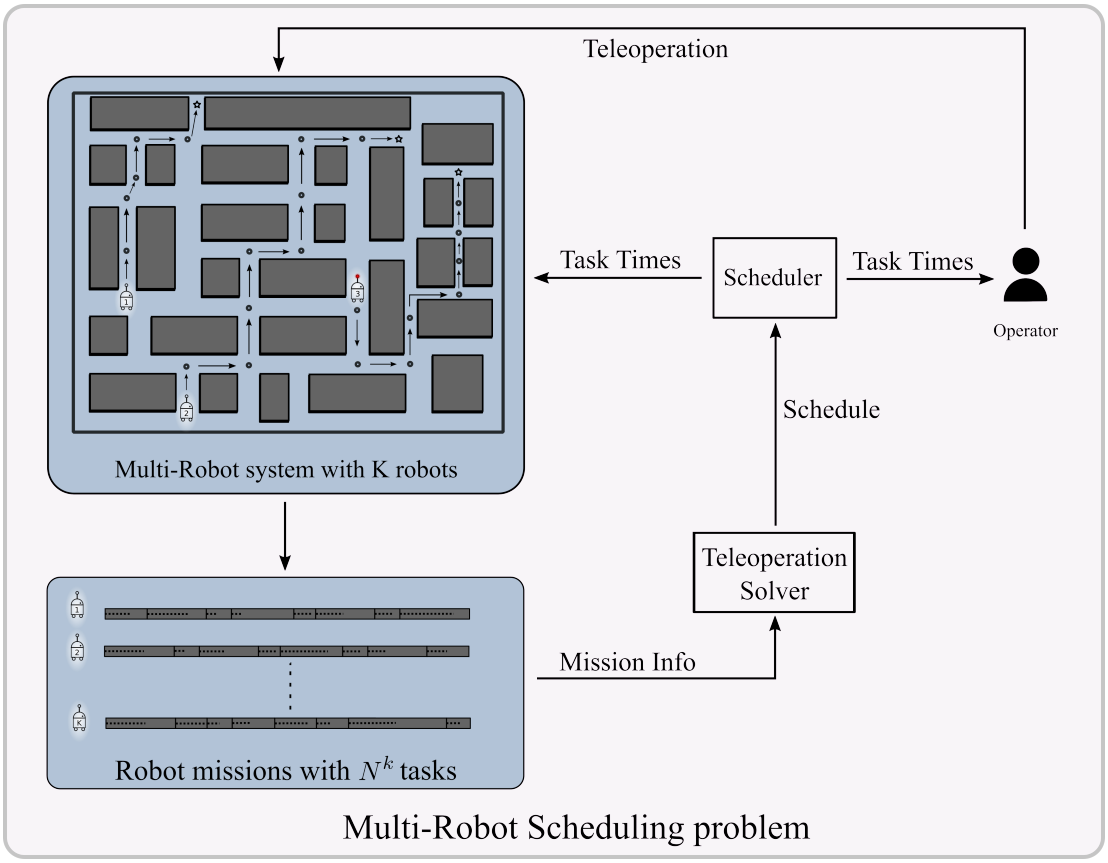

Figure 1.1: Information flow in the multi-robot teleoperation scheduling system with K robots. A robot k are assigned with an independent series of tasks. Given mission information, solver computes the schedule, which is then converted to information about task timing for both operator and robots. The operator assists on tasks assigned by the schedule.

1.1 Related Work

Human-multi-robot teams have found their application in search-and-rescue [34], [5], [44], smart factory operation [23], home care for seniors [2], and package delivery [14]. Robots possess the characteristics of tirelessness and high physical powers, while human workers are intelligent, flexible, and able to find their ways out of unfamiliar and complicated situations [22], [11]. Thus, human robot collaboration [22] and human supervision on robot mission [28] are studied in recent researches. Objectives of those human-robot-team studies include improving mission time, work quality, task allocation, and human

preference [28], [33], [47]. In many recent studies for human robot collaboration, human and robots can work on tasks independently or collaboratively in a shared space [20], [26], [8]. Human supervision and assistant to guarantee the work quality or handle critical tasks was also presented in [28], [48], [15]. However, such a team composition also brings the risk of increasing operator workload and decreasing in their situational awareness [45], [36]. In [9], 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 [7], [37], [12]. In addition, update on task allocation based on observation and prediction model is able to provide human robot team with good guidance and reduce the negative effect on efficiency and work quality brought by environment and agent capability uncertainty [39], [28], [49].

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 [14], or a pre-determined offline schedule, like in [21]. 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 [31], [16]. In [15], robots generate task and motion plan options with probability of delay and use human intervene to improve the plan to prevent delay and work overload for robots. Research on effective interaction interfaces also aims to facilitate human supervision of robot teams [42], [25]. For task allocation in human robot collaboration system, various constraints and objectives are considered to achieve synergy. In [20], a problem of human and robots working in a shared space with temporal and spatial constraints is optimized to achieve less completion time. Ergonomics are also consider as an objective in [32] when incorporating robots team to existing pure human manufacturing process. Work, presented in [29], assesses difficulty of task associated with object features and task procedures during task distribution among human and robots. 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 [19]. 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 dif-

ferent tasks to minimize performance metrics like makespan, idle time and etc [35]. A common way of solving the scheduling problem is through the linear programming and constraint programming [20], [43], [27], [17], 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 [38] 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 [3]. In contrast to the problem considered in [3], our problem is highly-coupled over time, and thus there aren't natural breakpoints in time to decompose the problem. In [1], aggregated models are used to schedule distributed energy resource, such that the computation complexity caused by uncertainty is reduced. Authors of [18] presents a novel approach, in which a model-free pair ranking heuristic mimicking apprenticeship that learns scheduling policy from human or expert demonstration is studies and developed.

Research work about task allocation also considers uncertainty from many sources and provides strategies to improve the team performance. For example, uncertainty introduced by human is modelled or predicted for task planner design in [33], [26], [8], [49]. Differently, in [15], robot workload is the source of uncertainty. Also, in [40], uncertainty is represented by contingency tasks raised by dynamic environment. In our study, uncertainty caused by unsure robot capability in uncertain environments is considered for task duration. Strategies of handling uncertainty including encoding the problem into a Markov Decision Process and using probabilistic solver [43], and take measurement on agents to assess their capability and attentively schedule the team [49].

Two closely related works to our problem are presented in [48], [21]. 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.

Work presented in [28] studies a similar problem to ours. In that literature, researchers study task allocation in human multirobot collaborative environment with possible use of human supervisions for work quality purpose. The combined objective of work quality, workload and completion time is optimized in their paper. A MILP formulation is developed to model the problem and give optimal nominal solution. Online reallocation happens when system observation shows a need, and the reallocation is done applying MILP with nominal and updated values. Our work develops an anytime offline algorithm that is ver-

ified to be time efficient. The online schedule update policy presented in this thesis also accesses the necessity of replan, but more than one method of replan is used in the policy and will be picked based on the team state.

Chapter 2

Deterministic Scheduling Problem

2.1 Problem Statement

In Chapter 2, we consider a deterministic 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]. α^k_j and β^k_j are fixed values for a given task e^k_j.

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 the index $\{k, j\}$ of some task e^k_j for $k \in \{1, \ldots, K\}, j \in \{1, \ldots, N^k\}$ that

[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.

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 $s_i = e_j^k$, 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_j^k. 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* [30], denoted as $\mu(\mathcal{S}, \mathcal{P}) \in \mathbb{R}_{>0}$.

We impose the following assumptions on the problem:

(A1) The operator can teleoperate 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.

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

To begin, we establish that this problem is NP-Hard.

2.2 Hardness Proof: Reduction from 2p1n-3SAT

To prove Problem 1 is NP-hard, we introduce an NP-complete variant of Satisfiability called *2p1n-3SAT* [46]. In *3SAT* problems, we are given a Boolean formula as a conjunction of several clauses where each clause is a disjunction of exactly 3 literals. A literal is either a variable or its negation. In *2p1n-3SAT*, each variable shows up exactly twice, and its negation shows up exactly once [46].

The *2p1n-3SAT* problem and the decision version of the scheduling Problem 1 are as follows:

Problem 2 (*2p1n-3SAT*)**.** Given a Boolean expression in the *2p1n* format with K clauses with v variables, does there exist an assignment to the variables that makes the formula evaluate to true?

Problem 3 (*Possible-Makespan*)**.** Given K robots, the missions $\{p^1, \ldots, p^K\}$ for each robot, and the autonomous and teleoperation task completion times α and β [2] for every task of each robot, and a target time $\overline{\mu} \in \mathbb{R}_{>0}$, does there exist a schedule $\mathcal{S}$ that results in a makespan $\mu(\mathcal{S}, \mathcal{P}) \leq \overline{\mu}$

Proposition 1. Problem 3 (*Possible-Makespan*) is NP-Complete.

Proof. We begin the proof by proposing a reduction from Problem 2 to Problem 3:

Reduction: To reduce an instance ϕ of Problem 2 into an instance ψ of Problem 3, we replace the literals in the Boolean formula with tasks in robots missions. Each clause in the formula corresponds to a robot in the scheduling problem. Fixing the order of variables arbitrarily, we create robot a mission from literals in a clause using the following four rules:

1. If a literal is the first positive appearance of a variable in the Boolean formula ϕ, we add two consecutive tasks in robot's mission. The first task has autonomous completion time $\alpha = z$ and teleoperation time $\beta = z - \delta z$, for some $z, \delta z \in \mathbb{R}_{>0}$ and $0 < \delta z \ll z$ (e.g., $z = 100, \delta z = 1$). For the second task, $\alpha = \beta = z$.

2. If a literal is the second positive appearance of a variable in ϕ, we again add the same two tasks as the first case, but in reverse order.

3. If a literal is the negative appearance of a variable, we add a single task in robot's mission with $\alpha = 2z$ and $\beta = 2z - \delta z$.

4. For each variable that is not present in the clause (when there are more than three variables), we add a single task in robot's mission with $\alpha = \beta = 2z$.

With this reduction, an instance of Problem 2 with v variables is converted to an instance of Problem 3 with $\overline{\mu} = 2zv$. An example reduction is illustrated in Fig. 2.1.

Under the reduction given above, the generation of ψ from ϕ takes polynomial time as one needs to parse each literal in every clause only once, and generate tasks in robots missions (a constant time operation) for each of those literals.

[2] Here, we omit subscript k and j from expressions of α and β for ease of notation.

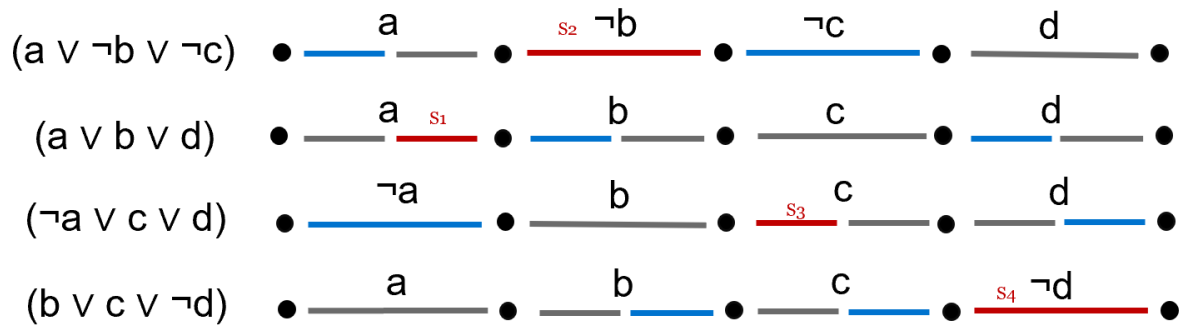

Figure 2.1: Converting *2p1n-3SAT* formula ϕ with four clauses to missions of four robots. Each mission is shown as a sequence of differently-colored tasks. Blue and red tasks have a difference of δz between their α and β, and grey tasks have no difference between their α and β.

