Master's thesis

Uncertainty-aware Human-Robot Collaboration using scheduling and reactive control

Marina Ionova

May 2023

Supervisor: Dr.-Ing. Jan Kristof Behrens

Czech Technical University in Prague
Faculty of Electrical Engineering, Department of Cybernetics

MASTER'S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Ionova Marina**	Personal ID number:	**474683**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Cybernetics**		
Study program:	**Cybernetics and Robotics**		
Branch of study:	**Cybernetics and Robotics**		

II. Master's thesis details

Master's thesis title in English:

Uncertainty-Aware Human-Robot Collaboration Using Scheduling and Reactive Control

Master's thesis title in Czech:

Kolaborace člověka a robota s ohledem na nejistotu pomocí plánování a reaktivního řízení

Guidelines:

This thesis aims to utilize task and motion scheduling [1] in industrial Human-Robot Collaboration (HRC) scenarios. Teams with a shared goal (e.g., assembling a product) must act in a coordinated way to ensure correct task execution and safety (e.g., avoiding collisions). Today, we can find efficient schedules, but in the face of execution uncertainty these become incongruent with the original assumptions about the world state. A flexible and robust control architecture is required to effectively use scheduling in HRC.
Instructions:
1) Define several HRC scenarios for collaborative workcells. Derive a simulation model to test scheduling and execution architectures systematically.
2) Review the literature (start with [2,3,4]) and summarize the different approaches. Implement a baseline method from the literature for simulation experiments. Consider the human factors of working in such an environment [4].
3) Design and implement an uncertainty (e.g., task durations, task rejection, task outcome) aware HRC planning and execution method (e.g., combine scheduling with a behavior tree-based control engine). Integrate the method into a collaborative workcell (Franka Emika Panda robot, HTC vive VR controllers, cameras).
4) Verify the method in simulation studies. Show, for example, the benefit (e.g., higher robustness, lower makespan) of utilizing quantitative knowledge about uncertainty (e.g., distribution of task duration or other parameters from the simulation model developed in step 1). Show qualitative results of the baseline and the developed method using the real workcell.

Bibliography / sources:

[1] J. K. Behrens, "Integrated task and motion scheduling for flexible manufacturing," Universität Bremen, 2022. doi: 10.26092/ELIB/1801.
[2] Marco Faroni, Manuel Beschi, Stefano Ghidini, Nicola Pedrocchi, Alessandro Umbrico, et al.. A Layered Control Approach to Human-Aware Task and Motion Planning for Human-Robot Collaboration. IEEE International Conference on Robot and Human Interactive Communication (RO-MAN), Aug 2020, Naples (virtual), Italy.
[3] M. Iovino, E. Scukins, J. Styrud, P. Ögren, and C. Smith, "A survey of Behavior Trees in robotics and AI," Robotics and Autonomous Systems, vol. 154, p. 104096, Aug. 2022, doi: 10.1016/j.robot.2022.104096.
[4] S. Li et al., "Proactive human–robot collaboration: Mutual-cognitive, predictable, and self-organising perspectives," Robotics and Computer-Integrated Manufacturing, vol. 81, p. 102510, Jun. 2023, doi: 10.1016/j.rcim.2022.102510.

Name and workplace of master's thesis supervisor:

Dr. Jan Kristof Behrens, MSc. Robotic Perception CIIRC

Name and workplace of second master's thesis supervisor or consultant:

Date of master's thesis assignment: **02.02.2023** Deadline for master's thesis submission: **26.05.2023**

Assignment valid until: **22.09.2024**

Dr. Jan Kristof Behrens, MSc.	prof. Ing. Tomáš Svoboda, Ph.D.	prof. Mgr. Petr Páta, Ph.D.
Supervisor's signature	Head of department's signature	Dean's signature

III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce her thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student's signature

Acknowledgement

I express my profound gratitude to my supervisor, Jan Kristof Behrens, for presenting me with challenging opportunities that have contributed to my profound comprehension of the problem at hand. I am sincerely appreciative of the invaluable experience and knowledge that I have acquired throughout the course of this thesis. Additionally, I extend my heartfelt thanks to the entire Imitrob team for their support, constructive feedback, and continuous motivation within the realm of research.

I am profoundly indebted to my family, whose unwavering belief in my potential granted me the opportunity to pursue higher education and steadfastly supported me throughout my academic journey. Furthermore, I would like to extend special recognition to my boyfriend, Pavel, for his unwavering care, encouragement, and support.

Once again, I would like to express my deepest appreciation to all those who have contributed to my academic and personal growth.

Declaration

Abstrakt

Tato práce zkoumá potenciál kolaborace robota s člověkem pro zvýšení přizpůsobivosti a flexibility výrobního procesu využitím silných stránek lidí i robotů. Integrace kolaborativních buněk do skutečných výrobních systémů je však v současné době náročná vzhledem k mnoha nejistotám a nepoužitelnosti stávajících pevně specifikovaných metod řízení robotů. K řešení těchto problémů tato práce navrhuje inovativní přístup k online rozvrhování v kolaboraci robota s člověkem pomocí programování s omezením a stromu chování, speciálně navrženým tak, aby zohledňoval nejistoty a nedeterminismus vnesené zapojením člověka. Metoda se řídí strukturovaným rámcem, který zahrnuje definování nejistot, formulaci problému rozvrhování, implementaci algoritmu přidělování úloh na základě rozvrhu a zohlednění lidského kognitivního vnímání a etických faktorů. Byly provedeny simulace pro vyhodnocení účinnosti a robustnosti navržené metody. Výsledky ukazují příznivou výkonnost, pokud jde o makespan (dobu trvání celého výrobního procesu) a stabilitu při konfrontaci se změnami prostředí. Metoda byla navíc úspěšně implementována s využitím reálného robota Franka Emika Panda.

Klíčová slova

kolaborace člověka s robotem, rozvrhování, optimalizace, neurčitost, přiřazování úkolů, programování s omezením, reaktivní řízení, strom chování

Abstract

This thesis explores the potential of Human-Robot Collaboration (HRC) to enhance customization and flexibility in the production process by harnessing the unique strengths of both humans and robots. However, integrating collaborative cells into actual production systems is currently challenging due to numerous uncertainties and the inapplicability of existing fixed robot control methods. To address these issues, this work proposes an innovative approach to online scheduling in HRC using Constraint Programming (CP) and Behavior Tree (BT), specifically designed to account for uncertainties and non-determinism introduced by human involvement. The method follows a structured framework that involves defining uncertainties, formulating a scheduling problem, implementing a task allocation algorithm based on schedule, and considering human cognitive perceptions and ethical factors. Simulations were conducted to evaluate the effectiveness and robustness of the proposed method. Results indicate that it exhibits favorable performance regarding makespan (the duration of the entire production process) and stability when confronted with environmental changes. Furthermore, the method was successfully implemented using the Franka Emika Panda robot in a real-world setup.

Keywords

human-robot collaboration, scheduling, optimization, uncertainty, task allocation, constraint programming, reactive control, behavior tree

Contents

1 Introduction 1
- 1.1 Aim of this work . . . 2
- 1.2 Thesis structure . . . 2

2 Related Work 4
- 2.1 State-of-the-art . . . 4
  - 2.1.1 Task allocation methods . . . 4
  - 2.1.2 Robot control . . . 5
  - 2.1.3 Workers mental health . . . 5
    - Cognitive perception . . . 6
- 2.2 Thesis contribution . . . 6

3 Terminology and Scenario Concepts 7
- 3.1 Tasks and Task Classification . . . 7
- 3.2 Scenarios . . . 10
  - 3.2.1 Ethical considerations . . . 12

4 Scheduling with uncertainty and task allocation control 13
- 4.1 Uncertainty model . . . 13
  - 4.1.1 Uncertainty in task durations . . . 13
  - 4.1.2 Uncertainty in human decision-making . . . 14
- 4.2 Scheduling problem . . . 15
  - 4.2.1 Soft constraints . . . 19
- 4.3 Control method . . . 19

5 Implementation 22
- 5.1 Software architecture . . . 22
- 5.2 Modeling and solving constraint programming . . . 23
  - 5.2.1 Updating model variables in constraint programming . . . 26
  - 5.2.2 Rescheduling estimation . . . 26
  - 5.2.3 Probabilistic simulation . . . 26
- 5.3 Real robot setup . . . 29
  - 5.3.1 Real robot system architecture . . . 29
  - 5.3.2 Reactive control node . . . 29
  - 5.3.3 User interaction via a Graphical User Interface . . . 32

6 Experiments and Results 33
- 6.1 Simulation . . . 33
  - 6.1.1 Experimental setup . . . 33
  - 6.1.2 Results . . . 35
    - Makespan . . . 36
    - Predictability . . . 39
- 6.2 Real robot experiments . . . 40
  - 6.2.1 Use case . . . 40
  - 6.2.2 Robot and schedule restiveness . . . 42

7 Conclusion 44
- 7.1 Future work . . . 44

Bibliography 46

Acronyms

AI	Artificial Intelligence.
ALBP	Assembly Line Balancing Problem.
BT	Behavior Tree.
CP	Constraint Programming.
CSP	Constraint Satisfaction Problem.
FSM	Finite-State Machine.
GUI	Graphical User Interface.
HRC	Human-Robot Collaboration.
HRI	Human-Robot Interaction.
IoT	Internet of Things.
LP	Linear Programming.
OOP	Object-Oriented Programming.
ROS	Robot Operation System.
WBS	Work Breakdown Structure.

Chapter 1

Introduction

Let us dive into the history of manufacturing. Previously, manufacturers produced goods only to individual customer specifications. Nevertheless, the advent of mass production methods allowed a more economical approach to production through automation. This shift in manufacturing methodology allowed for greater efficiency and economies of scale [1]. In the modern world, manufacturers can achieve the benefits of mass production, such as economies of scale, while still offering customers the ability to customize products to their individual needs by utilizing digital manufacturing methods. Mass customized and mass personalized production is one of the main goals of Industry 4.0 based on advanced technologies such as the Internet of Things (IoT), 3D printing, Artificial Intelligence (AI), and Human-Robot Collaboration (HRC) [2]. These technologies allow for real-time monitoring and control of manufacturing processes, enabling companies to respond quickly to changing customer needs and preferences. These tools are only means to accomplish the objective and must still be studied and improved.

One key aspect of product customization is the ability to quickly and easily modify production lines to meet the specific needs of individual customers. For example, a company producing custom-made furniture may be able to quickly adjust the size, shape, and materials used to create a unique piece for a specific customer. HRC is one way of combining the flexibility and dexterity of human worker with the efficiency gains of factory automation. By working alongside robots, human workers can leverage their unique skills and knowledge while benefiting from automated processes' speed and efficiency. It enables companies to produce customized products at scale while maintaining the high quality and attention to detail that customers expect.

Collaborative robots in factories often go unused because their integration into flexible manufacturing facilities is more difficult than programming traditional stationary robots. This is mainly due to the uncertainty resulting from Human-Robot Interaction (HRI). Robots are predictable, but humans are not. Humans can make decisions based on emotions, intuition, and personal preferences that machines cannot replicate. This causes planning uncertainties that modern systems must cope with and respond quickly and safely to humans. This work aims to fill this gap by developing a scheduling method that considers human uncertainties and how it can be adapted in case of changes. The second key aspect is developing a reactive method of controlling the robot, which allows it to act intelligently and understandably for a human.

Another crucial aspect of this thesis is representing the task selection process for human workers. While this may introduce additional uncertainty, it can also aid in customizing the work environment according to the individual preferences of workers. This, in turn, can reduce stress levels and eliminate work monotony, which is also one of Industry 4.0's goals.

1.1 Aim of this work

This work addresses the challenges posed by the non-deterministic and dynamic nature of the HRC environment. The goal is to achieve efficient scheduling approaches in a HRC setting, where the human and robot work towards a common goal. However, the environment's limited controllability and inherent uncertainty make a time-driven robot behavior inappropriate. To address this, planning and acting must consider uncertainty, and the robot must continuously monitor and update foundational assumptions to extend its horizon based on demand.

The objective is subdivided into several sub-goals to provide a structured approach and allow evaluation of the outcomes at each stage.