Trivially, *Possible-Makespan* is in NP. Given a schedule $\mathcal{S}$, it takes polynomial time to verify the if makespan is less than $\overline{\mu}$, by computing the makespan with the solution $\mathcal{S}$ and team information $\mathcal{P}$.

Therefore, the NP-Completeness of Problem 3 follows directly from Lemmas 1 and 2. □

Lemma 1. Under the reduction given in Proposition 1, if ϕ is a true instance of Problem 2, then ψ is a true instance of Problem 3.

Proof. For the purpose of the proof, we call the tasks in ψ where $\alpha > \beta$ as *effective* tasks. In order to have ϕ to be true, each clause must have at least one true literal by definition. The reduction is designed in a way that effective tasks corresponding to positive variables in different clauses do not overlap with each other. Therefore, if ϕ is a true instance of Problem 2, then it is possible to teleoperate the effective tasks corresponding to at least one true literal in each clause.

This will result in a reduction of at least δz in the travelling time of each robot, so the makespan is less than $2zv$. For example, Fig. 2.1 shows a true instance of the problem, and the schedule marked as red-colored tasks $\langle s_1, s_2, s_3, s_4 \rangle$ results in a makespan $\mu(\mathcal{S}, \mathcal{P}) = 2zv - \delta z$. □

Lemma 2. Under the reduction given in Proposition 1, if ψ is a true instance of Problem 3, then ϕ is a true instance of Problem 2.

Proof. Since every robot in ψ has the same autonomous task completion time, each of them must have at least one effective task teleoperated to have a makespan less than

$2zv$. The tasks are arranged in a way that a variable and its negation's effective task cannot both present in a satisfying schedule to result in a $\mu < 2zv$.(due to overlap in their times). Therefore, we can choose one teleoperated task from each robot mission, and set its corresponding literal to be true. This will result in an assignment for which ϕ is true. □

Since the decision problem *Possible-Makespan* is in NP-Complete, the problem of finding the optimal teleoperation schedule in Problem 1 is NP-Hard. Since all constraints in our problem are linear time constraints, we formulate our problem as a mixed integer linear program (MILP), which has been used to formulate a wide range of NP-Hard problems.

2.3 Compute Makespan

Given the deterministic problem set-up and schedule $\mathcal{S} = \langle s_1, \ldots, s_n \rangle$, we will show how to compute the makespan of the team. Each element in $\mathcal{S}$ is a task index, which records the corresponding robot and task number. The operator teleoperates tasks in $\mathcal{S}$ one by one. Robots run tasks not in $\mathcal{S}$ autonomously, but when a robot reaches the start of a task in $\mathcal{S}$, it will be teleoperated, or wait for the operator if the operator is still busy with a previous task in $\mathcal{S}$. Similarly, if the operator reaches the start of a task in $\mathcal{S}$, but the corresponding robot is still busy with its earlier task, the operator will wait for the robot. For all robots, we initialize a recording variable `lastVisitedTask`, $\mathcal{L} := \{l^1, \ldots, l^K\} = \mathbf{0}$, and `robotCompletionTime`, $\mathcal{T} := \{T^1, \ldots, T^K\} = \mathbf{0}$. Also, we initialize a variable `operatorWorkTime`, $T^{\text{opt}} = 0$, to track the operator work time.

The algorithm goes through the schedule vector one by one. For example, at time of T^{opt}(which is initially 0), with $s_1 = \{k, j\}$, we find the time needed for robot k to complete tasks in between robot k's `lastVisitedTask` and s_1 autonomously (there may be 0 task in between), recorded as t_{bwt}, and add t_{bwt} to T^k. Then, compare the magnitude of T^k and T^{opt}. If $T^k \geq T^{\text{opt}}$, robot k will be ready for s_1 later than or at the same time as the operator, so $T^k = T^k + \beta_j^k$. However, if $T^k < T^{\text{opt}}$, operator will be ready for the s_1 later than robot k, so $T^k = T^{\text{opt}} + \beta_j^k$. Lastly, we update the operator working time to s_1 completion time, $T^{\text{opt}} = T^k$, and robot k's `lastVisitedTask`, $l^k = j$. Then, we proceed to next task in $\mathcal{S}$, with time equals to the updated T^{opt}. After, we loop through all tasks in S, we check if there is any autonomous tasks left for each robot, and add time needed for those tasks to each T^k. A pseudo code for this procedure is presented in Algorithm 1. According to the definition of makespan, $\mu = \texttt{max}(\mathcal{T}[k])$ for $k \in \mathcal{K}$. Please note $\mathcal{P}$ contains task information (task sequence, task duration under autonomous and

teleoperation modes) for K robots. S is the schedule containing index for tasks with the order of teleoperation.

Algorithm 1 `Robot Completion Time`

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

Initialize $\mathcal{L} = \mathbf{0}$, $\mathcal{T} = \mathbf{0}$, $T^{\text{opt}} = 0$
for $i \in \langle 1, \ldots, n \rangle$ **do**
  $\{k, j\} \leftarrow s_i$
  for $h \in \langle \mathcal{L}[k] + 1, \ldots, j - 1 \rangle$ **do**
    $\mathcal{T}[k] \leftarrow \mathcal{T}[k] + \alpha_h^k$
  $\mathcal{T}[k] \leftarrow \texttt{max}(\mathcal{T}[k], T^{\text{opt}}) + \beta_j^k$
  $T^{\text{opt}} \leftarrow \mathcal{T}[k]$, $\mathcal{L}[k] \leftarrow j$
for $k \in \mathcal{K}$ **do**
  for $j \in \langle \mathcal{L}[k] + 1, \ldots, N^k \rangle$ **do**
    $\mathcal{T}[k] \leftarrow \mathcal{T}[k] + \alpha_j^k$
return $\mathcal{T}$

2.4 MILP Formulation

Since the Problem 1 is proved to be NP-Hard, we first encode it as a MILP. In the MILP formulation, our objective is to find a schedule $\mathcal{S}$ that minimizes team makespan $\mu(\mathcal{S}, \mathcal{P})$, 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.,

$$\varepsilon_j^k = \tau_j^k + (1 - x_j^k)\, \alpha_j^k + x_j^k\, \beta_j^k. \tag{2.1}$$

A MILP can then be formulated as follows:

$$\begin{aligned}
\text{Minimize:} \quad & \bar{\mu} \\
\text{Subject to:} \quad & \bar{\mu} \geq \varepsilon^k_{N^k} \qquad \forall\, k \in \mathcal{K}, && (2.2)\\
& \tau^k_1 \geq 0, \qquad \forall\, k \in \mathcal{K}, && (2.3)\\
& \tau^k_j \geq \varepsilon^k_{j-1}, \ \forall\, k \in \mathcal{K},\, j \in \{2, \ldots, N^k\}, && (2.4)\\
& x^k_j + x^l_i = 2 \implies \tau^k_j \geq \varepsilon^l_i \ \text{ or } \ \tau^l_i \geq \varepsilon^k_j, \\
& \qquad\qquad \forall\, k, l \in \mathcal{K};\, k \neq l, \\
& \qquad\qquad \forall\, j \in \{1, \ldots, N^k\}, \\
& \qquad\qquad \forall\, i \in \{1, \ldots, N^l\}, && (2.5)\\
& x^k_j \in \{0, 1\}, \forall\, k \in \mathcal{K},\, j \in \{1, \ldots, N^k\}. && (2.6)
\end{aligned}$$

Constraint (2.2) restricts the time needed for every robot to complete its mission to be not more than the objective $\bar{\mu}$. Constraint (2.3) sets the earliest start time for the robots. This constraint is not critical to the problem setup, but it gives a time reference to the program.

Constraint (2.4) ensures that the j^{th} task of a robot mission can only starts after the $(j-1)^{\text{th}}$ task is completed. Constraint (2.6) restricts the variables x^k_j to be a binary variable. Constraint (2.5) specifies no two tasks can be teleoperated with an overlapping time interval. Note that Constraint (2.5) 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 [4]. In order to convert Constraint (2.5) to Python Gurobi API encodable expressions, we need three auxiliary binary variables: C, D, E, for each pair of e^k_j and e^l_i, and the constraint is decomposed into:

$$\begin{aligned}
C &= x^k_j \wedge x^l_i, \\
C = 1 &\implies D + E = 1, \\
D = 1 &\implies \tau^k_j \geq \varepsilon^l_i, \\
E = 1 &\implies \tau^l_i \geq \varepsilon^k_j.
\end{aligned}$$

$\wedge$ in the above expression is the logic AND. $C = 1$ only when both x_j^k and x_i^l are 1, so it means the same as $x_j^k + x_i^l = 2$.

For instance, the implication $D = 1 \implies \tau_j^k \geq \varepsilon_i^l$ can be represented using two linear conditions $D\tau_j^k - D\varepsilon_i^l \geq 0$; $D \in \{0, 1\}$. The above implications are now written as indicator constraints, which are supported directly by Gurobi.

Note: To implement constraint (2.5), 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.

2.4.1 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 $m \in \mathcal{M}$. Though the operators are identical, we still need to index them to make sure teleoperated tasks with time overlapping are assigned to different operators.

A constraint is required to bound $\sum_{m \in \mathcal{M}} x_{jm}^k$, since each task can be assigned to at most one operator:

$$\sum_{m \in \mathcal{M}} x_{jm}^k \leq 1, \, \forall \, k \in \{1, \ldots, K\}, \, j \in \{1, \ldots, N^k\}. \tag{2.7}$$

Consequently, changes are made in expressions for ε_j^k and Constraint (2.5). In single-teleoperator setting, if two edges are both scheduled to be teleoperated, the travelling time of two cannot overlap. Analogously, with multiple operators, if two edges are scheduled to be teleoperated by the same operator (i.e., $x_{jm}^k = 1$ and $x_{im}^l = 1$), the traveling time of the two cannot overlap. Thus constraint (2.5) is modified as

$$x_{jm}^k + x_{im}^l = 2 \implies \tau_j^k \geq \varepsilon_i^l \text{ or } \tau_i^l \geq \varepsilon_j^k. \tag{2.8}$$

Constraint (2.6) is repeated for all x_{jm}^k.

2.4.2 Solving the MILP

A globally optimal solution to a MILP can be found using solvers like Gurobi or CPLEX. In this project, MILP is implemented using Python and solved with Gurobi API. However,

as mentioned earlier, while such solvers are effective for small problem instance, they do not scale to large instances involving many robots, each with ten or more tasks in its mission. In the next section, we present an efficient anytime algorithm that makes use of the problem structure to provide a fast and high-quality solution of Problem. 1.

2.5 Anytime Algorithm

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.

2.5.1 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 $\varepsilon^k_{N^k} = \mu(\mathcal{S}, \mathcal{P})$). We call the addition of a task e^k_i to schedule $\mathcal{S}$ a Greedy Insertion if the addition of e^k_i directly reduces $\varepsilon^k_{N^k}$, without increasing the team makespan.

Pseudo-code for the `Greedy Insertion` algorithm is presented in Algorithm 2. 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 $\varepsilon^k_{N^k}$, the mission time of any robot $k \in \overline{\mathcal{K}}$ by the most, while not increasing the makespan $\mu(\mathcal{S}, \mathcal{P})$. This task is then added to the schedule with the position that gives the best performance.

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. 2.2 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 $\mu(\mathcal{S}, \mathcal{P})$.

Algorithm 2 `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 $\mathcal{S}'$

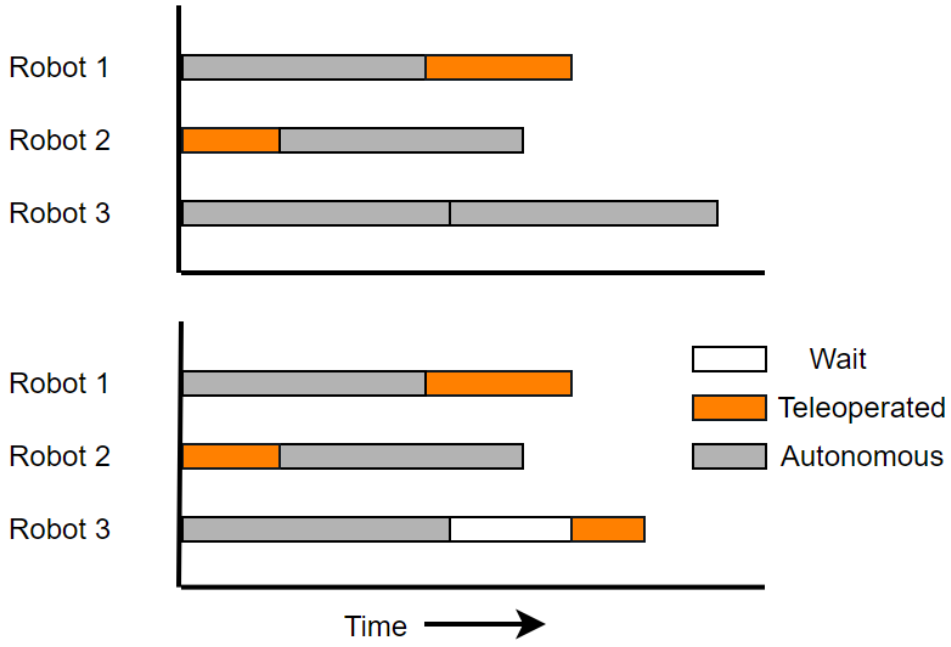

Figure 2.2: Example of Greedy Insertion. Robot 3 is the makespan robot, and by teleoperating its last task, we reduce its total mission time.

2.5.2 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. Since task durations are fixed values and all robots are released at the initial time, the start and finish time for each task can be calculated with $\mathcal{S}$ and $\mathcal{P}$. 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 whose mission duration is longest, such that the mission completion duration of this robot is the makespan (i.e., $\varepsilon^k_{N^k} = \mu$). We call the addition/insertion 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 allows further 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.

[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 delays the execution of later tasks in the schedule. Thus, such an idle time cannot help improve the makespan and we may skip it.

Pseudo-code for the `BlockRemoval` algorithm is presented in Algorithm 3. 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 3 `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. 2.3 to illustrate this operation. Given the schedule generated in Fig. 2.2, further Greedy Insertion is not possible. Adding task of e_1^3 to the schedule does not reduce makespan because the task final task of Robot3, e_2^3, will have to wait until the operator finishes the 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.

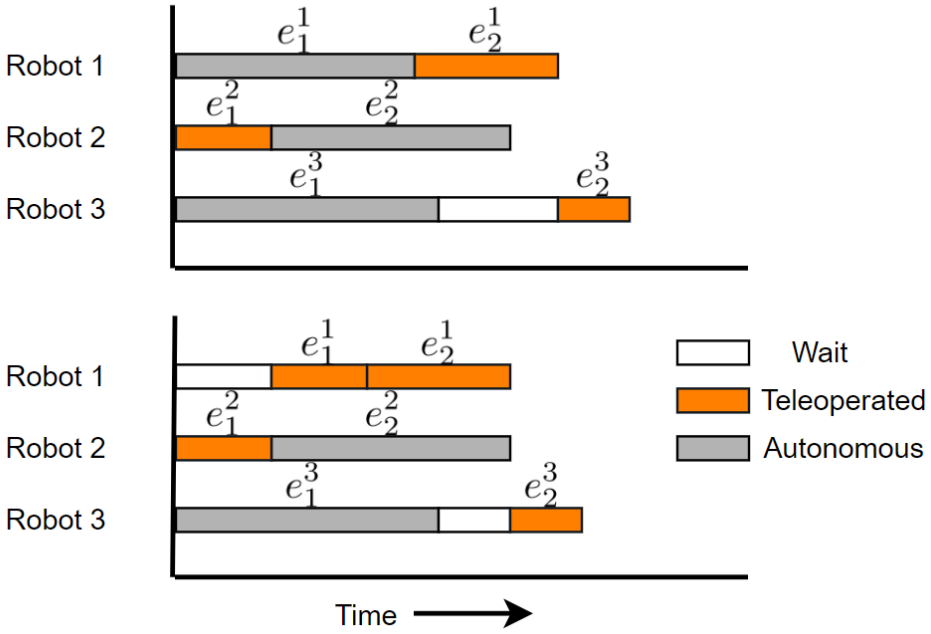

Figure 2.3: Example for Block Removal. Makespan Robot 3's mission time is reduced indirectly by teleoperating e_1^1.

2.5.3 Iterative Greedy

Starting with an empty teleoperation schedule, the Iterative Greedy algorithm first generates an intermediate schedule using Alg. 2. Using this schedule, we try the Block Removal routine using Alg. 3. The schedule is iteratively improved by applying Alg. 2 and Alg. 3 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 4 `Iterative Greedy`

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

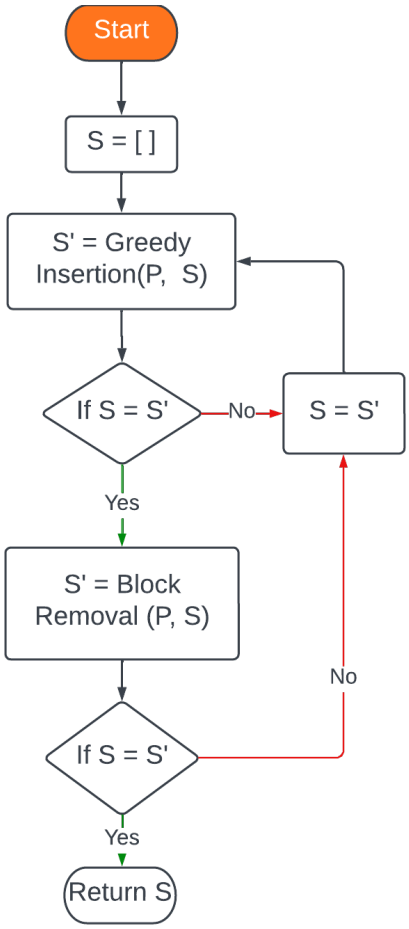

Figure 2.4: Iterative Greedy starts with an empty schedule and repeatedly call Greedy Insertion to expand the schedule. When Greedy Insertion fails to expand the schedule, Block Removal is called. When both cannot expand the schedule, algorithm terminates.

2.6 Baseline Heuristics

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

MILP Solution: The MILP formulation in Section 2.4 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 2.

2.7 Evaluation and Results

In this section, we present performance results for a simulated multi-robot scheduling problem under the following methods (details in Section 2.6): 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:

$$\begin{aligned} \beta_j^k \sim U[10, 20], \;\; \Delta\tau_j^k \sim U[0, 10], \\ \alpha_j^j \leftarrow \beta_j^k + \Delta\tau_j^k. \end{aligned} \tag{2.9}$$

2.7.1 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 2.1. 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 2.1. 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.

Table 2.1: Program Run 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

2.7.2 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 Iterative Greedy algorithm to the optimal schedule) is shown in Fig. 2.5. 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

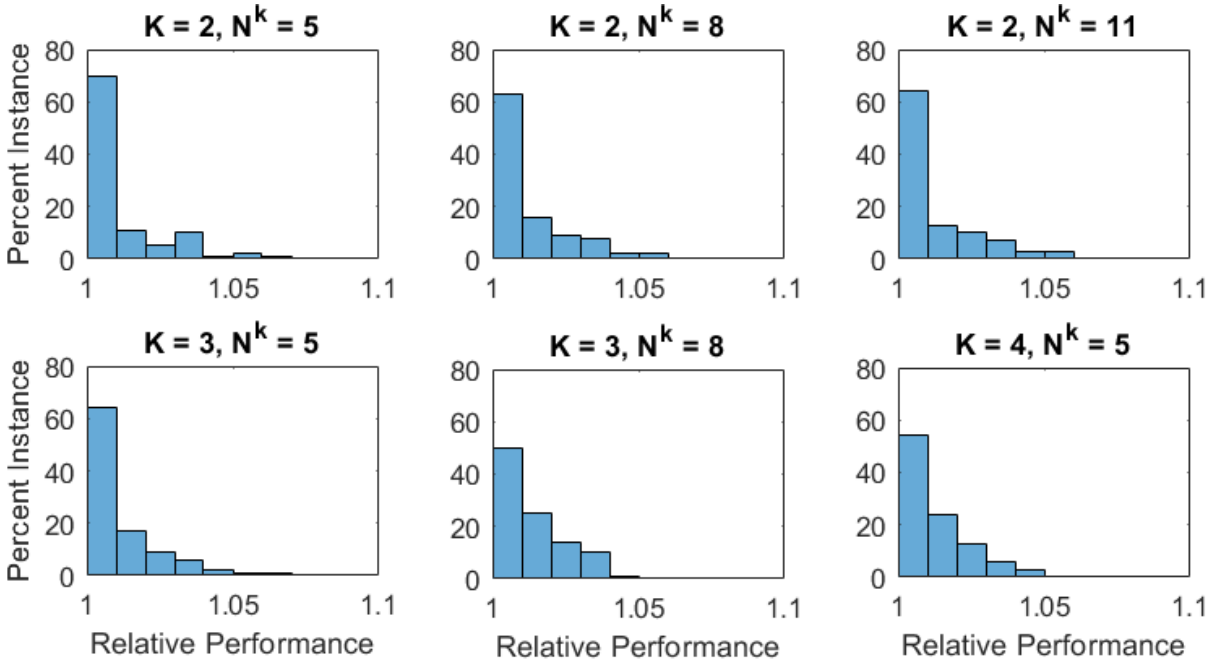

Figure 2.5: Relative performance of the Iterative Greedy methods compared to the optimal solution for number of robots $K \in \{2, 3, 4\}$, and number of tasks $N^k \in \{5, 8, 11\}$ for all robots. Each plot shows the distribution of 100 instances based on their relative performance (ratio of makespan under Iterative Greedy method to the optimal makespan). The x-asix, Relative Performance, is calculated using the makespan of Iterative Greedy divided by the optimal makespan (MILP solution)). The y-axis is percentage of instance that falls in the bin.

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.

2.7.3 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. 2.6 shows performance comparison of the different algorithms. The y-axis of Fig. 2.6 is the makespan of baseline policies divided by the makespan of Iterative Greedy, then minus 1. 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 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.

2.7.4 Problem Instance

Fig. 2.7 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 gaps (idle time) in operator's schedule. Then the Block Removal algorithm removes these gaps and results in a schedule with very little idle time for the operator. The MILP solution shows that a better performing schedule is possible even with a greater idle time.