- **Structuring the task assignment of the collaboration.** To minimize inaccuracies during execution, it is crucial to formulate the problem precisely and clearly. A key aspect of achieving this goal is establishing a job hierarchy and systematically analyzing the interrelationships between tasks. The desired outcome of this research endeavor is a comprehensive set of information about each task necessary for its successful execution, as well as a variety of scenarios that illustrate different types of task dependencies.

- **Identify and represent the uncertainties for the involvement of humans.** The objective is to identify uncertainties related to action controllability and suggest modeling methods based on observability. The next step is to decide how to incorporate this model data into the scheduling process.

- **Create a scheduling problem statement and provide a solution.** Based on the outcomes of the previous objectives, the task requirements and system uncertainties are analyzed to schedule the completion of tasks. A decision-making algorithm for task allocation is proposed, which is based on the schedule and takes into account the uncertainties identified. In the task allocation process, if there are any differences from the original schedule, adjustments are made, and rescheduling is performed. To evaluate the efficiency and reliability of this approach, simulations should be conducted, with a focus on assessing the benefits of the proposed method compared to a baseline method, such as lower makespan and predictability.

- **Implement a task allocation algorithm based on scheduling and integrate it into a real robotic workspace.** This objective also involves creating a reactive control system that can quickly and safely respond to environmental changes, which is essential for collaborative robot systems. The result should be a demo to demonstrate the ability of the system to react to changes in the environment and schedule.

- **Consider the cognitive perception of humans and ethical considerations** when developing the system. While high production autonomy is vital in Industry 4.0, ensuring humans' physical and mental well-being is equally essential. Therefore, any research related to HRC should also address this goal.

This work presents a promising approach for HRC management. The outcome is an algorithm that allocates tasks according to a schedule while considering the uncertainties introduced by human involvement. The effectiveness and robustness of the suggested solution have been verified through experiments in simulation and with a real robot setup.

1.2 Thesis structure

The thesis is structured into seven chapters, where the Chapter 1 provides an introduction to the work, including the aim of the study and the thesis structure. Chapter 2 presents the related work, including state-of-the-art research and the contribution of this thesis to the field. Chapter 3 introduces the terminology and scenario concepts used in the study, including task classification and ethical considerations related to the task assignment. Chapter 4 discusses the scheduling problem in uncertain environments, including the uncertainty model, scheduling problem, and control method. Chapter 5 describes the implementation of the proposed approach, including the structure of Object-Oriented Programming (OOP), Constraint Programming (CP) modeling and solving. The latter section of this chapter discusses the concept of reactive control for robot and how it was applied in a practical robotic system. Finally, Chapter 6 presents the experiments and results, including experiments in simulation and real robot demo. The conclusion and discussion about further work are in Chapter 7.

Chapter 2

Related Work

To accomplish the objectives outlined in Chapter 1, the initial phase is to assess the present status of research and accomplishments in the field of HRC in various directions, such as task allocation and scheduling problems with uncertainties, reactive methods for robot control, and the impact of robot collaboration on human mental states and stress levels. The first section of this chapter provides more detailed information on literature from each field, while the second section outlines the thesis's specific contributions.

2.1 State-of-the-art

HRC continues to face numerous unresolved challenges. This study specifically addresses online task scheduling and robot control in uncertainty and a dynamically changing environment. A crucial aspect of addressing these challenges is incorporating human perception of the collaboration process and prioritizing intuitive control over task allocation and robot movements.

2.1.1 Task allocation methods

Collaborative assembly, a widely-invested example of HRC in Industry 4.0, is cost-effective because it combines each agent's capabilities, such as strength, repeatability, clarity, and analytical abilities [3] [4]. However, in cases where both agents perform tasks equally well, it is important to determine who should perform the task and when which is known as the Assembly Line Balancing Problem (ALBP). The ALBP can be solved through two main solutions: online task allocation based on agent availability or a predetermined schedule with the possibility of dynamic adjustment. Criteria such as agent experience, decision-making structure, and minimizing execution time can be used to evaluate the most suitable task for the first solution [5] [6] [7]. These algorithms do not require extensive preparation and can begin job execution immediately without wasting time planning the entire process. They can adapt to the current situation as events unfold. However, a significant drawback is their inability to accurately predict the duration of the work, which is essential for effective planning of the production line. Additionally, the lack of planning limits the ability to assess the optimality of the choices made by the algorithms.

The second way is scheduling. Scheduling is a process that involves two primary stages: schedule generation and scheduling revisions [8]. The schedule generation is the process of forecasting and determining the starting and ending times for production tasks based on predefined requirements and constraints before the production process begins. Schedule methods such as stochastic scheduling or robust optimization are typically employed when job durations are stochastic. Popular techniques for generating schedules include genetic algorithms [9][10][11], Petri Nets [12][13], state-space analysis[14] [15], and linear programming [16]. These methods help to ensure that the schedules generated are feasible and efficient.

The process of scheduling revisions is reactive and involves monitoring the schedule execution and dealing with any unforeseen events that may occur during the process. This type of scheduling is used specifically in cases where it is necessary to change the original scheduling policy or create a new one in response to uncertainty. Nonlinear programming algorithms such as bilinear programming [17] or constraint programming [18] [19] [20] and advanced genetic programming methods such as NSGA [21], are used for this purpose. Comparative experiments have shown that constraint programming has performed better than the widely used genetic algorithm in scheduling. Although the genetic algorithm is popular, its inferior performance suggests that constraint programming may be a more effective approach[19].

Unlike the first group of methods, scheduling can predict the makespan. However, the presence of humans in the workspace adds complexity to the problem, as it depends on several factors. One drawback of scheduling is the high computational cost and the algorithm's time complexity, which increases with the number of tasks and agents.

To reduce the level of uncertainty by predicting human actions, various learning methods are utilized. These methods include Markov processes, Bayesian sequential decision-making, and neural network training [22, 23, 24]. These techniques enable the anticipation of human actions by analyzing the patterns and trends in their behavior, thereby facilitating better adaptation of the system in case of changes.

2.1.2 Robot control

Finite-State Machines (FSMs) were the primary method for controlling robot movements for a long time; however, in recent years, Behavior Trees (BTs) have gained increasing attention as an alternative approach [25]. One of the primary benefits of BTs is their flexibility and modularity, which allows for the creation of complex behaviors by composing simpler behaviors. This makes it easier to modify and extend the robot's behavior without having to restructure the entire control system. Furthermore, the flexibility of behavior trees enables alternative approaches to learning and extending them [26] [27]. In addition, BTs provide a more natural and intuitive way to program complex robot behaviors compared to FSMs.

BTs are particularly well-suited to HRC, where the robot needs to adapt its behavior to the changing needs and preferences of the human collaborator. The reactive nature of BTs allows the robot to quickly adapt to environmental changes, providing a more efficient and intuitive approach to robot behavior control.

2.1.3 Workers mental health

Studies have been conducted to investigate the impact of Human-Robot Interaction (HRI) on human stress levels. Factors contributing to stress and tension have been identified, including the robot's size, speed, trajectories, and social touching of the robot [28]. Monotonous work has also been found to affect physical and mental health, decreasing productivity and satisfaction [29].

Furthermore, the level of trust, negotiation, and ethics in human-robot interactions has been extensively studied in research. Trust is a crucial factor in the success of HRC, as it affects how humans perceive and interact with the robot [30]. Negotiation and ethical considerations are also important in ensuring that the robot's behavior aligns with human values and expectations, particularly in sensitive and complex situations.

Cognitive perception

Human cognitive perception plays a crucial role in interaction system design. It determines how users perceive, interpret, and interact with the system. Human perception is influenced by a wide range of factors, including visual, auditory, and haptic cues, as well as cognitive and emotional factors. Understanding these factors is essential for designing effective and user-friendly interaction systems [31] [32].

The relationship between trust in a robot's actions and the stress level experienced during the interaction is directly proportional. Through a meta-analysis, Hancock and colleagues reviewed multiple factors influencing trust in HRI [33]. They found that factors related to the robot, specifically performance-related, strongly correlated with trust. These performance-based factors included the level of automation, behavior, dependability, reliability, and predictability of the robot. When humans know and understand the robot's movements, they are more likely to adapt to them intuitively. A transparent system, when the user knows what is happening and why enables more efficient operation since the robot will not have to stop and change its motions constantly [34]. As trust in the robot's actions increases, the stress level experienced during the interaction may decrease. On the other hand, if trust is low and a person is uncertain about the robot's actions, this may increase their stress level and the likelihood of accidents or errors. These systems need to keep humans well informed about the intentions and actions of the robot. Too much trust in systems also has its drawbacks.

Khavas, in his work, summarizes the criteria on which trust is based [35]. The main of them is the comprehensibility and intuitiveness of the decision-making logic of the system.

2.2 Thesis contribution

The field of HRC has a broad spectrum of challenges. This thesis focuses only on some of the problems mentioned earlier, which align with the goals stated in Chapter 1.

- This thesis introduces an approach to task distribution between a robot and a human based on schedule with the ability to provide choice of task for human, which is a relatively unexplored area. Although this approach increases uncertainty, it offers a new form of human-robot interaction for the industry of future.

- The proposed method involves a reactive schedule that considers the individual's preferences and estimates the approximate duration of tasks to create a schedule and predict the expected completion time.

- The thesis introduces a decision-making algorithm in the form of BT for a robot, enabling it to adapt its actions in response to environmental changes and avoid collisions with humans.

Chapter 3

Terminology and Scenario Concepts

This chapter describe the importance of precise job descriptions for any production process, particularly when automated. The chapter covers dividing job requests into individual tasks and the importance of correctly sequencing them to achieve the desired outcome. It introduces the necessary terms and hierarchy of task division and the types of dependencies between them. The chapter also delves into the ethics of assigning human tasks within automated systems. The chapter aims to provide insight into the structure and organization of task assignments in automated systems while considering the ethical implications.

3.1 Tasks and Task Classification

The Work Breakdown Structure (WBS) is vital to any production process. It breaks down the process into smaller, more manageable tasks with dependencies between them, allowing for the efficient and effective completion of complex designs. This structure becomes increasingly complex in the case of multi-agent production processes. It can benefit significantly from using standardized terminology, such as that presented in Zlot's dissertation [36]. In his dissertation work, he proposed a systematic division of the tasks. For easy understanding, let us look at the main definitions directly with an example.

Definition 1. An **elemental** (or atomic) **task** is a task that is not decomposable.
Example: take the phone, open an application, etc.

Definition 2. A task t is **decomposable** if it can be represented as a set of subtasks σ_t for which satisfying some specified combination (ρ_t) of subtasks in σ_t satisfies t. The combination of subtasks that satisfy t can be represented by a set of relationships ρ, including constraints between subtasks or rules about which or how many subtasks are required. The pair (σ_t,ρ_t) is also called a decomposition of t. The term decomposition can also refer to the process of decomposing a task.
Example: You need to order a taxi. To do this, you must take the phone, unlock it, open the application, set the address, etc.

Definition 3. A task t is **multiply decomposable** if there is more than one possible decomposition of t.
Example: You can order a taxi through the app, call or catch one on the street.

Definition 4. A **decomposable simple task** is a task that can be decomposed into elemental or decomposable simple subtasks, provided that there exists no decomposition of the task that is multi[agent]-allocatable.
Example: You must take your phone to order a taxi through the app. However, if you ask a

friend to submit it, the "order a taxi" task will no longer be a decomposable simple task, as two people are already involved.

Definition 5. A **simple task** is either an elemental task or a decomposable simple task.
Example: You have your phone in your hands; now you must unlock it, open the app, and place your order. The "placing an order" task is broken down into subtasks such as setting the current and target addresses, selecting the payment method, and confirming the order. These are all elemental tasks, and "placing an order" is a decomposable simple task that is also multiply decomposable because it makes no difference whether you set the current address first or the target address.

Definition 6. A task t is **fully decomposable** if a set of simple subtasks can be derived within a finite number of decomposition steps. Such a decomposition (containing only simple subtasks) is called a **full decomposition** of t.
Example: Ordering a taxi can be described as a finite sequence of actions. The same cannot be said for writing a thesis, which can be corrected and improved countless times before the deadline.