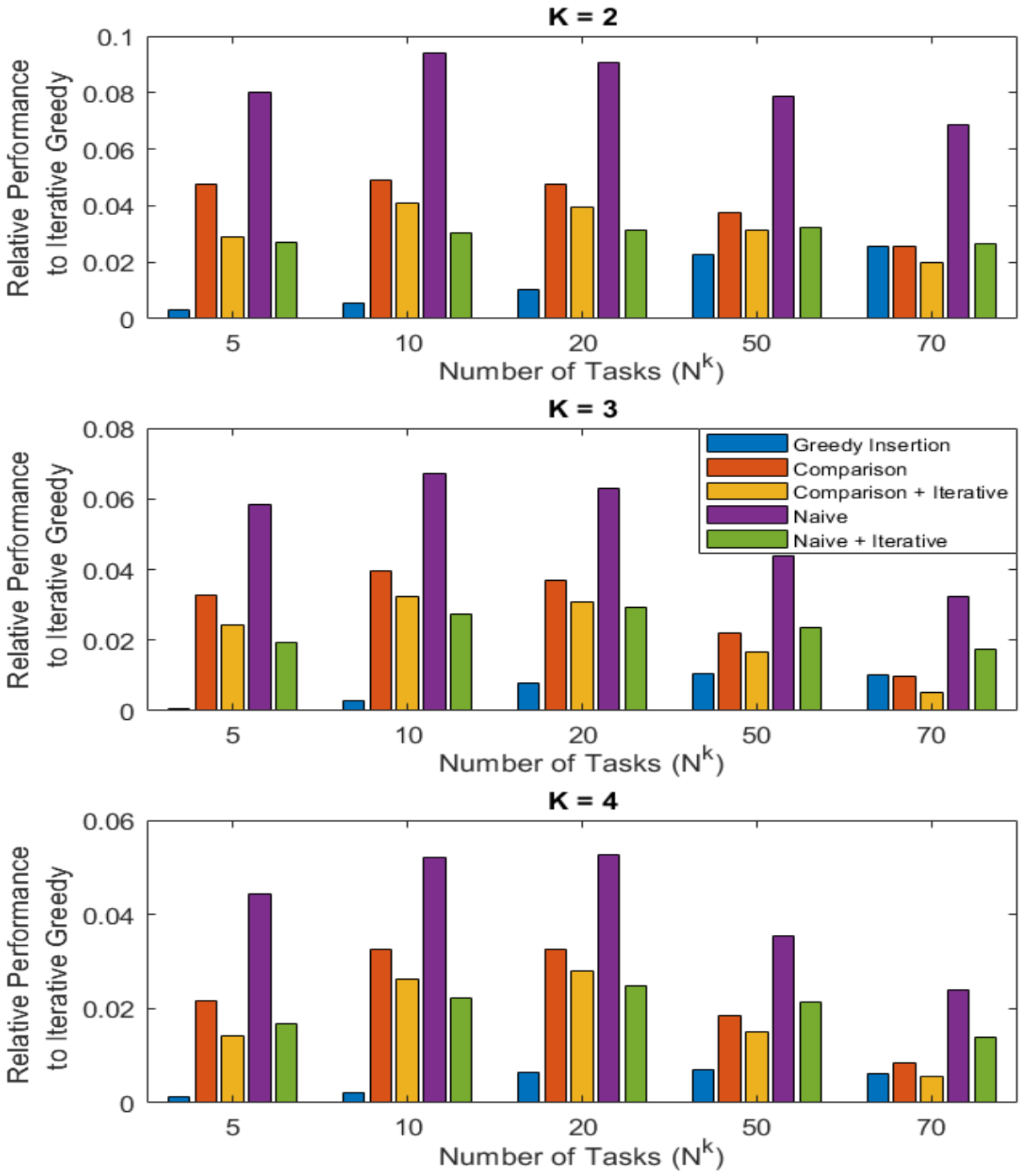

Figure 2.6: Performance comparison of baseline solution techniques to the proposed Iterative Greedy algorithm. The plots show relative performance of different techniques for up to 4 robots and 70 tasks each.

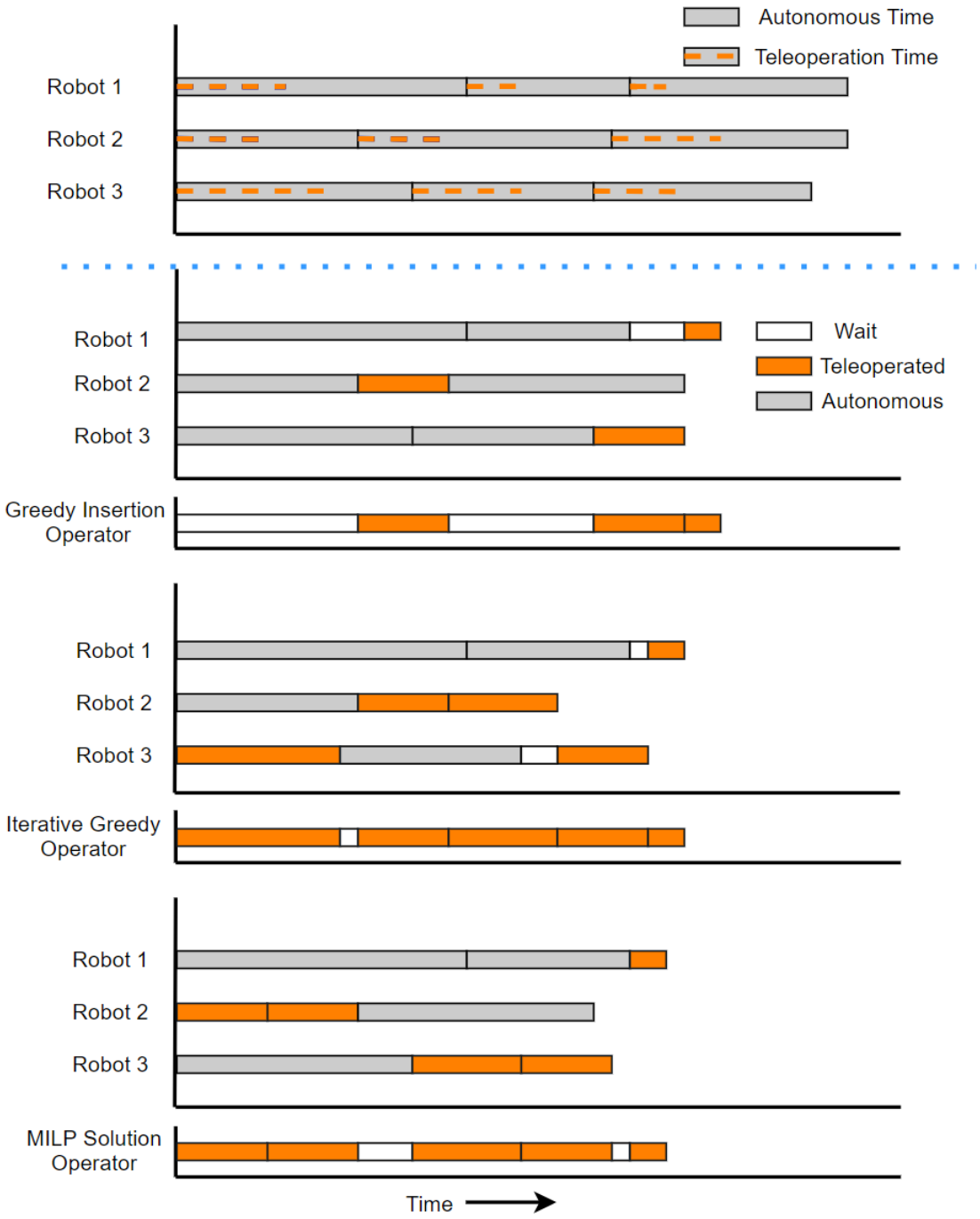

Figure 2.7: Scheduling of Iterative Greedy, Greedy Insertion and MILP Solution on a Multi-robot Mission Instance.

Chapter 3

Stochastic Scheduling Problem

3.1 Stochastic Problem Statement

We consider a system consisting of one human operator supervising a fleet of K identical autonomous robots. Each robot $k \in \mathcal{K} := \{1, \ldots, K\}$ is assigned a mission $p^k \in \mathcal{P} := \{p^1, \ldots, p^K\}$. To complete its mission p^k the robot k is required to complete a sequence of N^k tasks. The j^{th} task of robot k is denoted as e_j^k. For each task, a robot can either operate autonomously or be teleoperated by the human operator. Task durations in the two operating modes are random variables with a known underlying distribution model. Mean values of these random variables are independent across tasks and reflect task difficulty levels. Task duration for e_j^k under autonomous operation is denoted by random variable A_j^k with corresponding mean α_j^k. Similarly, task duration under teleoperation is denoted as B_j^k with a mean of β_j^k. In later sections, we refer to α_j^k and β_j^k as nominal values. Also, we denote the realization of autonomous and teleoperated completion time using lowercase characters: a_j^k and b_j^k. The human operator teleoperates the robots according to schedule $\mathcal{S} := \langle s_1, s_2, \ldots \rangle$, which is a sequence of tasks to be teleoperated.

We can use logic shown in the Algorithm 1 to find expression for completion of each robot give a schedule $\mathcal{S}$ with random variable task duration, $\mathcal{T} = \{T^1, \ldots, T^K\}$, the makespan of the team is determined by the maximum of all robot completion time, $\mu = \max\{T^1, \ldots, T^K\}$.

We studied and derived properties of exponential distribution and hypoexponential distribution (summation of independent exponential random variables with different parameters). If task completion time random variables follow exponential distribution, those

properties give us the formula for expected makespan $\mathbb{E}(\mu)$ of simple examples. However, there are some limitations of those properties which needs to be further solved, so that they can be applied on more complicated examples. The derivation of those properties are included in Appendix A.

Problem 4. (Stochastic Scheduling) Given the set $\mathcal{K}$ of robots, the missions $\{p^1, \ldots, p^K\}$ for each robot, and random autonomous and teleoperation task durations A_j^k and B_j^k, find a policy for computing schedule $\mathcal{S}$ that minimizes the expected makespan $\mathbb{E}(\mu)$.

Our goal for the Stochastic Scheduling problem is to minimize the expected makespan $\mathbb{E}(\mu)$ given an instance of Problem 4 by finding a policy that computes a teleoperation schedule and updates that schedule given a state of the team X^t at time t. The state of the team includes the task that each robot is working on (or waiting to start) in p^k, and the robot being teleoperated, if any.

Since robots completion times are random variables, we define the expected makespan of the system as:

Definition 5 (Expected Makespan)**.** Let, X^t be the state of the team and $\mathcal{S}^t$ be the schedule in use at time t. At time t, given the team state X^t and schedule $\mathcal{S}^t$, the expected makespan is given by $\mathbb{E}(\mu|X^t, \mathcal{S}^t) = \mathbb{E}(\max_{k\in\mathcal{K}}\{T^k|X^t, \mathcal{S}^t\})$.

3.2 Selective Replanning

Form our literature review, stochastic human multi-robot task allocation problems take the approach of replanning or dynamic scheduling based on observation along task execution. [28], [33]. Also, through our study and experiment on the system represented in Problem 4, replanning according to team state during mission execution improves the average performance over realizations of one instance. As the replanning happens while the team is executing missions, and the replanning trying to optimize performance based on in-time observation of the system, it should be time efficient, and in the best case, with negligible computation time.

The first step movement of the operator and robots need to be determined before we go into replanning. At this point, the only information is $\mathcal{P}$ which includes each robot's tasks and the nominal completion time for each task under autonomous mode and teleoperation. As random variables are used to represent task durations, over infinitely many realization of an instance of Problem 4, each robot may have a chance to be the makespan robot,

so an approach to obtain a smaller expected makespan is reducing the completion time of robot(s) that with largest or larger chance to contribute to the expected makespan over infinitely many realizations, as the expected makespan can also be seen as the average of makespan over infinitely many realizations of an instance.

Say robot k is associated with an indicator function $I(k)$; $I(k)$ takes the value 0 when robot k is not the makespan robot for an instance of Problem 4, or takes the value of 1 if it is the makespan robot. Again, due to the stochastic setup, the makespan robot is not constant over multiple realization of an instance.

Definition 6 (Expected Makespan Robot)**.** The expected makespan robot is defined as the robot k that has the largest probability to have its indicator function equals 1, $k = \arg\max_{k \in \mathcal{K}} \mathbb{P}(I(k) = 1)$.