Definition 7. A **compound task** t is a task that can be decomposed into a set of simple or compound subtasks with the requirement that there is exactly one fixed full decomposition for t (i.e., a compound task may not have any multiply decomposable tasks at any decomposition step).
Example: If the taxi ordering application is given a precise sequence of ordering information, this "placing an order" task is no longer multiply decomposable and becomes a component.

Definition 8. A **complex task** is a multiply decomposable task for which there exists at least one decomposition that is a set of multi[agent]-allocatable subtasks. Each subtask in a complex task's decomposition may be simple, compound, or complex.
Example: A task becomes complex as soon as another agent can complete any of the sub-tasks. You can ask your friend to order a taxi.

Definition 9. Given a team of agents, a task is **allocatable** if it is possible to assign the task to a another agent with the appropriate capabilities and resources to achieve it. *Example: You are going to take a taxi with your friend and choose who will order the taxi, given that each of you has an app on your phone and all the information needed.*

This decomposition structure into subtasks applies to the definition of a collaboration *job* in the context of this work. Job and all other components are called *tasks* according to the Zlot classification [36]. Supplementary terminology was introduced to provide a more comprehensible and lucid illustration of the task structure utilized in this study. The diagram Figure 3.1 illustrates the breakdown of a job, with each segment allocated to distinct types in the aforementioned classification. A detailed mapping of those terms to Zlot's nomenclature is indicated in the table Table 3.1 and depends on several factors. The job consists of tasks. These tasks can be classified into *allocatable* and *non-allocatable* types, with the former being performed by either a human or robot. At the same time, the latter can only be assigned to a specific agent. Task allocation can be based on factors such as location, physical abilities, analytical skills, and coordination [37] [38]. This approach allows the most use of both the robot and human capabilities. This work takes the allocability of tasks as given and does not explore causality, which tasks may or may not be. The presence of allocatable tasks makes the job a *complex task*.

In this work's context, the tasks are further broken down into actions defined differently for robots and humans. Following the definition, actions are *elemental tasks*. For robots, a task is

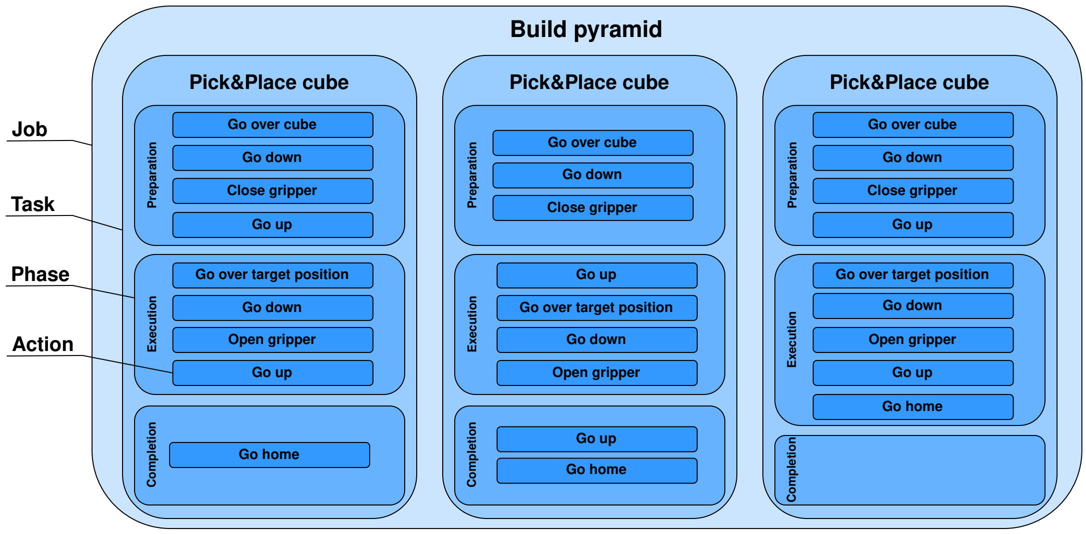


Figure 3.1 This diagram visualizes the hierarchy of dividing job into tasks, phases, and actions. In the columns, there are several possible distributions of actions into phases. This indicates the abstractness of the concept of phase, which can be defined differently depending on when the second agent is allowed to start the execution phase in the case of two dependent tasks.

Name	**Execution by**		**Zlot's nomenclature [36]**	**Example**
	Human	Robot		
Job	x		Complex task	Build a pyramid
		x	Simple task	
	x	x	Complex task	
Task	x		Complex task	Pick&Place cube
		x	Compound task	
Phase	x		Complex task	Preparation
		x	Compound task	
Action		x	Elemental Task	Go over cube

Table 3.1 The table shows the correspondence between the breakdown job structure and types of tasks defined in Zlot's work [36].

a precise sequence of actions leading to the execution of it, and it is classified as a *simple task*. For a human, tasks are assumed to be a sequence of actions necessary to perform them but can vary depending on human behavior. For instance, when a person should put cubes C and D, he may take one cube at a time or take both simultaneously and put them after the robot, resulting in a different sequence of human actions that lead to the same outcome. In other words, the human task is *multiply decomposable* and corresponds to the definition of a *complex task*.

Until this paragraph, the phases of the task were not discussed because it is abstract, and its boundaries alter depending on the sequence of actions. An example Figure 3.1 is the Pick&Place operation, where there are different ways to divide the actions into phases. This division was created to avoid idle time for the second agent in a task that depends on completing the first task and avoid collisions by controlling the access to the shared area. By dividing the task into phases, the second agent can begin conditional preparation for the execution of its task during the execution of the first task, reducing idle time. This condition is described in more detail in Section 4.2 and shown in Figure 4.2. The boundaries of these phases depend on the managed resource and can be explored separately to optimize the division of tasks. This work uses the division shown on the right in the Figure 3.1.

Breaking down tasks into categories is essential for developing a comprehensive approach to address task scheduling problems while also considering the nuances of human-aware planning. Consequently, the hierarchical separation of tasks is the foundation for stable and resilient scheduling and organization of the complete manufacturing process.

3.2 Scenarios

After understanding the job structure, analyzing dependencies between tasks and creating basic collaboration scenarios is possible. The iTax, a classification of task dependencies in multi-robotic systems proposed by Korsah [39], is based on the definitions provided in the previous section. iTax identifies four main types of task dependencies:

- **No Dependencies**: These are task allocation problems with simple or compound tasks that have independent agent-task utilities. That is, the effective utility of an agent for a task does not depend on any other tasks or agents in the system.

- **In-Schedule Dependencies**: These are task allocation problems with simple or compound tasks for which the agent-task utilities have intra-schedule dependencies. That is, the effective utility of an agent for a task depends on what other tasks that agent is performing. Constraints may exist between tasks on a single agent's schedule or might affect the overall schedule of the agent.

- **Cross-Schedule Dependencies**: These are task allocation problems with simple or compound tasks for which the agent-task utilities have inter-schedule dependencies (in addition to any in-schedule dependencies). That is, the effective utility of an agent for a task depends not only on its own schedule but also on the schedules of other agents in the system. For this class, allowable dependencies are "simple" dependencies in that the task decomposition can be optimally pre-determined prior to task allocation. Constraints may exist between the schedules of different agents.

- **Complex Dependencies**: These are task allocation problems for which the agent-task utilities have inter-schedule dependencies for complex tasks (in addition to any in-schedule and cross-schedule dependencies for simple or compound tasks). That is, the effective utility of an agent for a task depends on the schedules of other agents in the system in a manner that is determined by the particular task decomposition that is ultimately chosen. Thus, the optimal task decomposition cannot be decided prior to task allocation but must be determined concurrently with task allocation. Furthermore, constraints may exist between the schedules of different agents.

While iTax is designed for multi-robotic systems, the task performed by a human is considered a complex task, according to the previous section. Therefore, if at least one task exists for a human, the entire task allocation problem is considered complex. However, there is an opportunity to make a relaxation. If the task for a human cannot be broken down into actions, it is categorized as an elemental task and does not affect the type of task dependencies.

The allocatable tasks are another factor that contributes to a *Complex dependency*. In the context of optimization of the job execution time, this ability of task is of utmost importance, and therefore, its definition remains unaltered.

To conduct comprehensive testing, two types of scenarios were created. The first type only includes non-allocatable tasks, which are further divided into three cases (scenario 1 with In-Schedule dependency and Cross-Schedule dependency in scenarios 3 and 5 in Table 3.2). The

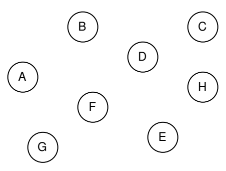
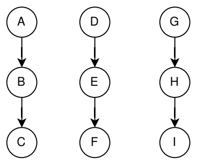
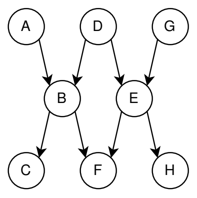
Scenario	Depen-dencies	Task description					Dependency graph
		ID	Task	Object	Agent	Conditions	
1	In-Schedule	1	Pick&Place	A	Human	-	
		2	Pick&Place	B	Robot	-	
		⋮	⋮	⋮	⋮	⋮	
2	Complex	1	Pick&Place	A	Human	-	
		2	Pick&Place	B	Allocatable	-	
		⋮	⋮	⋮	⋮	⋮	
3	Cross-Schedule	1	Pick&Place	A	Human	-	
		2	Pick&Place	B	Robot	Task 1	
		3	Pick&Place	C	Human	Task 2	
		⋮	⋮	⋮	⋮	⋮	
4	Complex	1	Pick&Place	A	Allocatable	-	
		2	Pick&Place	B	Robot	Task 1	
		3	Pick&Place	C	Allocatable	Task 2	
		⋮	⋮	⋮	⋮	⋮	
5	Cross-Schedule	1	Pick&Place	A	Human	-	
		2	Pick&Place	B	Robot	Task 3	
		3	Pick&Place	C	Robot	Task 1, 2	
		4	Pick&Place	D	Human	-	
		⋮	⋮	⋮	⋮	⋮	
6	Complex	1	Pick&Place	A	Human	-	
		2	Pick&Place	B	Allocatable	Task 1, 4	
		3	Pick&Place	C	Robot	Task 2	
		4	Pick&Place	D	Allocatable	-	
		⋮	⋮	⋮	⋮	⋮	

Table 3.2 Scenarios of job descriptions with different execution conditions for each type of dependency.

first case involves tasks with *In-Schedule dependencies*, meaning each agent carries out tasks independently. The second case involves tasks with a maximum dependency on one other task, while the third considers tasks directly dependent on multiple previous tasks. The second and third cases are called *Cross-Schedule dependencies*, as the sequence of tasks for at least one agent depends on another. The last column of the Table 3.2 contains dependency graphs that visualize the relationship between tasks.

The second scenario type consists of the same three cases of task dependencies as the first type but with the addition of some tasks that can be allocatable. This renders all cases as complex dependencies according to the iTax taxonomy.

The scheduling problem described in Section 4.2 is influenced by various types of dependencies, which must be considered when formulating the problem conditions. Dependencies also influence the robustness of a solution. They introduce more conditions that could break or delay the execution. Its goal is to allocate tasks in the most efficient way possible in order to minimize the overall execution time of the tasks and be robust under dependencies and considered uncertainties.

3.2.1 Ethical considerations

Automatic task allocation methods have gained popularity due to their potential to reduce human workload and improve efficiency. However, the ethical considerations of removing humans from the task allocation process have been discussed in human-robot teams [30]. This raises the question of whether it is appropriate to completely remove humans from the task allocation process and in what situations it might be acceptable. While automating the task allocation process can have benefits, there may be situations where a human's common sense and ability to assess a situation quickly could prevent negative outcomes that a well-designed task allocation method might not be able to anticipate. For example, a robot is assigned a task that manipulates delicate and fragile components. The automated task allocation method may not have the capability to consider subtle environmental changes or nuanced conditions that could impact the safe handling of these components. However, a human worker, with their common sense and ability to quickly assess the situation, might notice an unforeseen issue or potential danger and take appropriate action to prevent damage or accidents.