For the initial schedule which determines the first step for all agents, if it can minimize the contribution of robots to the expected makespan, it will help the team move towards a smaller expected makespan. As the nominal completion time is also the expected completion time of each task in long run, the makespan robot of the deterministic Problem 1 has larger chance to be makespan robot in realization of the corresponding instance of stochastic Problem 4. The proposed solution for Problem 1, *Iterative Greedy*, works by reducing the completion time of the deterministic makespan robot, and it is time-efficient. Thus, we use the schedule $\mathcal{S}$ obtained by applying *Iterative Greedy* with nominal task duration to instruct the first step of robots and operator. $\mathcal{S}$ stores the indices of teleoeprated tasks with the order of teleoepration.

With the first step decided, we propose an upper bound frequency for replan which is task completion. If we replan using a frequency higher than this, replanning will happen when all robots are in the middle of a task. Because of the uncertainty of task completion time, the schedule that replanned when all robots are in the middle of tasks, may not still be an ideal one any more when any robot finishes its task and has the work force to execute the replanned schedule. Every time a task (let's call it e^k_{j-1}) completes, Algorithm 5: `Branch Policy` is called to find the best next step movement for robot k or the robot k and the operator, depending on the operation mode of e^k_{j-1}.

In our system, task completion is divided into Autonomous Task Completion ($\{k, j-1\} \notin S$) and Teleoperated Task Completion ($\{k, j-1\} \in \mathcal{S}$). The later one also frees the operator work force. Based on this, these are two main branches of our policy.

3.2.1 Teleoperated Task Completion

When a teleoperated task $\{k, j-1\}$ finishes, robot k and the operator are both free. We need to determine the next step movement for both. Once the operator finishes teleoperating the task $\{k, j-1\}$, the policy first checks if the execution of e^k_{j-1} deviates from the nominal value more than a certain amount using inequality 3.1.

$$|b^k_{j-1} - \beta^k_{j-1}|/\beta^k_{j-1} > \delta \tag{3.1}$$

δ is a positive constant. Inequality 3.1 measures the deviation of realization b^k_{j-1} of e^k_{j-1} from the nominal value β^k_{j-1}. If inequality 3.1 is satisfied, a partial schedule will be computed for tasks that haven't started yet using `Iterative Greedy` and nominal task duration. Time needed for tasks that are under operation will be taken into account using exponential distribution model, but those tasks' operation mode will not be changed in the partial schedule planning.

User can tune δ based on preference or performance needed. When δ is closer to 0, the policy is sensitive to the difference between realization and nominal value, and more replanning will happen. However, of course, the average makespan is expected to be lower with higher frenquncy of replanning. As the value of δ increases, replanning will be less frequent.

In the simulation, the progress of the schedule $\mathcal{S}$ is tracked by a schedule pointer `Ptr`. `Ptr` moves to the next teleoperated task once e^k_{j-1} is done in our simulation system. Thus, when the partial schedule is computed, the task that `Ptr` is points will be a task that haven't started yet. Thus, the partial schedule will be concatenated to $\mathcal{S}$ after the position of `Ptr` -1, as shown in Line 4 of Algorithm 5.

3.2.2 Autonomous Task Completion

When robot k completes task e^k_{j-1} autonomously, the branch will go down one level to check if the robot's next task e^k_j is in the remaining part of the schedule $\mathcal{S}$. Different strategies will be applied to two sub-branches based on situation assessed.

3.2.3 Task $\{k, j\}$ in Schedule

In this sub-branch, at time t, the robot k's next task e^k_j is in schedule $\mathcal{S}$, and there could be ≥ 0 tasks between the current teleoperated task $s' = \{h, i\}$ and $\{k, j\}$. The replanning for

this branch is straight forward. We would compare the nominal autonomous completion time of the e_j^k and teleportation completion time following the original schedule, and choose the one that gives earlier nominal completion time for e_j^k. The time needed for operator to be ready for e_j^k depending on how long the operator needs from the current time t to finish teleoperation tasks prior to e_j^k. This time is called $T_{kj}^{\texttt{opt}}$. If we neglect the situation where operator waits for robots, $T_{kj}^{\texttt{opt}}$ can b computed using the sum of nominal teleoperation task completion time. Then, the expected completion time for e_j^k under teleoperation is $T_{kj}^{\texttt{opt}}+\beta_j^k$, since operator will definitely be ready for e_j^k later than robot k. If $\alpha_j^k \leq T_{kj}^{\texttt{opt}}+\beta_j^k$, then $\mathcal{S} = \mathcal{S}\backslash\{e_j^k\}$. Robot k will run e_j^k autonomously. Otherwise, waiting for the operator may benefit robot k more, so robot k will wait for the operator.

Letting robot k run e_j^k autonomously can be analyzed in two situations: 1) the robot has only autonomous tasks left, and 2) the robot has one or more teleoperated task left. For 1), the robot may finish its task autonomously, and other than serious delay happening to this robot, its work has not interference with the rest of the team and/or makespan. For 2), without delay in following tasks, the robot will be further ahead of the operator, but we can simply ask the robot to wait for the operator at the beginning of the next teleoperated task of teleoperation for this task is desired. This means the decision of running e_j^k autonomously, given $\alpha_j^k \leq T_{kj}^{\texttt{opt}}$, will not impact the peformance of the team in a negative way.

3.2.4 Task $\{k, j\}$ not in Schedule

When robot k is ready for task e_j^k at time t, and e_j^k is not in $\mathcal{S}$, the decision to be made here is whether there is a need to teleoeprate e_j^k and when should it be teleoeprated. The demand for replanning here would increase, if execution of e_{j-1}^k deviates from the nominal value. Thus, the percentage difference of actual and nomianl value of e_{j-1}^k's duration is checked using inequality 3.2

$$(a_{j-1}^k - \alpha_{j-1}^k)/\alpha_{j-1}^k > \delta \tag{3.2}$$

δ here is the same as the one in inequality 3.1. Absolute value is not applied in inequality 3.2, not like in 3.1. This is decided through experiment with our simulation. Without absolution value, inequality 3.2 performs about the same as with absolute value, and results in less rounds of replan for a given size of instances. If inequality 3.2 is not satisfied, then no replanning is needed and robot k will run e_j^k autonomously.

However, if inequality 3.2 is satisfied, the policy moves on to check if e_j^k can have an earlier finish time by making it the next task to teleoperate. $T_{\text{current}}^{\text{opt}}$ the expected time

needed for the operator to finish its current task. If $\alpha_j^k \leq T_{\text{current}}^{\text{opt}} + \beta_j^k$, then for sure there is a chance that robot k will benefit from be teleoperated. And, a partial schedule will be computed for tasks that have not started yet, using `Iterative Greedy` with nominal values of task completion time. The partial schedule will be concatenated to $\mathcal{S}$ after the position of `Ptr`, shown in Line 13 of Algorithm 5). Robot k will follow this partial schedule for the operation of e_j^k.

Algorithm 5 `Branch Policy`

Input: $\mathcal{P}$, $\mathcal{S}$, `Ptr`, X^t, δ

1: **if** $\{k, j-1\} \in \mathcal{S}$ **then**
2: **if** $|b_{j-1}^k - \beta_{j-1}^k|/\beta_{j-1}^k > \delta$ **then**
3: `partialS` $=$ `IterativeGreedy`$(\mathcal{P}, X^t)$
4: $\mathcal{S} = \mathcal{S}[1 : \texttt{Ptr} - 1] + \texttt{partialS}$
5: **else**
6: **if** $\{k, j\} \in \mathcal{S}$ **then**
7: **if** $\alpha_j^k \leq T_{kj}^{\text{opt}} + \beta_j^k$ **then**
8: delete $\{k, j\}$ from $\mathcal{S}$
9: **else**
10: **if** $(a_j^k - \alpha_j^k)/\alpha_j^k > \delta$ **then**
11: **if** $\alpha_j^k \geq T_{\text{current}}^{\text{opt}} + \beta_j^k$ **then**
12: `partialS` $=$ `IterativeGreedy`$(\mathcal{P}, X^t)$
13: $\mathcal{S} = \mathcal{S}[1 : \texttt{Ptr}] + \texttt{partialS}$
14: Follow $\mathcal{S}$

In Algorithm 5, `Iterative Greedy` is expanded to take team state X^t as an input, so that Iterative Greedy will clearly know which tasks are those have not started yet and those are under execution.

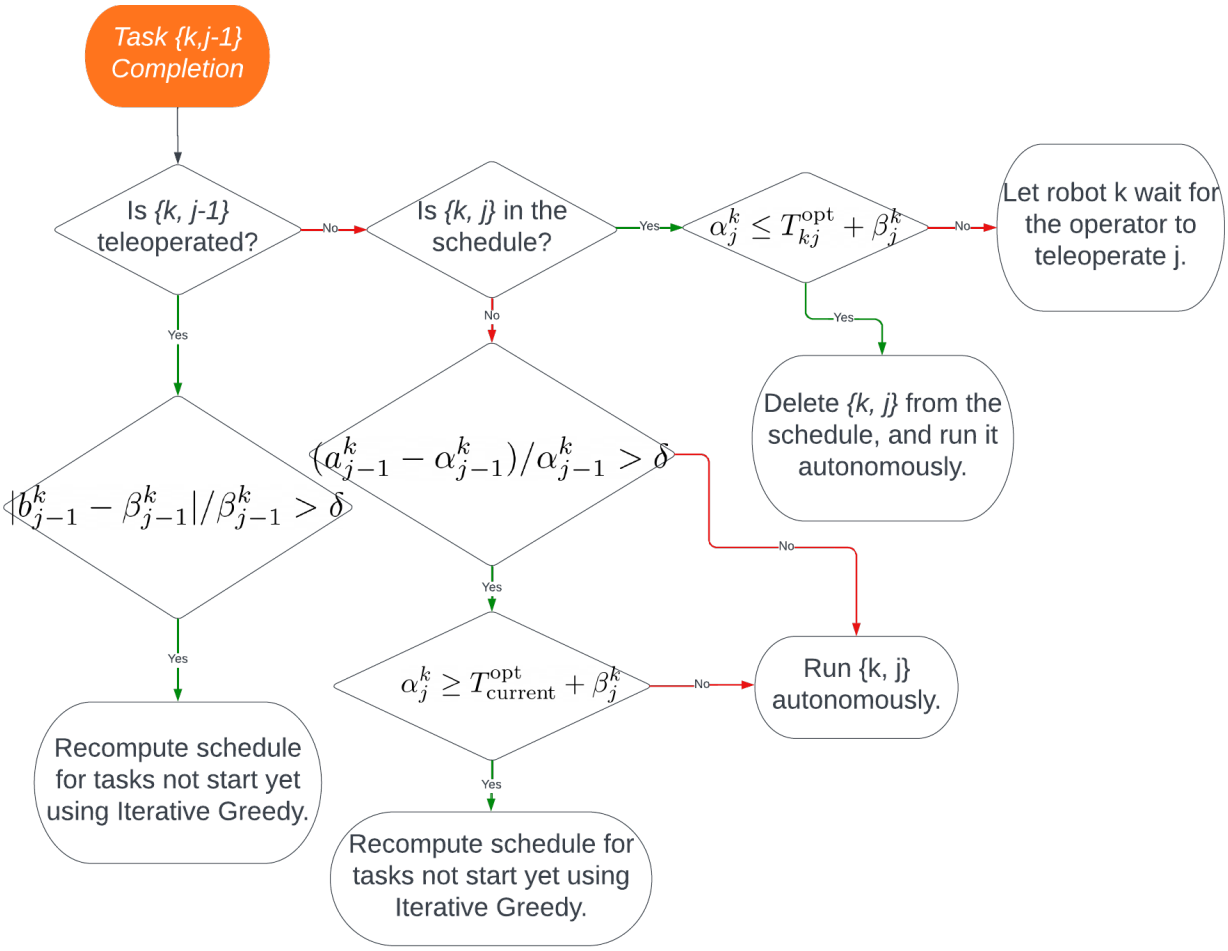

Figure 3.1: Selective Replan Flow Chart

3.3 Baseline Policies

We consider the following baseline solution methods to assess the performance of the presented Selective Replan Policy. These policies differ in terms of when and how they update the schedule given the current state of the system. Note that, in all of those polices, we still only replan schedule when at least one robot is available to start its next task (i.e. Task Completion).