Roncone, Mangin, and Scassellati proposed a negotiation method in which a robot either offers to perform a task or asks a human to do it [40]. The person can then accept or decline the request. This ethical aspect and negotiation method are also applied in the distribution of tasks in this work. The presence of universal tasks makes it possible to consider a person's desire and allow him to accept or refuse the proposed tasks and allow the robot to perform the task allocated for the human instead of him; in other words, help him.

Allowing the option for human intervention in task allocation has various implications, particularly in relation to introduced uncertainty and scheduling considerations. One notable effect pertains to the definition of task availability time. In a typical scenario, let's consider a sequence of tasks where Task A is offered to the human worker, and Task B requires extensive preparation time. If the robot initiates the preparation for Task B upon offering Task A to the human, and the human subsequently refuses Task A, the prior preparation for Task B becomes redundant and inefficient.

This situation highlights the need for flexibility in task availability determination. The availability of a task should be assessed considering the possibility of human intervention and their decision-making process. Rather than assuming immediate availability upon offering a task, the scheduling system should allow for a buffer period or additional confirmation to ensure the human's acceptance before initiating time-consuming preparations for subsequent tasks. By incorporating such flexibility, the system can adapt to the uncertainties introduced by human decision-making, minimizing wasteful efforts and optimizing task execution efficiency.

Chapter 4

Scheduling with uncertainty and task allocation control

In the context of HRC, uncertainty is present in humans and machines, but the uncertainty introduced by humans is more difficult to predict and systematize than by robots. The uncontrollable nature of human collaborators must be considered when planning and executing jobs. To address this challenge, modern algorithms must be designed to handle human and robot behavior uncertainty. The first part of this chapter presents an approach to modeling uncertainties. Its uses in scheduling and for testing using the new probabilistic simulation environment. The second describes the method of scheduling with their incorporation. The third section explains the procedure for allocating tasks while considering the flexibility of task timing and the preferences of a human.

4.1 Uncertainty model

Uncertainty is an inherent aspect of work environments, affecting both humans and robots during task execution. To effectively address this uncertainty in HRC, a common approach is to utilize probability distributions instead of specific values, as highlighted in prior research on scheduling under uncertainty [8]. Probability distributions provide a flexible and dynamic representation of uncertainty, accommodating variations in human and robot performance. By capturing the range of potential values and associated probabilities, decision-making becomes more robust and adaptive in managing uncertainty within HRC.

This study focuses on modeling various uncertainties that significantly impact scheduling and task execution. These uncertainties involve human choice and task execution time, focusing on small deviations and delays resulting from robot failure (such as the inability to bring an object) or human factors (such as forgetting a tool).

4.1.1 Uncertainty in task durations

The offline determination of the minimum time needed to accomplish a task can be achieved through trajectory planning. However, in the context of HRC, online safety adjustments, such as reducing the robot's speed when in close proximity to a human, or the implementation of reactive skills like force-controlled peg insertion, can introduce delays in the execution of robot actions. However, the maximum execution time of a task can vary greatly depending on the complexity of the manipulations and the time spent by a person in the collaborative space. Therefore, for robots, any distribution of task completion time can be described based on these minimum and varying maximum values. On the other hand, determining task completion time for humans is more challenging, as it's harder to define precise boundaries. However, an aver-

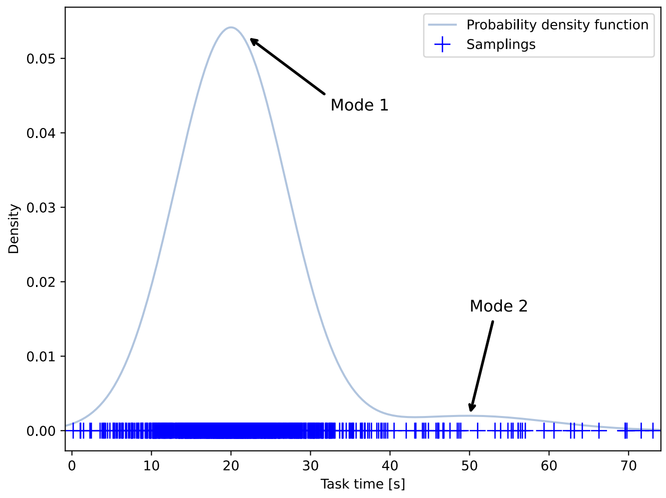


Figure 4.1 The figure shows a bimodal probability density function. Each mode is defined as a normal distribution. The first distribution is represented by the mode parameter $\mathcal{N}_1(\mu = 20, \sigma^2 = 7)$ and corresponds to the probability distribution of the task execution time without error or failure. The second distribution is represented by the mode parameter $\mathcal{N}_2(\mu = 50, \sigma^2 = 10)$ and corresponds to the probability distribution of the task execution time with error or failure, which has probability 5%.

age value can be used, and symmetric probability distribution can be applied to estimate task completion time.

From this information, it can be inferred that uncertainty in task execution time typically has a relatively low variance compared to the expected value. On the other hand, mistakes such as a robot failing to capture a part, losing a part, or a person getting distracted can have a much more significant impact on the overall time required to complete a task. However, these types of uncertainties are less likely to occur. While corner cases are not significantly important in scheduling, they are crucial in probabilistic simulations to evaluate the system's efficiency under extensive delays. These simulations allow for comprehensive system performance testing, considering a wide range of potential scenarios, including those with substantial delays.

The bimodal distribution is suitable for modeling inaccuracies in HRC [41]. Unlike the normal distribution, which has only one peak, the bimodal distribution consists of two distinct peaks representing different information groups. Figure 4.1 shows a bimodal probability density function consisting of two normal distributions. The first distribution represents the duration of a task with a possible delay, which is modeled using a normal probability distribution $\mathcal{N}_1(\mu = 20, \sigma^2 = 7)$. The second distribution represents the error rate, which is assumed to be 5%, and is modeled using another normal distribution $\mathcal{N}_2(\mu = 50, \sigma^2 = 10)$. It's important to note that the distribution may differ for each task and each agent, considering the unique characteristics and capabilities of the human and the robot involved in the collaboration.

4.1.2 Uncertainty in human decision-making

Human choice significantly influences scheduling and task execution in human-robot collaboration, introducing uncertainties that must be considered. When a task is offered to a human worker, their decision to accept or reject it is uncontrollable and observed during this process. To account for human choice, algorithms and models are developed to estimate the probability of task acceptance or rejection [15]. This consideration of human choice promotes ethical considerations (see Section 3.2.1) by reducing monotony in production and increasing job satisfaction. By incorporating human preferences, the system becomes more flexible and adaptive, enhancing overall efficiency and well-being in the human-robot collaborative environment.

In this work, human choice employs a categorical distribution, a discrete probability distribution that characterizes the potential outcomes of a random variable that can assume one of K distinct categories. In the context of this work, $K = 2$, represents the options of rejection or acceptance.

- **Probability of rejection and acceptance of each task.**
  This approach involves keeping track of how many times a person agrees to a task and collecting statistics for each task separately. This allows for analyzing the acceptance rate of different tasks, which can provide insights into individual preferences and tendencies.

  $$P_i(\text{rejection}) + P_i(\text{acceptance}) = 1, \forall i \in T/\{T_r\}, \tag{1}$$

  where $T/\{T_r\}$ is a set of all allocatable and non-allocatable human tasks.

- **General probability of rejection and acceptance.**
  It involves modeling a person's choice regardless of the specific task and conditions, aiming to capture the overall decision-making behavior of individuals in the HRC setting.

  $$P(\text{rejection}) + P(\text{acceptance}) = 1 \tag{2}$$

These different approaches to modeling uncertainty in human choice provide varying levels of complexity in understanding and predicting human decision-making in the context of HRC. Careful consideration of these uncertainties is essential in developing effective task distribution strategies and reactive control mechanisms in such collaborative settings.

This thesis focuses on testing scheduling and task allocation using both approaches, and the results of these tests are described in Chapter 6. The aim is to compare the effectiveness of these different approaches in managing uncertainty in human choice within the context of HRC.

4.2 Scheduling problem

The primary goal of scheduling is to optimize the sequencing of tasks to minimize the overall time required to complete the tasks according to a given job, a state of the world, and some (imprecise) knowledge of the uncertainties. It is necessary to plan ahead to avoid deadlocks and job constraint violations. Constraint Programming (CP) has gained popularity as a method for solving optimization problems, particularly in scheduling, where numerous constraints limit the search space. Recent studies in the Parallel Machine literature have demonstrated promising results for CP compared to heuristics and Mixed Integer Programming [18] [19] [20]. CP algorithms are specialized for solving Constraint Satisfaction Problem (CSP). A solution of CSP is assigning a value from its domain to each variable, satisfying all the constraints. CP algorithms utilize constraint propagation techniques and search strategies to find solutions and can optimize an objective function utilizing repeated satisfaction. Exhaustively searching all feasible solutions will find the optimal solution to the problem.

In CP, the problem is represented by parameters, a set of variables, each with a defined domain of possible values, and a set of constraints that restrict the valid combinations of values for the variables. Compared with Linear Programming (LP), conditions and an objective function can be formulated by non-linear expressions as logic, especially $\oplus$ - *XOR* [42].

1. **Parameters**
   - A - set of agents
   - T - set of tasks
   - $T_a \subset T$ - set of non-allocatable tasks for each agent $a \in A$; $\bigcap_{a \in A} T_a = \emptyset$
   - $R_i^a \in \mathbb{Z}_0^+$ - processing time of task i, when it is performed by agent a; $a \in A$
   - $Y_{i,j} \in \{0,1\}$ -$Y_{i,j}$ is 1 when task j depends on task i, otherwise 0; $i \neq j$, $i, j \in T$
   - $K_{a,i,j} \in \mathbb{Z}_0^+$ - overlapping of task j to task i, when the latter is performed by agent a; $i \neq j$, $i, j \in T$, $a \in A$
   - $l = \sum_{a \in A} \sum_{i \in T_a} R_i^a + \sum_{i \in T / \{\bigcap_{a \in A} T_a\}} \max_a (R_i^a)$ - horizon
2. **Variables and domains**
   - $s_i \in \{0, 1, ..., l\}, \forall i \in T$ - start time of task i
   - $e_i \in \{0, 1, ..., l\}, \forall i \in T$ - end time of task i
   - $x_i^a \in \{0,1\}$. $x_i^a = 1$ if task i is performed by the agent a, otherwise $x_i^a = 0$, $\forall i \in T$, $\forall a \in A$
3. **Objective**
   - minimize makespan: minimize $(\max_{i \in T} e_i)$
4. **Constraints**

$$\sum_{a \in A} x_i^a = 1, \forall i \in T, \tag{3a}$$

$$(x_i^a = 1), \forall i \in T_a, \forall a \in A \tag{3b}$$

$$(x_i^a = 1) \Rightarrow (e_i - s_i = R_i^a), \forall i \in T, \forall a \in A \tag{3c}$$

$$((Y_{i,j} = 0) \wedge ((x_i^a \wedge x_j^a) = 1) \Rightarrow ((s_j \geq e_i) \oplus (s_i \geq e_j)), i \neq j, \forall i, j \in T, \tag{3d}$$

$$((Y_{i,j} = 1) \wedge ((x_i^a \wedge x_j^a) = 1) \Rightarrow (s_j \geq e_i), i \neq j, \forall i, j \in T, \tag{3e}$$

$$((Y_{i,j} = 1) \wedge ((x_i^a \wedge x_j^a) = 0) \Rightarrow (s_j \geq s_i + K_{a,i,j}), i \neq j, \forall i, j \in T, \forall a \in A, \tag{3f}$$

To formulate an optimization problem, it is important first to identify the available data and assign appropriate indexes and sets. The list of agents A can be indexed by the label a. The task list T can be indexed by i and j. Each task has a description that includes constraints on its execution. Certain tasks are non-allocatable, and as a result, they are grouped into subsets $T_a, \forall a \in A$ of set T, and the intersection of all sets is empty because a specific agent can only carry out non-allocatable tasks.

The next step is to define the problem's parameters, shown in item 1. As either agent can complete some tasks, it is not possible to assign a constant execution time for such tasks. This is because the execution time depends on the agent assigned to the task. Therefore, set R_i^a is defined, which contains information about the processing time of task i by agent a. For non-allocated tasks, the processing time of the agent that cannot complete the task is set to 0. This value will not be chosen anyway due to hard constraints. The duration uncertainty is factored into the schedule in the variable R_i^a. It is defined as the sum of the durations of the task phases described in Section 3.1. The duration of each phase is taken from the probability distribution, which displays information about potential delays.

The dependency constraints are represented by a Boolean parameter $Y_{i,j}$ with indexes i and j, indicating that task j depends on task i. Calculating the overlap constant $K_{a,i,j}$ relies on the phase duration of both tasks. The overlap is determined by adding up the preparation and execution times of the main task and subtracting the preparation time of the dependent task.

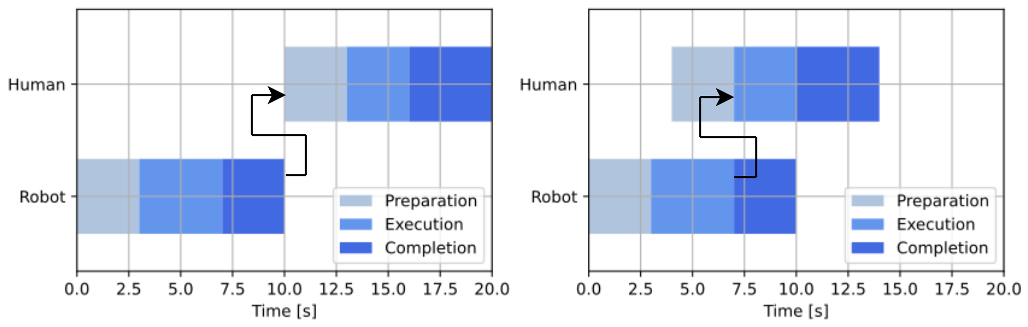


Figure 4.2 The figure illustrates overlapping in a scheduling process where a human task depends on a robot task. The image on the left shows a schedule where no overlapping is considered, and the makespan is 20 seconds. However, overlapping is considered in the image on the right, and preparation for the human task starts before the robot task is finished, resulting in a reduction makespan of 7 seconds.

It's important to note that the duration of task execution and its phases vary across agents. Consequently, the overlap is calculated beforehand and is included in the model as a constant with three indexes: a - the agent responsible for the main task, i - the main task, and j - the dependent task. Figure 4.2 presents a visualization of a task sequence with and without overlapping. Specifically, the figure shows that if the preparation and execution of the first task take 7 seconds in total, and the preparation of the second task takes 3 seconds, the second task can be started at 4 seconds after the beginning of the first task due to the overlap.

This problem aims to find the optimal way to distribute allocatable tasks and determine their order of execution along with non-allocatable ones to achieve the minimum time to complete all work. The start and end times of each task are determined by discrete time. These values contain the solution variables s_i and e_i for the start and end of task i shown in item 2. The start and end variable domains are bounded from zero to the variable l, representing the job's maximum horizon.

Each task can be assigned to only one agent and performed only once. A variable x_i^a determines the task i assigned to agent a. The variable takes a value of 1 if the task is assigned to agent a and 0 if it is not assigned to agent a.

The primary objective is to minimize the makespan, as previously mentioned. The objective function in item 3 aims to minimize the maximum e_i within the T set. Several constraints need to be met to find the correct optimal solution. Equation 3a ensures that only one agent completes the task. Equation 3b specify the value of variable x_i^a based on the input parameters. This notation represents hard constraints in CP which are non-negotiable conditions that must be strictly satisfied for a valid solution, and violating them renders the solution infeasible. Equation 3c establishes an interval condition that ensures the difference between the end and start of the job is equivalent to the task duration based on the assigned agent.

Three additional constraints ensure that tasks are performed in the correct sequence and consist of literals. Literals serve as building blocks for expressing constraints and guide the search for valid solutions in CSP. Each constraint uses the "if" construct to adjust the conditions as necessary. The constraint described in the Equation 3c determines the order in which one agent performs independent tasks. The first literal on the left-hand side of the expression indicates no dependency between tasks. The second literal returns true when both tasks are assigned to the same agent. The literal on the right-hand side with XOR ensures that one task cannot begin until the previous task is completed. Otherwise, two tasks may be assigned to the same agent simultaneously. The last two constraints ensure the correct sequence of dependent tasks, as indicated by the literal with parameter $Y_{i,j}$ on the left sides of both expressions. The second literal defines whether the tasks are assigned to the same agent. It implies that the start of the dependent task can only occur after completing it from the previous condition. If the agents are working on different tasks, the agent performing the dependent task can start preparing for it before the end of the previous task. The specific value of the start of preparation is defined by the overlap parameter $K_{a,i,j}$.

The solution to the problem is presented by specifying which tasks are performed by which agent and in what time interval. An example of a completed schedule is shown in Figure 4.3, which includes a Gantt chart and a graph of task dependencies. The Gantt chart displays the types of tasks, whether they are allocatable or not (shades of green and blue), and to whom they are allocated: a human or a robot. Shades of color correspond to the different phases of the task: preparation, execution, and completion. The dependency graph indicates that task C can only be performed after task B. However, there is allowed overlap, which helps minimize the makespan.

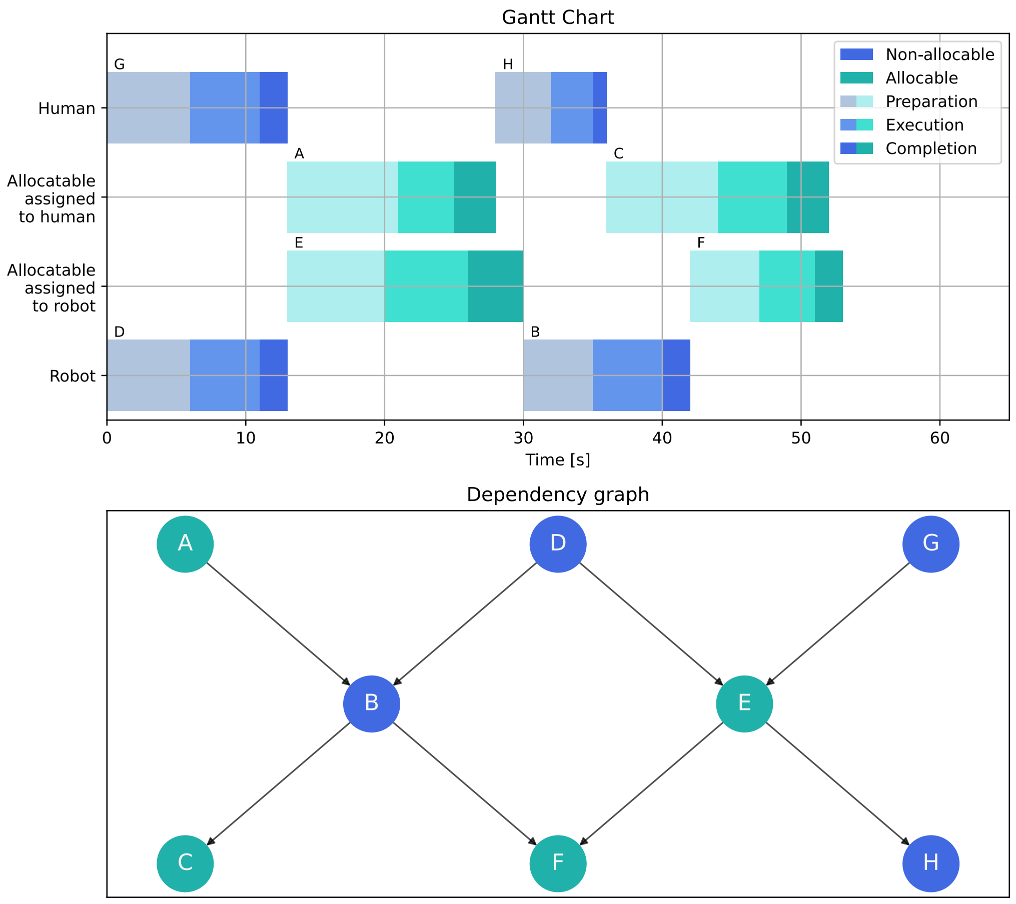


Figure 4.3 This figure illustrates the output of the schedule in two parts. The top part is a Gantt Chart that depicts the start times of the tasks, with non-allocated tasks in the first and last rows and allocated tasks in the middle rows assigned to agents based on the schedule. The colors of the tasks are divided into blue and green to represent allocatable and non-allocatable, respectively. The gradient of these colors corresponds to the task phase described in the legend. The bottom part of the figure shows a dependency graph, with nodes representing tasks and edges depicting the dependencies between them.

4.2.1 Soft constraints

Using constraint programming for scheduling aims to achieve optimal task allocation, but allowing human choice can disrupt the sequence and reduce scheduling optimality. Additionally, it can impact the reliability of the predicted completion time. It's possible to consider human choice during scheduling to avoid task cancellation and rescheduling.

Soft constraints can be added if the data about preferences is available to incorporate a person's preferences into the scheduling process. The first two models of human choice described earlier in the chapter can be used for this purpose. For instance, if the assignment-based model is used, the soft constraints are represented as a sum of variables that depend on the person's choice and are added to the objective function. Thus, it takes the form

$$\text{minimize}(\max_{i \in T} e_i + \lambda \sum_{i \in T/\{T_a\}, \forall a \in A} P(i)(e_i - s_i)), \tag{4}$$

where $P(i), \forall i \in T/\{T_a\}, \forall a \in A$ is the probability of rejection by human allocatable task i multiplied by task processing time. $\lambda \in \mathbb{Z}_0^+$ scales the importance of the rejection risk penalty compared to the makespan. This constant sets the cost increase and must be selected based on the mean task duration. If the value is too high, the soft constraints become more important than the makespan. Conversely, the solver disregards the cost constraints if the value is too low. The soft constraint can be interpreted as the cost of choice. The cost of selecting a task for a person with a lower probability of rejecting it is lower than for a person with a higher probability.

To put the theoretical problem into practice, the Google OR-Tools were utilized. A more comprehensive explanation of the implementation of CP model, its solutions, and modifications can be found in Chapter 5.

4.3 Control method

Many studies on making technology, specifically robots, act more as humans focus on cognitive decision-making principles [43] [44]. However, what if we approached human-robot interaction as human-human interaction? For example, imagine a scenario where you and a friend cook dinner together. You divide the tasks, with your friend completing one portion of the recipe and you completing the other. If you finish your task first, you can wait for your friend to finish or offer to help. The goal is to make dinner together as you both are hungry. By offering to help, you make the task easier for your friend and may achieve the common goal faster. This concept of mutual aid can be applied to working with robots to simulate human interaction.

The allocation approach proposed in paper [7] has a similar idea but lacks the inclusion of scheduling. To begin with, the non-allocatable tasks are explored in the task list for each agent individually. The search proceeds to the allocatable task list if no such task is found. When exploring non-allocatable tasks, the available tasks are sorted by their duration, from shortest to longest. On the other hand, when exploring allocatable tasks, the tasks are sorted according to the highest benefit in terms of their execution time relative to the other tasks. This task allocation method mimics human behavior, prioritizing agents completing their assigned tasks before considering shared tasks to expedite completion. However, if humans know the sequence of tasks for themselves and the robot beforehand, it can provide a more comprehensive understanding of the process and aid in holistic production planning.

This paper builds upon the idea of task allocation through mutual aid and adds scheduling and individual choice to the process. Figure 4.4 illustrates the decision-making process for task allocation, which involves defining states for agents and tasks during the transition between solving links. Four main states are used to track task availability, with transitions occurring at the beginning or end of a task.

- *-1* - the task cannot be started; it's unavailable. The conditions of this task have not yet been started.
- *0* - the task can be completed; it's available. The task has no conditions, or the conditions have already been met.
- *1* - the task is in progress. The control logic has received a positive response from the person/robot requesting the task.
- *2* - the task is completed. The position of the part corresponds to the task.

The tasks start with a state of *-1*, indicating that they have not available yet. When the system updates the states at the beginning of a cycle, it checks whether there are any dependent tasks for a given task. If there are no dependent tasks or they have a state of *1* or *2* (which indicates that they are in progress or have been completed), then the task is assigned a state of *0*, indicating that it is available for allocation to an agent. When an agent accepts a task, its state is set to *1*, indicating that it has been assigned to an agent. Finally, after the agent completes the task and provides feedback, the task's state is set to *2*, indicating that it has been completed. Similarly, agents have states such as:

- *Available* - the agent has finished the previous assignment and is waiting for the next one.
- *Unavailable* - The agent has accepted the assignment and is completing it.