No Replanning: Given a problem instance, a schedule is obtained using iterative greedy presented in Section 2.5, with nominal task duration. This schedule is then followed until all robots complete their missions.

Always Replanning: Under this policy, we start with a schedule obtained using the No Replanning policy. Every time a robot finishes a task, an updated schedule is obtained using the iterative greedy policy given current state of the system and nominal value of task duration. Tasks already in progress are excluded from being scheduled.

Makespan-Triggered Replanning: Every time a robot finishes a task, expected makespan of the system is computed (Sampling 100 times for the problem and take the average makespan.). If the new value is larger than the previous computed expected makespan, then an updated schedule is obtained using the iterative greedy policy given current state of the system. This policy tries to limit the number of schedule recomputations by only running iterative greedy algorithm if expected makespan increases beyond the previous estimate.

Naïve Greedy Policy: Under this policy, every time a task from the schedule is completed, the operator is simply scheduled to teleoperate the next available task of the expected makespan robot (Sampling 100 times for the problem, and sum the completion time for each robot only when they are the makespan robot of a turn. Take the robot has largest sum.). If the makespan robot is still executing a task, the operator waits for the robot.

3.4 Evaluation and Results

In this section, we validate the efficacy of the proposed solution approach using a simulated multi-robot stochastic scheduling problem. We present performance comparison for the scheduling problem under the following policies: 1) No Replanning, 2) Always Replanning, 3) Selective Replanning, 4) Makespan-Triggered Replanning, and 5) Naïve Greedy. The problem and the solution frameworks for all algorithms were implemented using MATLAB.

We set up the system with K robots working on independent missions each consisting between n_1 and n_2 tasks. 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 nominal task duration under teleoperation λ_β, and the sum of two is used as the nominal task duration under autonomous operation λ_α:

$$\begin{aligned} \lambda_\beta \sim U[30, 60], \;\; \Delta\tau \sim U[0, 40], \\ \lambda_\alpha \leftarrow \lambda_\beta + \Delta\tau. \end{aligned} \tag{3.3}$$

The probability model for task duration is chosen to be exponential distribution. In the system, a robot communicates with the scheduler only when a task is finished. Exponential distribution has the memoryless property, and it suits to describe a situation like robot keeping trying a task until it is successfully completed. Typical examples are a robot trying to go across the street at where is no traffic light and robot trying to grab a highly deformed item.

For testing, 1000 problem instances were created for a given instance size and each instance was simulated 100 times under each of the above solution methods. Average makespan of the problem instance was recorded at the end of the simulations.

Fig. 3.2 shows the average makespan of the simulated system with policies relative to the deterministic estimate in percentage difference. This measurement is called Relative Difference in Makespan (R_μ) and is defined in equation 3.4. $\mu(S^p)$ is the average makespan achieved following a policy, and μ^d is the deterministic estimation of makespan computed using nominal task duration.

$$R_\mu = \frac{\mu(S^p) - \mu^d}{\mu^d} \tag{3.4}$$

R_μ is obtained for various team sizes (number of robots and tasks in missions). In the figure, we have the following observations: 1) As number of robots in the system increases, the relative performance of all policies degrade. 2) As number of tasks in robots' missions increase, the relative performance of all policies improves slightly. 3) The Always-Replanning policy performs the best with the Selective-Replanning being quite similar in performance.

In addition to take the R_μ average of 100 iterations for each problem instance, we capture the worst 20 realizations out of 100 and take average for these portion. Fig. 3.3 shows the Selective Replan outperforms most other policies in the worst 20% cases, with a minor disadvantage when compared to the Always Replan policy. An interesting finding

here is the Makespan-Triggered Policy outperforms the Greedy Policy in worst 20%, while the converse happens in the average case.

The main advantage of the Selective Replanning policy is seen when we look at the number of schedule replan that it executes. Table 3.1 shows the average number of times that the schedule is replanned under three different policies with varying number of robots and range for number of tasks per robot. Recall that the Offline and the Greedy policies do not replan the schedule. From the table, we observe the following:

1) The Makespan-triggered replanning policy results in the least amount of replanning followed by the Selective-replanning and then the Always replanning policy.

2) As the number of robots and tasks increases, the average amount of replanning also increases under all three policies.

The results demonstrate that the Selective Replanning policy is able to perform just as good as the policy that replans at every state change while reducing the amount of replan on average by 55.85%.

Table 3.1: Average Number of Replanning Done under Different Policies

	$K = 2$ $N^k \in [5, 10]$	$K = 4$ $N^k \in [5, 10]$	$K = 6$ $N^k \in [5, 10]$	$K = 2$ $N^k \in [15, 20]$	$K = 4$ $N^k \in [15, 20]$	$K = 6$ $N^k \in [15, 20]$
Always Replanning	13.1	26.3	38.7	32.7	65.5	98.5
Selective Replanning ($\delta = 0.4$)	6.9	11.5	15.7	16.4	26.4	36.9
Makespan-Triggered Replanning	2.0	1.3	1.1	1.7	1.2	1.0

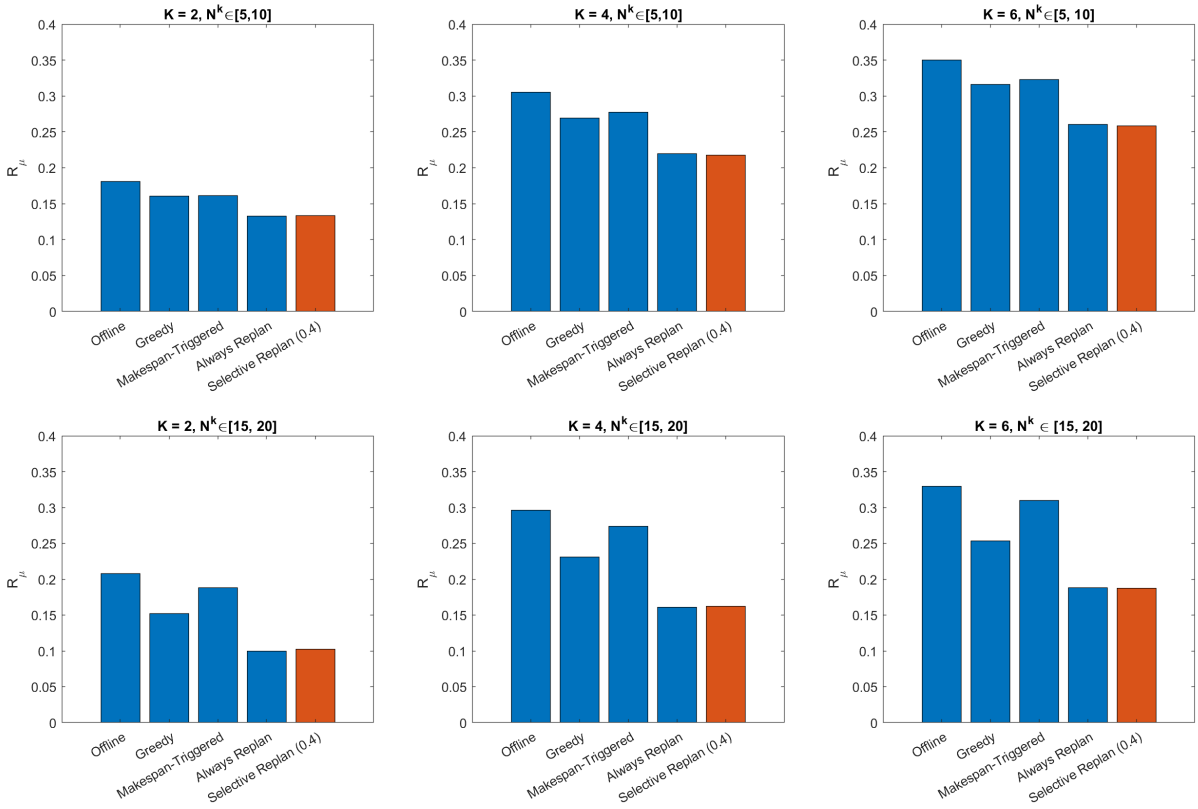

Figure 3.2: Average system makespan under different policies relative to the deterministic estimate.

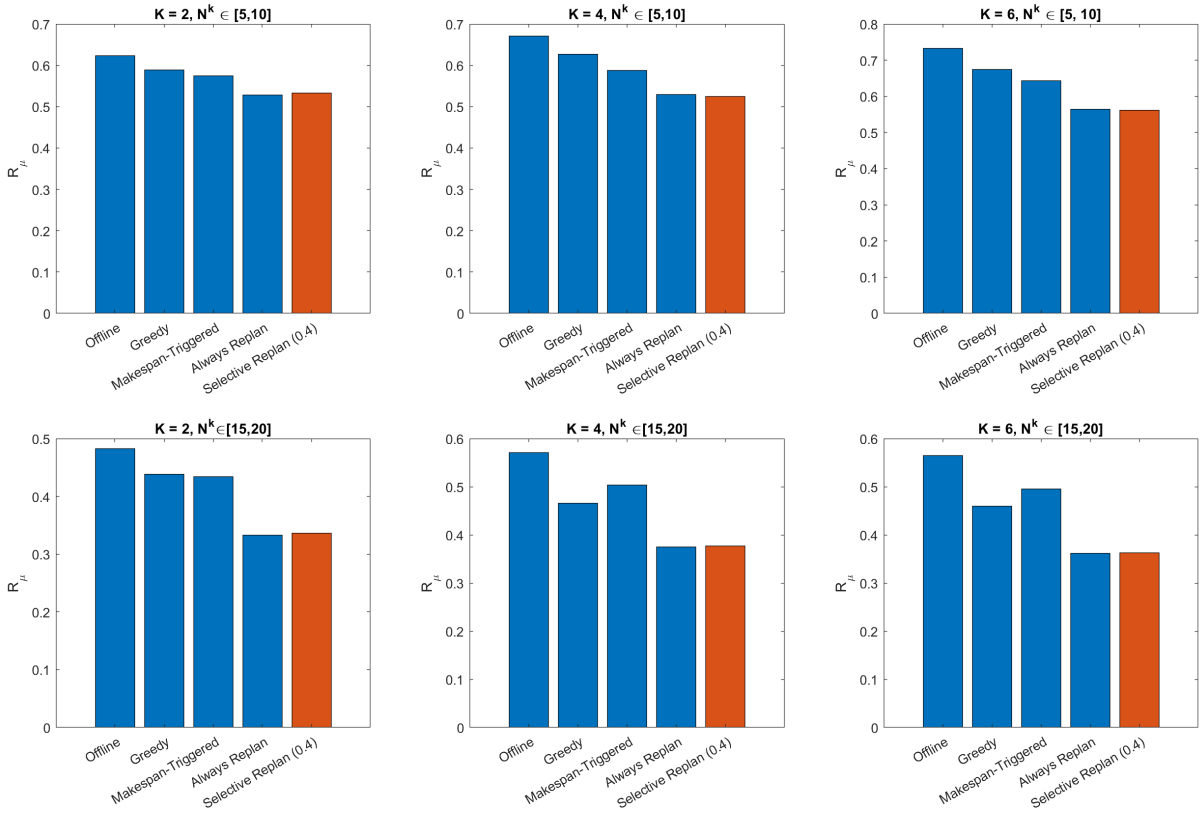

Figure 3.3: Worst 20% average system makespan under different policies relative to the deterministic estimate.