When the system starts, all agents are initialized with a state of *Available*, indicating they are free to perform tasks. When an agent accepts a job, their state changes to *Unavailable*, indicating that they are currently occupied with a task. During the task execution, the system monitors the feedback from the agents to check whether the job is completed. If the task is completed, the agent's state is updated back to *Available*, indicating they can perform other tasks. This cycle continues as new tasks are assigned and completed, with the state of agents being updated accordingly.

A logic diagram in Figure 4.4 describes a decision-making process for allocating tasks based on schedule, agents' availability, and the status of the tasks. The process begins by updating the task status and agent availability. The system checks the availability of the robot agent, and if it is available, it checks for any available tasks assigned to it with a status of 0. If an available task is found, the robot is commanded to execute it. If not, the system checks for any allocation tasks with a status of 0 and asks the human agent if the robot can do the task instead. If the human agent agrees, the robot starts the task; otherwise, searching for an available task continues.

If the robot agent is unavailable, the availability of the human agent is checked next. If the human agent is available, the system checks the list of tasks assigned to the human agent and determines if any are available with a status of 0. If an available task is found and allocated only to the human, then the task is sent to the human agent for execution. If the task is allocatable, the human agent can choose whether to do it. If the human agent declines the task, the task is assigned to the robot agent, and the scheduling function is called to determine the appropriate time for completion.

If the human agent has no available tasks with a status of 0, the system checks the list of tasks assigned to the robot agent to see if the human agent can help with any of them. If so, the

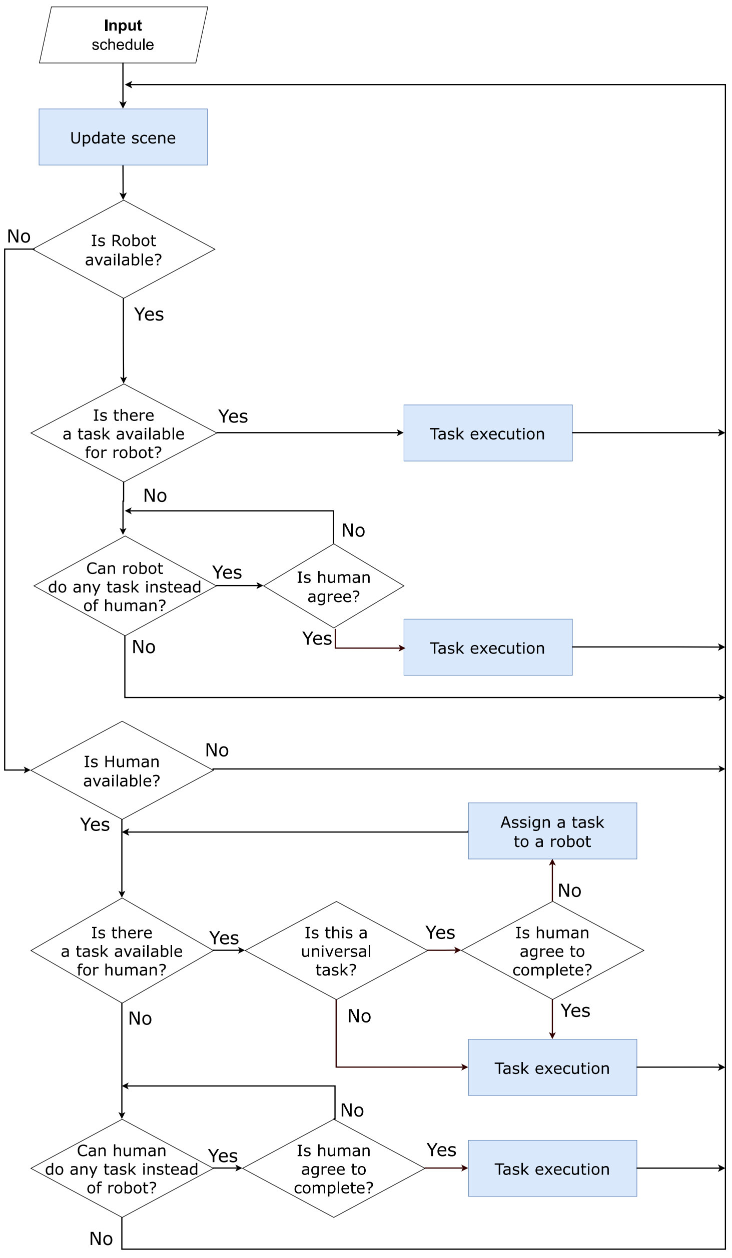

Input schedule
Update scene
Is Robot available?
No
Yes
Is there a task available for robot?
Yes
Task execution
No
No
Can robot do any task instead of human?
Yes
Is human agree?
Yes
Task execution
No
Is Human available?
No
Yes
Assign a task to a robot
No
Is there a task available for human?
Yes
Is this a universal task?
Yes
Is human agree to complete?
Yes
No
No
Task execution
No
Can human do any task instead of robot?
Yes
Is human agree to complete?
Yes
Task execution
No

Figure 4.4 Decision-making diagram

system asks the human agent if they agree, and if not, the search for an available task continues until there are no more available tasks.

In addition to considering the uncertainty of task times when scheduling, the control logic is proposed in such a way as to avoid agent downtime. The ’Can robot/human do any task instead of human/robot?’ component has an algorithm to manage changes in the schedule. If an agent does not have a task, available and allocatable tasks of another agent are considered. However, the order in which tasks are offered to the second agent is important. It evaluates all possible rescheduling options and sorts them by ascending makespan. The control logic compares this value with the current makespan and suggests a task exchange if it is lower.

Chapter 5

Implementation

The implementation was done in multiple stages with individual component testing. The first part of the chapter discusses the implementation of the model and the reformulation of the mathematical problem to a solver-friendly form. The next part describes the implementation of the decision diagram and other necessary components for task allocation among agents. A separate subsection is dedicated to the numerical simulation process, which tests the scheduling and task allocation for robustness and stability. The final section focuses on integrating the above structure into a real setup with a robot, cameras, and gloves, emphasizing motion control based on the behavior tree.

5.1 Software architecture

The program is divided into several classes that interact with each other to achieve task allocation. Here is a breakdown of each class instance created during runtime:

- When a job creation request comes in, the *Job* class is activated, which takes in a field of dictionaries as input. Each dictionary specifies the characteristics of each task, such as ID, object, place, constraints, and allocability.
- The *Job* class creates a list of all tasks, and each task is defined as an object of the *Task* class. This class stores all information about the task, including its ID, allocability, allocated agent (if any), condition, and expected duration. During task execution, the class records the state, start time, and end time information.
- After creating the task list, the *Schedule* class is instantiated with the *Job* class object as an input parameter. This class creates a CP model that is solved initially and updated as needed to revise the schedule or evaluate possible task redistribution. During initialization, the class searches for the optimal schedule, which is recorded as a dictionary with the names of agents and a list of assigned tasks. The mode of the implementation of the model is in the next section.
- The *Agent* class is created next from an abstract class, and each agent is assigned a list of allocated tasks, and the availability status is initialized. In the case of simulation, this class inherits from the *Simulation* class, which stores data about the parameters of probabilistic distribution and the randomization seed. The class's functions numerically simulate the execution of tasks, and the simulation algorithm is described in more detail in the following section.
- Finally, all the previous objects are initialized from the *ControlLogic* class, which is the primary class that controls all the others. After obtaining the schedule and initializing the

Type	Symbol	Domain	Literal
Variables	d_i	$\{R_i^a\}, \forall a \in A$	$(e_i - s_i) == d_i$
	$g_{i,j}$	$\{0, 1\}$	$(x_i^a \wedge x_j^a) == g_{i,j}$
	$b_{i,j}$	$\{0, 1\}$	$(s_j \geq e_i)$ if $b_{i,j} == 1$
			$(s_i \geq e_j)$ if $b_{i,j} == 0$

Table 5.1 Additional variables for implementing decision literals. The first variable represents the task duration and its domain comprises the possible durations defined by all agents. The second variable has a value of 1 when both jobs i and j are assigned to the same agent. The third variable dictates the task order, ensuring that no two tasks are assigned to the same agent simultaneously.

agents, the algorithm described in the previous section is executed, Figure 4.4. Within each cycle, the task status information is updated first, provided by an object of the *Agent* class. Then, the availability of agents is checked in turn, and if available, an accessible task is sought. The search continues until the task starts or the list ends. If there is no available task for an agent, or the agent refuses the task, the algorithm to find an available task for another agent is invoked to help it. The implementation of the rescheduling evaluation function is shown in Algorithm 3 and described in the Section 5.2.2. Unlike finding one's task, this algorithm always asks for the human's consent to changes in the schedule.

Once the search for available tasks for each agent has been completed, the cycle starts again. For simulation, the time required for each task is calculated in cycles. This involves setting a global time variable at the start, incremented by one at the end of each cycle. The time variable is a real timer if the task allocation algorithm is used in a real system. Once the simulation concludes, it becomes feasible to represent the initial and final schedules visually. Additionally, the simulation allows for the playback of an animation illustrating the schedule modifications.

This work limits its consideration to job types with a maximum of 20 tasks. It may be beneficial to divide more complex jobs into smaller parts and limit planning to a reasonable amount. Long-term planning in uncertain conditions may not be optimal and could result in slower production. Therefore, planning up to a visible horizon of 20 tasks is recommended.

5.2 Modeling and solving constraint programming

There exist several freely available solvers for combinatorial optimization problems, and one such solver is Google OR-Tool [1]. Google OR-Tools stands out for its efficiency, scalability, versatility, ease of integration, customization options, and active development and support compared to other CP-SAT-solving tools or libraries. Other tools may also be suitable depending on specific requirements or preferences. Still, Google OR-Tools is widely recognized and used in constraint programming for its capabilities and performance.

Implementing a CP is similar to creating the mathematical description given in Section 4.2. The main difference is that only two agents, human and robot ($A = \{h, r\}$), are considered in the implementation. The initial phase is to create the required variables. There are three fundamental functions, namely *NewBoolVar()*, *NewIntVar()*, and *NewIntVarFromDomain()*, to create variables in the model. The third function creates the additional variable d_i, representing the task duration. The domain of this variable consists of two values, the time for a task to be completed by a human or a robot. It is also necessary to indicate the dependencies between the variables s_i, d_i, and e_i so that the difference between the end and the task start is equal

[1] https://developers.google.com/optimization

to the duration. This is implemented with another fundamental function *NewIntervalVar()* that creates constraints within the model.

The *CP_model* library requires the creation of additional variables to store the results of certain conditions, presented in Equation 5:

$$((Y_{i,j} = 0) \wedge ((x_i^a \wedge x_j^a) = 1) \Rightarrow ((s_j \geq e_i) \oplus (s_i \geq e_j)), i \neq j, \forall i, j \in T, \tag{5a}$$

$$((Y_{i,j} = 1) \wedge ((x_i^a \wedge x_j^a) = 1) \Rightarrow (s_j \geq e_i), i \neq j, \forall i, j \in T, \tag{5b}$$

$$((Y_{i,j} = 1) \wedge ((x_i^a \wedge x_j^a) = 0) \Rightarrow (s_j \geq s_i + K_{a,i,j}), i \neq j, \forall i, j \in T, \forall a \in A, \tag{5c}$$

Table 5.1 describes the additional variable, their respective domains, and literals. For instance, the result of the literal $(x_i^a \wedge x_j^a)$ responsible for the same agent for tasks i and j is stored in variable $g_{i,j}$. Similarly, the results of literals $(s_i \geq e_j)$ and $(s_j \geq e_i)$ are stored in variable $b_{i,j}$, which automatically performs an XOR function to ensure that both literals do not have the same value.

Algorithm 1 Variables

```
 1: Input: Tasks                                            // list of tasks
 2: Input: $R^h$, $R^r$                                     // lists of tasks duration
 3: Input: $T_h$, $T_r$                                     // lists of non-allocatable tasks
 4: Initialize: CpModel()
 5: l = set_horizon()
 6: for i in Tasks do
 7:     Initialize Boolean variables: $x_i$
 8:     Initialize integer variables from the domain $\{0, 1, ..., l\}$: $s_i$, $e_i$
 9:     Initialize integer variables from the domain $\{R_i^h, R_i^r\}$: $d_i$
10:     Add interval constraints: $(s_i, d_i\ e_i)$
11:     if i in $T_h$ then
12:         $x_i \Leftarrow$ True
13:     else if i in $T_r$ then
14:         $x_i \Leftarrow$ False
15:     end if
16:     $(x_i ==$ True$) \implies (d_i == R_i^h)$
17:     $(x_i ==$ False$) \implies (d_i == R_i^r)$
18: end for
```

All the required variables and literals have been obtained and can now be integrated into the model. The goal of Algorithm 1 is to define various aspects of a task, such as its start time, end time, duration, and whether it can be allocated to an agent. Algorithm 1 takes as input a list of tasks, lists of task durations for humans and robots, and lists of non-allocatable tasks for both agents (rows 1-3). It initializes a CP model and sets the maximum horizon l of all the tasks in the job for all the agents based on the equation:

$$l = \sum_{a \in A} \sum_{i \in T_a} R_i^a + \sum_{i \in T / \{\bigcap_{a \in A} T_a\}} \max_a (R_i^a) \tag{6}$$

For each task, it initializes Boolean variables and integer variables for start and end times from the domain $\{0, 1, ..., l\}$ and the duration variable from the domain consisting of the corresponding task duration list (rows 7-9). It then adds interval constraints for the task start, duration, and end time (row 10). Then sets hard constraints for non-allocatable tasks. If the task is non-allocatable and can be performed only by a human, it sets the Boolean variable to true. If it is non-allocatable and can be performed only by a robot, it sets the Boolean variable to false (rows 11-15). If the task is allocated, the solver determines this value. The duration is set to the corresponding duration value based on the Boolean variable value (rows 16,17).

The second part of the code described in Algorithm 2 creates relationships between pairs of tasks using equations Equation 5a, 5b and 5c. Inputs are a list of tasks, dependencies between tasks, and recalculated overlapping. It then loops through all pairs of tasks and initializes Boolean variables (same agent, task i before task j, dependencies) and integer variables (permitted overlapping) for each task (rows 7,8). The code creates variables for each pair of tasks to indicate that they have the same agent (rows 9,10). It sets hard constraints to Boolean variable $y_{i,j}$ which represents the value of the dependency parameter$Y_{i,j}$ (rows 11-15). Then the overlapping values for each agent are included in the model (rows 16,17). All the necessary variables are in the model, and the next step is to implement the constraints following the equations.

Algorithm 2 Constraints

```
1: **Input:** Tasks                                                // list of tasks
2: **Input:** Y                                      // 2D array of tasks conditions
3: **Input:** K                                 // 3D array of overlapping parameters
4: **for** i **in** Tasks **do**
5:     **for** j **in** Tasks **do**
6:         **if** i != j **then**
7:             Initialize Boolean variables: $g_{i,j}$, $b_{i,j}$, $y_{i,j}$
8:             Initialize integer variables: $k_i$
9:             $(g_{i,j} == True) \implies (x_i == x_j)$
10:            $(g_{i,j} == False) \implies (x_i != x_j)$
11:            **if** $Y_{i,j} ==$ True **then**                  // task $j$ depends on task $i$
12:                $y_{i,j} \Leftarrow$ True
13:            **else**
14:                $y_{i,j} \Leftarrow$ False
15:            **end if**
16:            $(y_{i,j} ==$ True **and** $x_i ==$ True$) \implies (k_i == K_{h,i,j})$
17:            $(y_{i,j} ==$ True **and** $x_i ==$ False$) \implies (k_i == K_{r,i,j})$
18:            $(y_{i,j} ==$ False **and** $g_{i,j} ==$ True **and** $b_{i,j} ==$ True$) \implies (s_j \geq e_i)$   // 5a
19:            $(y_{i,j} ==$ False **and** $g_{i,j} ==$ True **and** $b_{i,j} ==$ False$) \implies (s_i \geq e_j)$  // 5a
20:            $(y_{i,j} ==$ True **and** $g_{i,j} ==$ True$) \implies (s_j \geq e_i)$           // 5b
21:            $(y_{i,j} ==$ True **and** $g_{i,j} ==$ False$) \implies (s_j \geq s_i + k_i)$     // 5c
22:        **end if**
23:    **end for**
24: **end for**
```

To set up an objective function for an optimization problem, a new integer variable is first created, *obj_var*, with a lower bound of 0 and an upper bound of *horizon*. It then adds a maximum equality constraint to *obj_var*, which equals the maximum end time of all tasks using *AddMaxEquality()* function. Finally, it specifies that the objective is to minimize *obj_var*, which means that the solver will try to find a solution that minimizes the makespan.

```
obj_var = model.NewIntVar(0, horizon, 'makespan')
model.AddMaxEquality(obj_var, [task.end for task in job])
model.Minimize(obj_var)
```

5.2.1 Updating model variables in constraint programming

In order to incorporate the observed progress, additional constraints need to be imposed on the model, which is not directly supported by OR-Tools. Nevertheless, an intermediate representation can be obtained and modified directly, eliminating the need for code generation. The domain of the variable is first cleared by setting it to an empty list with

```
model.Proto().variables[start_var[i].Index()].domain[:] = []
```

Then, the new domain is created by taking the lower and upper bounds of the variable and extending it to include all the intervals in the domain using the

```
new_domain = cp_model.Domain(current_time, horizon)
                                        .FlattenedIntervals()
```

In this particular case, the start time of the task is restricted to the current, as the task cannot begin in the past. Consequently, the variable's domain needs to be limited. Finally, the domain of the variable is updated with the new domain

```
model.Proto().variables[start_var[i].Index()].domain
                                        .extend(new_domain)
```

5.2.2 Rescheduling estimation

Various reasons can lead to the unavailability of tasks for an agent, such as the delay of another agent on whom others depend or the rejection of tasks offered to a human. In this case, there is a possibility that one of the agents may not have any available tasks at a given moment. However, if there are allocatable tasks, it is possible to assign and complete the task to the remaining agent instead. This raises the question of whether it would be advantageous to minimize the execution time of the work, and if so, which task redirection would yield the most optimal outcome. To address this, a search is initiated to find tasks the first agent can perform instead of the other. Assigning tasks based on priority or subjective evaluation of execution time is not recommended. Instead, it is suggested to evaluate the makespan after a potential rescheduling.

Algorithm 3 present a solution for evaluating possible changes. It takes a list of coworker available allocatable tasks as input, an agent, and an existing CP model. The code then initializes an empty list for task priorities and updates the CP model based on the current progress. It then iterates through the tasks and creates a copy of the existing CP model. Suppose the agent is 'Human', a Boolean variable is set to True. Otherwise, it is set to False. The model is solved to calculate the makespan for each task. The task and its corresponding makespan are added to the task priority list. Finally, the task priority list is sorted in ascending order of makespan and returned.

Another option is to search for a new optimal solution in the updated model without fixing the variable. Adopting this alternative approach would involve redistributing the remaining tasks, which could result in a significant change to the schedule. However, displaying such a modified schedule may confuse or mislead individuals, as it contradicts the intuitive and easily understandable nature of the planned actions.

5.2.3 Probabilistic simulation

A probabilistic simulation was developed to assess the effectiveness of the scheduling method in handling uncertainties. The simulation served as a validation tool to ensure the correct allocation and sequencing of assignments. It also verified the reliability and accuracy of the rescheduling process. The simulation was designed to replicate agents' signals regarding task

Algorithm 3 Rescheduling evaluation

```
1: Input: Tasks                          // list of coworker allocatable tasks with status 0
2: Input: agent
3: Input: model                                                     // existing CP model
4: function SET_LIST_OF_POSSIBLE_CHANGES(Tasks, agent, model)
5:     Initialize: $task\_priority = []$
6:     Initialize: update model
7:     for i in Tasks do
8:         Initialize: $model\_copy$
9:         if agent is 'Human' then
10:            $x_i = \texttt{True}$
11:        else
12:            $x_i = \texttt{False}$
13:        end if
14:        $makespan = \text{Solve}(model\_copy)$
15:        $task\_priority$ add $[tasks[i], makespan]$
16:    end for
17:    $possible\_changes \Leftarrow task\_priority$ in ascending $makespan$
18:    return $possible\_changes$
19: end function
```

progress, such as completion or rejection. These events are not controllable by the control logic and are only observable. Using this simulation, the performance and correctness of the scheduling method could be evaluated without relying on real-world execution.

To simulate uncertainty, the simulation controls the outcome of uncontrollable variables, such as the duration of a task. Experiments are generated by sampling the necessary values from appropriate distributions, such as the bimodal distribution introduced in a Section 4.1.1. Distributions are modeled based on oracle estimates, where the assumed values are treated as the true values to estimate system behavior. The first mode represents a regular processing time of phase without delays due to errors. The second distribution mode, which represents the error in the task's phase duration, had a mean of three times the normal execution time and a variance of three times. Consequently, when sampling the time of a task, a bimodal probability distribution is generated for each phase of the task. The overall duration of the task is determined by the sum of the durations of these three phases.

For task execution, the control logic sends an assignment command to the Agent object for each agent. The control logic is only informed about the tasks assigned and receives feedback about task progress as it shown in the sequence diagram in Figure 5.1. Once the task executed by a robot is completed, the timer reaches the end of the task in the simulation context, and the Agent class sends a response to the control logic. Thus the control logic only has data about the task progress and does not know exactly when it will end. On the other hand, human feedback is more complex as it is difficult to estimate the current movement of a human. Therefore, the human Agent class only sends a message to the control logic indicating that they have accepted and completed the task without feedback about progress. In other words, the simulation considers the unobservability of human task phases.

Determining the duration of a task is not always based solely on probability distribution sampling. An example in Figure 5.2 shows a scenario where a human task's availability depends on a task performed by a robot. In such cases, the control logic can initiate task preparation, but a human will have to wait until the completion of the previous task's execution phase. The simulation considers this waiting time to ensure that tasks do not end before their due date.

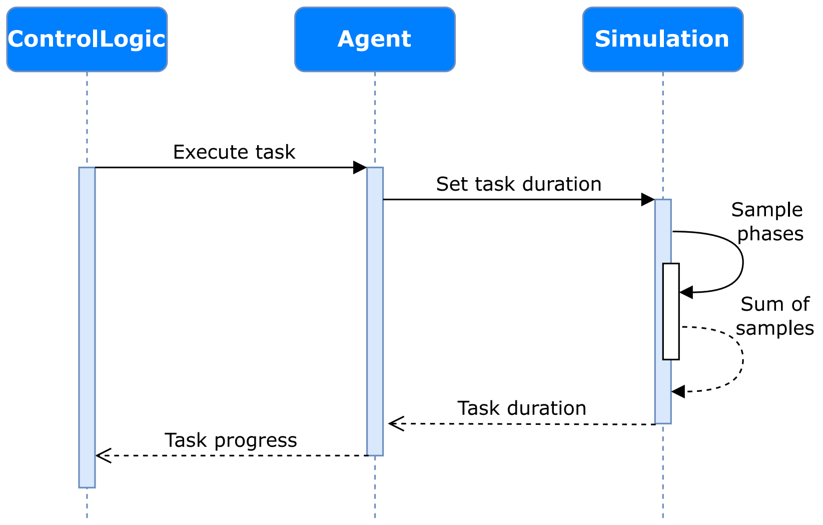


Figure 5.1 The sequence diagram illustrates the communication flow between different classes involved in simulating a task execution. Control logic initiates the task by sending a command to the agent, specifically the robot in this case. The Agent class, which inherits the Simulation class methods, utilizes a method to sample the duration of task phases. Based on the sampled duration and the current time, the agent provides feedback on the progress of the task to the controlling logic. In this way, the robot's task process is uncontrollable but observable. In contrast, when a task assignet to human, only information indicating that the task is in progress is sent to the controlling logic during the task's duration, simulating the inherent uncontrollability and unobservability of human task progress.