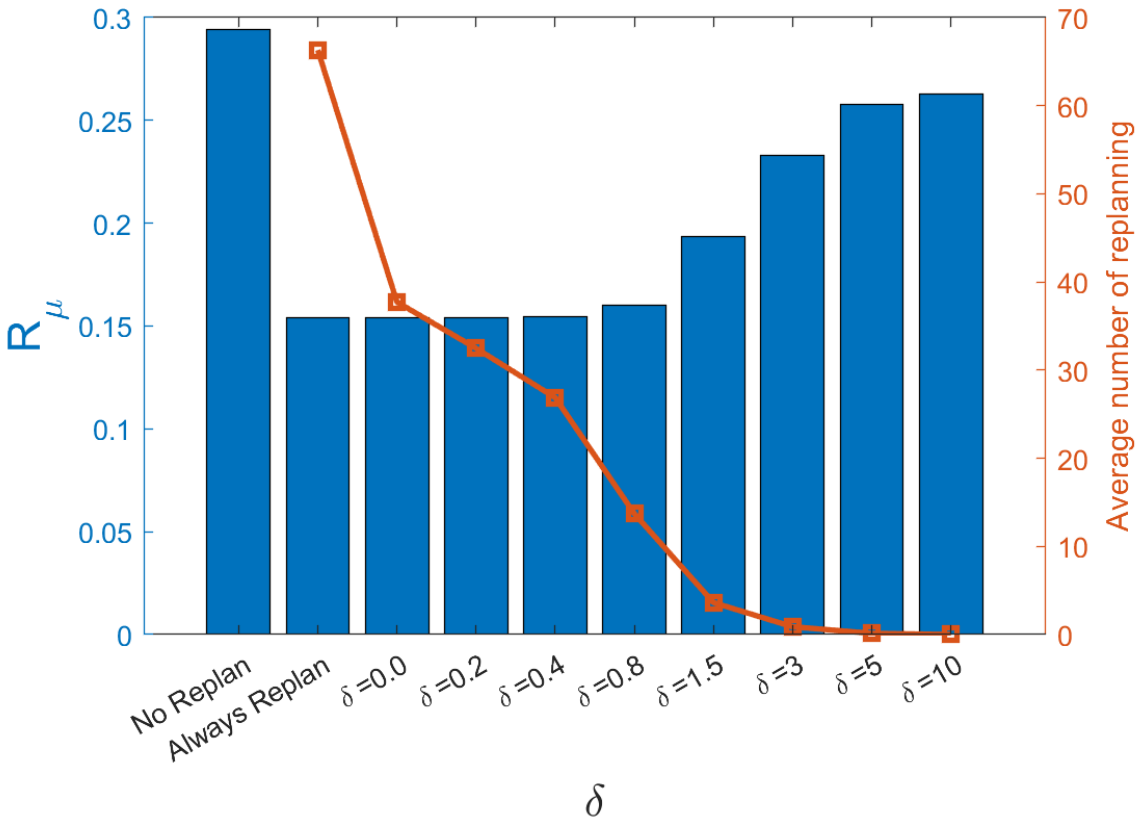

Figure 3.4: Average system makespan of Selective replan policy and number of average replanning executed for different values of δ relative to the deterministic estimate.

Chapter 4

Discussion and Summary

We study a system consisting one human operator and multiple robots, with possible extension to multiple human operators. The goal is to schedule the human operator to a team of multiple robots to help robot complete some challenging tasks with high efficiency, such that the team makespan is minimized. Stochasticity is also considered in task completion time of autonomous and teleoperation modes.

We first show that the deterministic version of the problem is NP-Hard through reduction from the $2p1n-3SAT$ problem. MILP model is established for the deterministic problem to generate the optimal solution. Due to the nonscalbility of MILP, we developed the `Iterative Greedy` algorithm that cycles through two sub-routines: Greedy Insertion and Block Removal. This algorithm repeatedly expand the teleoperation list by one task every cycle. Greedy Insertion is the primary selection methods, and insertion from Block Removal takes place to improve the schedule by removing *blockages* when needed. The algorithm scales well with problem size shown in our simulation with instance up to 4 robots and 70 tasks each. It finds schedule which performs close to MILP solution for small to moderate sized problem. Also, it outperforms baseline greedy solutions. The `Iterative Greedy` algorithm can be applied to any greedily-generated schedule to further improve the performance.

A parameterized branch policy (Selective Replanning) utilizing `Iterative Greedy` for re-planning is developed for the stochastic version of the problem. With the initial schedule computed with `Iterative Greedy`, the policy checks if re-plan is needed with a serious of conditions and decide the next step movement for agent(s) based on the terminal leaded by condition checks. Users can change the parameter to adjust the frequency of re-plan, based on user preference and performance level they are seeking. The simulation results

show Selective Replanning only takes one third to half of replans compare to the Always Replanning policy, and the average performance is almost the same to Always Replan for parameter value up until 0.4. Selective Replanning achieves good average performance with limited number of replans.

For future research, our interest mainly falls in developing more comprehensive human-robot-collaboration model by incorporating more objective, such as workload, and constraints based on real-world collaboration setup and environment. Additionally, we are interested in study different models of predicting the robot completion time, such as normal distribution with bounds. Instead of distribution of completion time, we can also try a fixed distance with velocity prediction model.

References

[1] Riccardo Remo Appino, Veit Hagenmeyer, and Timm Faulwasser. Towards optimality preserving aggregation for scheduling distributed energy resources. *IEEE Transactions on Control of Network Systems*, 8(3):1477–1488, 2021.

[2] Patrick Benavidez, Mohan Kumar, Sos Agaian, and Mo Jamshidi. Design of a home multi-robot system for the elderly and disabled. In *2015 10th System of Systems Engineering Conference (SoSE)*, pages 392–397, 2015.

[3] Aldo Bischi, Leonardo Taccari, Emanuele Martelli, Edoardo Amaldi, Giampaolo Manzolini, Paolo Silva, Stefano Campanari, and Ennio Macchi. A rolling-horizon optimization algorithm for the long term operational scheduling of cogeneration systems. *Energy*, 184:73–90, 2019.

[4] Gerald G Brown and Robert F Dell. Formulating integer linear programs: A rogues' gallery. *INFORMS Transactions on Education*, 7(2):153–159, 2007.

[5] Stefano Carpin, Mike Lewis, Jijun Wang, Steve Balakirsky, and Chris Scrapper. Bridging the gap between simulation and reality in urban search and rescue. In *Robot Soccer World Cup*, pages 1–12. Springer, 2006.

[6] Jessie YC Chen and Michael J Barnes. Supervisory control of multiple robots: Effects of imperfect automation and individual differences. *Human Factors*, 54(2):157–174, 2012.

[7] Jessie YC Chen and Michael J Barnes. Human–agent teaming for multirobot control: A review of human factors issues. *IEEE Transactions on Human-Machine Systems*, 44(1):13–29, 2014.

[8] Yujiao Cheng, Liting Sun, Changliu Liu, and Masayoshi Tomizuka. Towards efficient human-robot collaboration with robust plan recognition and trajectory prediction. *IEEE Robotics and Automation Letters*, 5(2):2602–2609, 2020.

[9] Shih-Yi Chien, Michael Lewis, Siddharth Mehrotra, Nathan Brooks, and Katia Sycara. Scheduling operator attention for multi-robot control. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pages 473–479, 2012.

[10] Shih Yi Chien, Michael Lewis, Siddharth Mehrotra, and Katia Sycara. Imperfect automation in scheduling operator attention on control of multi-robots. In *Human Factors and Ergonomics Society Annual Meeting*, volume 57, pages 1169–1173, 2013.

[11] Shih-Yi Chien, Huadong Wang, and Michael Lewis. Human vs. algorithmic path planning for search and rescue by robot teams. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, volume 54, pages 379–383. SAGE Publications Sage CA: Los Angeles, CA, 2010.

[12] Jacob W Crandall, Mary L Cummings, Mauro Della Penna, and Paul MA De Jong. Computing the effects of operator attention allocation in human control of multiple robots. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 41(3):385–397, 2010.

[13] Jacob W Crandall, Michael A Goodrich, Dan R Olsen, and Curtis W Nielsen. Validating human-robot interaction schemes in multitasking environments. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 35(4):438–449, 2005.

[14] Abhinav Dahiya, Nima Akbarzadeh, Aditya Mahajan, and Stephen L. Smith. Scalable operator allocation for multi-robot assistance: A restless bandit approach. *IEEE Transactions on Control of Network Systems*, 2022. To Appear.

[15] Neel Dhanaraj, Rishi Malhan, Heramb Nemlekar, Stefanos Nikolaidis, and Satyandra K Gupta. Human-guided goal assignment to effectively manage workload for a smart robotic assistant. In *2022 31st IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*, pages 1305–1312. IEEE, 2022.

[16] M Bernardine Dias, Balajee Kannan, Brett Browning, E Jones, Brenna Argall, M Freddie Dias, Marc Zinck, M Veloso, and Anthony Stentz. Sliding autonomy for peer-to-peer human-robot teams. In *International Conference on Intelligent Autonomous Systems*, pages 332–341, 2008.

[17] S Alireza Fayazi, Ardalan Vahidi, and Andre Luckow. Optimal scheduling of autonomous vehicle arrivals at intelligent intersections via milp. In *2017 American Control Conference (ACC)*, pages 4920–4925. IEEE, 2017.

[18] Matthew Gombolay, Reed Jensen, Jessica Stigile, Toni Golen, Neel Shah, Sung-Hyun Son, and Julie Shah. Human-machine collaborative optimization via apprenticeship scheduling. *Journal of Artificial Intelligence Research*, 63:1–49, 2018.

[19] Piyush Gupta and Vaibhav Srivastava. Optimal fidelity selection for human-in-the-loop queues using semi-markov decision processes. In *American Control Conference*, pages 5266–5271, 2019.

[20] Andy Ham and Myoung-Ju Park. Human–robot task allocation and scheduling: Boeing 777 case study. *IEEE Robotics and Automation Letters*, 6(2):1256–1263, 2021.

[21] Sai Krishna Kanth Hari, Abhishek Nayak, and Sivakumar Rathinam. An approximation algorithm for a task allocation, sequencing and scheduling problem involving a human-robot team. *IEEE Robotics and Automation Letters*, 5(2):2146–2153, 2020.

[22] S Ehsan Hashemi-Petroodi, Simon Thevenin, Sergey Kovalev, and Alexandre Dolgui. Operations management issues in design and control of hybrid human-robot collaborative manufacturing systems: a survey. *Annual Reviews in Control*, 49:264–276, 2020.

[23] Yin Huang, Yi Zhang, and Hong Xiao. Multi-robot system task allocation mechanism for smart factory. In *IEEE International Information Technology and Artificial Intelligence Conference*, pages 587–591, 2019.

[24] Amro Khasawneh, Hunter Rogers, Jeffery Bertrand, Kapil Chalil Madathil, and Anand Gramopadhye. Human adaptation to latency in teleoperated multi-robot human-agent search and rescue teams. *Automation in Construction*, 99:265–277, 2019.

[25] Elsa A Kirchner, Su K Kim, Marc Tabie, Hendrik Wöhrle, Michael Maurus, and Frank Kirchner. An intelligent man-machine interface—multi-robot control adapted for task engagement based on single-trial detectability of p300. *Frontiers in Human Neuroscience*, 10:291, 2016.

[26] Jessica Leu, Yujiao Cheng, Changliu Liu, and Masayoshi Tomizuka. Robust task planning for assembly lines with human-robot collaboration. *arXiv preprint arXiv:2204.07936*, 2022.

[27] Michael Lin, Nuno C Martins, and Richard J La. Queueing subject to activity-dependent server performance: Task-assignment control policies for utilization rate reduction. *IEEE Transactions on Control of Network Systems*, 2021.

[28] Martina Lippi, Paolo Di Lillo, and Alessandro Marino. A task allocation framework for human multi-robot collaborative settings. *arXiv preprint arXiv:2210.14036*, 2022.

[29] Ali Ahmad Malik and Arne Bilberg. Complexity-based task allocation in human-robot collaborative assembly. *Industrial Robot: the International Journal of Robotics Research and Application*, 2019.

[30] Sandra Mau and John M Dolan. Scheduling to minimize downtime in human-multirobot supervisory control. Carnegie Mellon University, 2006.

[31] Selma Musić and Sandra Hirche. Control sharing in human-robot team interaction. *Annual Reviews in Control*, 44:342–354, 2017.

[32] Margaret Pearce, Bilge Mutlu, Julie Shah, and Robert Radwin. Optimizing makespan and ergonomics in integrating collaborative robots into manufacturing processes. *IEEE Transactions on Automation Science and Engineering*, 15(4):1772–1784, 2018.

[33] Andrea Pupa, Wietse Van Dijk, and Cristian Secchi. A human-centered dynamic scheduling architecture for collaborative application. *IEEE Robotics and Automation Letters*, 6(3):4736–4743, 2021.

[34] Qilei Ren, Ka Lok Man, Eng Gee Lim, Jinkyung Lee, and Kyung Ki Kim. Cooperation of multi robots for disaster rescue. In *IEEE International SoC Design Conference (ISOCC)*, pages 133–134, 2017.

[35] Spyros Reveliotis and Young-In Kim. Min-time coverage in constricted environments: Problem formulations and complexity analysis. *IEEE Transactions on Control of Network Systems*, 2021.

[36] Jennifer M Riley and Mica R Endsley. Situation awareness in HRI with collaborating remotely piloted vehicles. In *Human Factors and Ergonomics Society Annual Meeting*, volume 49, pages 407–411, 2005.

[37] Ariel Rosenfeld, Noa Agmon, Oleg Maksimov, and Sarit Kraus. Intelligent agent supporting human–multi-robot team collaboration. *Artificial Intelligence*, 252:211–231, 2017.

[38] Janne Roslöf, Iiro Harjunkoski, Tapio Westerlund, and Johnny Isaksson. Solving a large-scale industrial scheduling problem using milp combined with a heuristic procedure. *European Journal of Operational Research*, 138(1):29–42, 2002.

[39] Shaurya Shriyam and Satyandra K Gupta. Incorporating potential contingency tasks in multi-robot mission planning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3709–3715. IEEE, 2018.

[40] Shaurya Shriyam and Satyandra K Gupta. Incorporation of contingency tasks in task allocation for multirobot teams. *IEEE Transactions on Automation Science and Engineering*, 17(2):809–822, 2019.

[41] Gokul Swamy, Siddharth Reddy, Sergey Levine, and Anca D Dragan. Scaled autonomy: Enabling human operators to control robot fleets. In *IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 5942–5948, 2020.

[42] Daniel Szafir, Bilge Mutlu, and Terrence Fong. Designing planning and control interfaces to support user collaboration with flying robots. *The International Journal of Robotics Research*, 36(5-7):514–542, 2017.

[43] Gricel Vázquez, Radu Calinescu, and Javier Cámara. Scheduling multi-robot missions with joint tasks and heterogeneous robot teams. In *Annual Conference Towards Autonomous Robotic Systems*, pages 354–359. Springer, 2021.

[44] Huadong Wang, Michael Lewis, Prasanna Velagapudi, Paul Scerri, and Katia Sycara. How search and its subtasks scale in n robots. In *Proceedings of the 4th ACM/IEEE international conference on Human robot interaction*, pages 141–148, 2009.

[45] Choon Yue Wong and Gerald Seet. Workload, awareness and automation in multiple-robot supervision. *International Journal of Advanced Robotic Systems*, 14(3), 2017.

[46] Ryo 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*, pages 235–249. Springer, 2005.

[47] Pian Yu and Dimos V Dimarogonas. Time-constrained leader-follower multiagent task scheduling and control synthesis. *IEEE Transactions on Control of Network Systems*, 9(1):367–379, 2021.

[48] Sebastián A Zanlongo, Peter Dirksmeier, Philip Long, Taskin Padir, and Leonardo Bobadilla. Scheduling and path-planning for operator oversight of multiple robots. *Robotics*, 10(2):57, 2021.

[49] Shaobo Zhang, Yi Chen, Jun Zhang, and Yunyi Jia. Real-time adaptive assembly scheduling in human-multi-robot collaboration according to human capability. In *2020*

IEEE International Conference on Robotics and Automation (ICRA), pages 3860–3866. IEEE, 2020.

[50] Kuanhao Zheng, Dylan F Glas, Takayuki Kanda, Hiroshi Ishiguro, and Norihiro Hagita. Supervisory control of multiple social robots for navigation. In *ACM/IEEE Int. Conf. on Human-Robot Interaction*, pages 17–24, 2013.

APPENDICES

Appendix A

Properties of Exopnential and Hypoexponential Distribution

A.0.1 Exponential Distribution

Suppose that X_1 and X_2 are independent random variables with distribution of $Exp(\lambda_1)$ and $Exp(\lambda_2)$, the density function of $X = max\{X_1, X_2\}$ can be determined as follow.

$$P(X_1 \leq x) = F_1(x) = \int_0^x f_1(x)dx.$$

$$P(X_2 \leq x) = F_2(x) = \int_0^x f_2(x)dx.$$

As X_1 and X_2 are indpendent,

$$P(X = max\{X_1, X_2\} \leq x) = P(X_1 \leq x)P(X_2 \leq x).$$

$$F_X(x) = F_1(x)F_2(x) = (1 - e^{-\lambda_1 x})(1 - e^{-\lambda_2 x}).$$

Then, the density function for X is

$$f_X = \frac{d}{dx}F_X(x) = \lambda_1 e^{-\lambda_1 x} + \lambda_2 e^{-\lambda_2 x} - (\lambda_1 + \lambda_2)e^{-(\lambda_1+\lambda_2)x}.$$

The expectation of $X = max\{X_1, X2\}$ is

$$\mathbb{E}(X) = \int_0^\infty xP(X = x)dx.$$

$$= \int_0^\infty x f_X dx.$$

$$= \int_0^\infty x(\lambda_1 e^{-\lambda_1 x} + \lambda_2 e^{-\lambda_2 x} - (\lambda_1 + \lambda_2) e^{-(\lambda_1+\lambda_2)x}) dx.$$

$$= \frac{1}{\lambda_1} + \frac{1}{\lambda_2} - \frac{1}{\lambda_1 + \lambda_2}.$$

A.0.2 Hypoexponential Distribution

Hypoexponential distribution is defined as the sum of independent exponential random variables with different rate λ (generalized Erlang Distribution). Suppose there are G independent exponentially distributed random variables X_g, $g \in \mathcal{G} = \{1, \ldots, G\}$, with rate parameters $\{\lambda_1, \ldots, \lambda_G\}$, $\lambda_i \neq \lambda_j$, given $i \neq j$. The random variable $X = \sum_{g=1}^{G} X_g$ follows Hypoexponential Distribution.

Hypoexponential distribution has the probability distribution function as the weighted average of the individual exponential distribution. The weight for the g^{th} element is denoted by W_g, and expressed in Equation A.2.

$$f(X = x) = \sum_{g=1}^{G} W_g \cdot (\lambda_g e^{-\lambda_g x}), \tag{A.1}$$

$$W_g = \prod_{j \in \mathcal{G} \setminus \{g\}} \frac{\lambda_j}{\lambda_j - \lambda_g}. \tag{A.2}$$

The cumulative distribution function $F(X)$ for Hypoexponential is as follow:

$$P(X \leq x) = F(x) = \int_0^x f(x) dx, \tag{A.3}$$

$$= \sum_{g=1}^{G} W_g \cdot (1 - e^{-\lambda_g x}).$$

Suppose that Y_1 and Y_2 are two independent Hypoexponentially distributed random variables. Y_1 is the sum of independent exponential random variables with rate $\{\lambda_{g_1}, \ldots, \lambda_{g_G}\}$, and Y_2 is the sum of independent exponential random variables with rates $\{\lambda_{h_1}, \ldots, \lambda_{h_H}\}$.

Let $Y = max\{Y_1, Y_2\}$. Then the cumulative distribution function for Y is the multiplication of cumulative distribution functions of Y_1 and Y_2.

$$F_Y(y) = F_{Y_1}(y) \cdot F_{Y_2}(y), \tag{A.4}$$

$$= \left(\sum_{i=1}^{G} W_{g_i}(1 - e^{-\lambda_{g_i} y}) \right) \left(\sum_{j=1}^{H} W_{h_j}(1 - e^{-\lambda_{h_j} y}) \right). \tag{A.5}$$

We can obtain the probability density function $f(Y)$ by taking the first order derivative of $F(Y)$ as shown in Equation A.6.

$$f_Y(y) = \frac{d}{dy} F_Y, \tag{A.6}$$

$$= \left(\sum_{i=1}^{G} W_{g_i} \lambda_{g_i} e^{-\lambda_{g_i} y} \right) \left(\sum_{j=1}^{H} W_{h_j}(1 - e^{-\lambda_{h_j} y}) \right)$$

$$+ \left(\sum_{i=1}^{G} W_{g_i}(1 - e^{-\lambda_{g_i} y}) \right) \left(\sum_{j=1}^{H} W_{h_j} \lambda_{h_j} e^{-\lambda_{h_j} y} \right).$$

Finally, the expectation of Y is the first moment.

$$\mathbb{E}(Y) = \sum_{i=1}^{G} \sum_{j=1}^{H} W_{g_i} \cdot W_{h_j} \cdot \left(\frac{1}{\lambda_{g_i}} + \frac{1}{\lambda_{h_j}} - \frac{1}{\lambda_{g_i} + \lambda_{h_j}} \right) \tag{A.7}$$

A.0.3 Expand for k Hypoexponential Variables

Suppose that $\{Y_1, \ldots, Y_K\}$ are K independent Hypoexponentially distributed random variables. Y_1 is the sum of independent exponential random variables with rates $\{\lambda_{g11}, \ldots, \lambda_{g1G^1}\}$, and Y_k is the sum of independent exponential random variables with rates $\{\lambda_{gk1}, \ldots, \lambda_{gkG^k}\}$.

Let $Y = max\{Y_1, \ldots, Y_k\}$. Then, the cumulative distribution function for Y is the multiplication of cumulative distribution functions of Y_1 to Y_k.

$$F_Y(y) = F_{Y_1}(y) \cdot F_{Y_2}(y) \ldots \cdot F_{Y_k}(y), \tag{A.8}$$

$$= \prod_{i=1}^{k} \left(\sum_{j=1}^{G^i} W_{gij}(1 - e^{-\lambda_{gij} y}) \right). \tag{A.9}$$

To simplify the expression, let's call each term in the above product

$$P_i = \left(\sum_{j=1}^{G^i} W_{gij}(1 - e^{-\lambda_{gij} y}) \right). \tag{A.10}$$

We can obtain the probability density function $f(Y)$ by taking the first derivative of $F(Y)$ as shown in the following equation.

$$f_Y(y) = \frac{d}{dy} F_Y, \tag{A.11}$$

$$= \sum_{i=1}^{k} \left(\left(\sum_{j=1}^{G^i} W_{gij} \lambda_{gij} e^{-\lambda_{gij} y} \right) \right. \tag{A.12}$$

$$\left. \cdot \prod_{l \in \mathcal{K} \setminus i} \left(\sum_{l=1}^{G^l} W_{gil}(1 - e^{-\lambda_{gil} y}) \right) \right) \tag{A.13}$$

The expression of expectation of Y can be obtained by finding first moment of $f_Y(y)$