This waiting time is calculated as follows:

$$preparation_i + execution_i - current\_time + s_i, \tag{7}$$

where i is a task on which the execution of the second task depends.

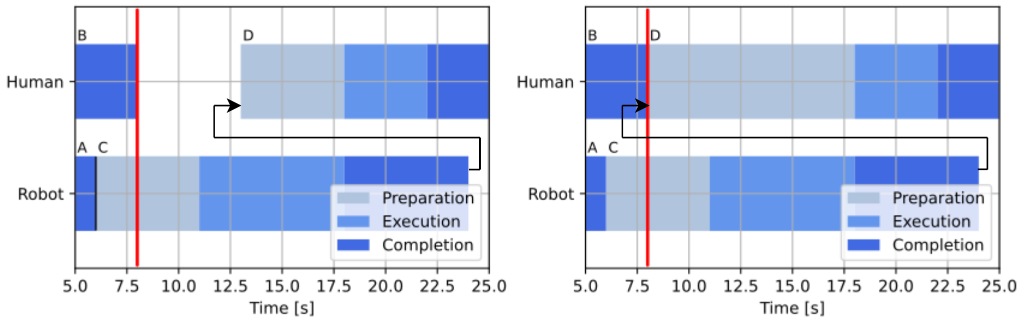


Figure 5.2 Human task D availability depends on a previous task C by the robot, leading to waiting times considered in the simulation to avoid early task completion.

The second part of the simulation deals with human choice. Considering the previous discussion in Section 3.2.1 on allowing individuals to reject assigned tasks and proposing task reallocation, it is essential to incorporate these aspects into the simulation to assess the system's response to schedule changes. Three main types of questions were taken into account:

1. Execution of the allocatable task.
2. An offer to perform an allocatable task instead of a robot,
3. Proposal to allow a robot to perform an allocatable task instead of a human.

In all ways, a method of predicting the person's choice is used, as discussed in Section 4.1.2. The first method calculates the probability of accepting or rejecting an offer independent of the task. The second method calculates the probability of accepting or rejecting each task separately. When considering a third question to allow a robot to perform a task instead of a human, the probability of task rejection is reversed. This means that the likelihood of rejecting the task becomes the likelihood of accepting the proposal.

5.3 Real robot setup

One of the aims of this study is to implement the scheduling and allocation of tasks using the proposed method in a real robot setup. The integration process involved multiple components, including Franka Emika Panda robot control, camera-based object tracking in the workspace, and tracking of human hands with Hi5 VR Gloves. To facilitate communication and instruction with the individual, a Graphical User Interface (GUI) was developed. Lastly, the control logic was integrated and responsible for task execution commands based on the schedule and monitoring it.

5.3.1 Real robot system architecture

Robot Operation System (ROS) 2 was used to integrate a system; its architecture is shown in Figure 5.3. A control logic module is created in the main node, which receives information about the environment and allocates tasks. The control node wraps the OOP objects and communicates with the robot and human nodes using objects of the Agent class. Task assignment for a human is achieved through a ROS action. For the robot, the task and feedback are implemented with two ROS topics, which functionally correspond to the structure of the action. This is a forced substitution due to the limited functionality of the library for the behavior tree described in the next subsection. Queries to humans for task acceptance or redistribution are accomplished through the ROS service pattern. Other relevant information is published using ROS topics.

The camera node processes data from the camera and performs object detection using ArUco[2] markers recognized in the OpenCV library. The camera node then publishes a list of the detected objects and their coordinates.

5.3.2 Reactive control node

The robot's movement is controlled by a behavior tree, which is a reactive framework that guides the robot's actions based on its current state and the desired outcome [25]. The tree controls task-related operations and monitors safety-relevant conditions, such as possible collisions between the robot and the human.

The BT has been implemented using the *py_trees*[3] and *py_trees_ros*[4] libraries. These libraries provide tools for building and executing behavior trees, allowing complex behavior to be decomposed into smaller, more manageable tasks. With this approach, the robot's behavior can be easily modified and adapted to different situations, making it a flexible and versatile tool for completing tasks in various settings. For instance, the BT enables independent sub-tree testing to expand functionality. In this study, the sub-trees responsible for picking and placing tasks were tested individually and then integrated into a complete tree as functions. This structure also offers advantages in terms of code comprehension.

[2]https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html
[3]https://py-trees.readthedocs.io/en/devel/
[4]https://py-trees-ros.readthedocs.io/en/devel/

Figure 5.4 presents a generalized BT without specific implementation of behaviors in the leaf nodes. The tree is ticked regularly, with a tick occurring every $10ms$, during which the nodes are traversed from right to left. Flow control nodes in the tree can be categorized into three main types: sequence ($\rightarrow$), fallback (?), and decorator (δ). Each node returns a state when ticked that can either be SUCCESS, FAILURE, or RUNNING. Intermediate nodes return a state based on the return values of their children. A sequence node returns SUCCESS only if all its children return SUCCESS and can also remember the value of child nodes to avoid repeating them. These nodes are indicated as an arrow with an asterisk ($\rightarrow^{*}$). This node type is commonly used when a robot needs to execute a sequence of motions, such as when picking up an object. A fallback node returns SUCCESS when it finds the first child element that returns SUCCESS and is useful for creating conditions. The third type of node is a decorator node. A node, such as an *inverter*, is one of the decorator types. It inverts the resulting value. Executor nodes are action nodes (□) that perform an action and condition nodes (◯) that check some conditions.

The behavior tree operates according to task prioritization (goes from left to the right), where ensuring safety by avoiding collisions with a human takes precedence over task completion. The arrangement of actions in the behavior tree also governs the prerequisites for executing actions. The initial node of BT involves processing information about the scene at the start of each tick by gathering information from ROS topics. The information includes the positions of objects in the workspace, the presence of a human hand, and new tasks. The collected data is then stored on the Blackboard for all nodes. If a human is detected in the workspace, 'Human in scene' node returns SUCCESS, and the fallback node inverts it to FAILURE. Upon receiving

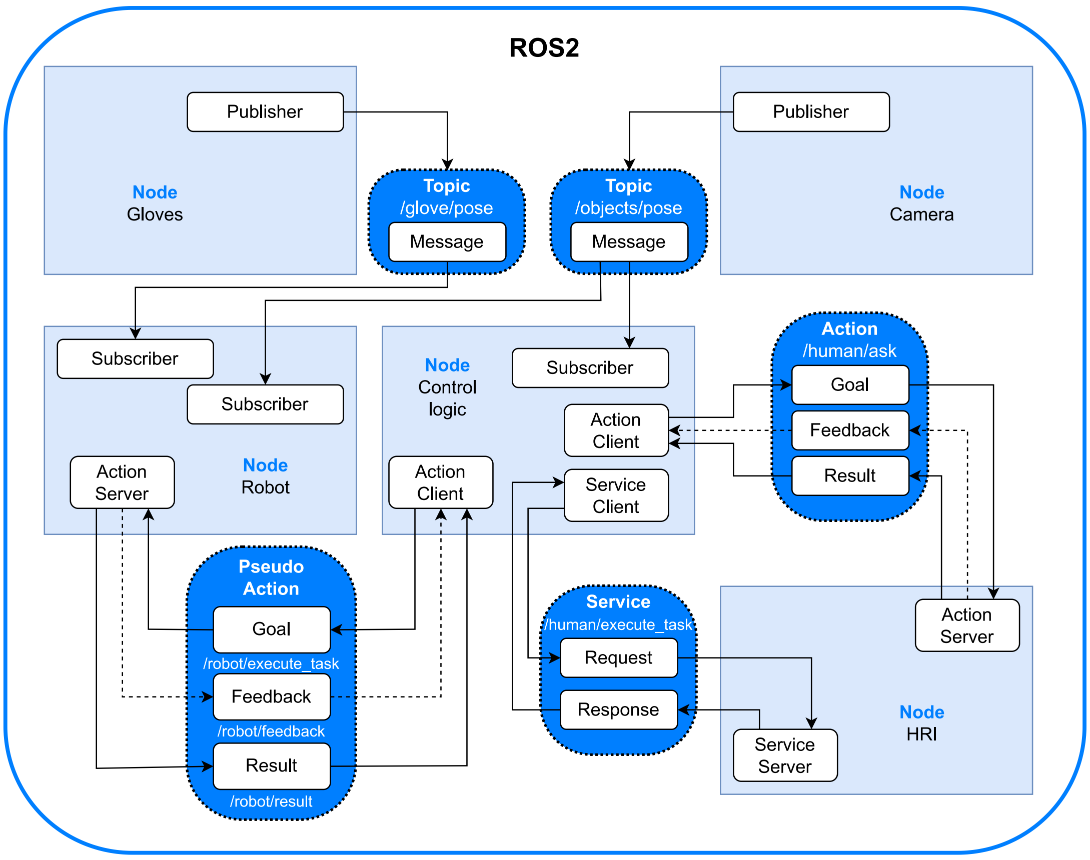


Figure 5.3 ROS architecture

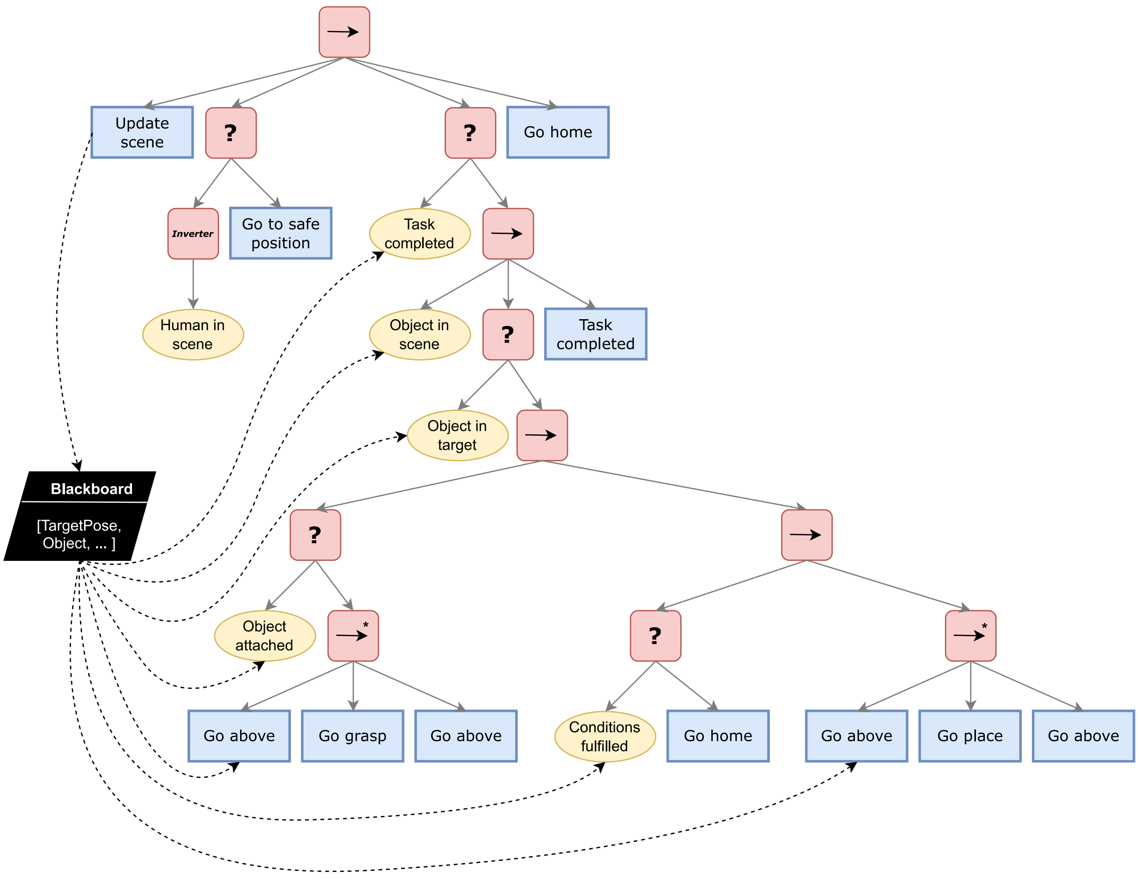
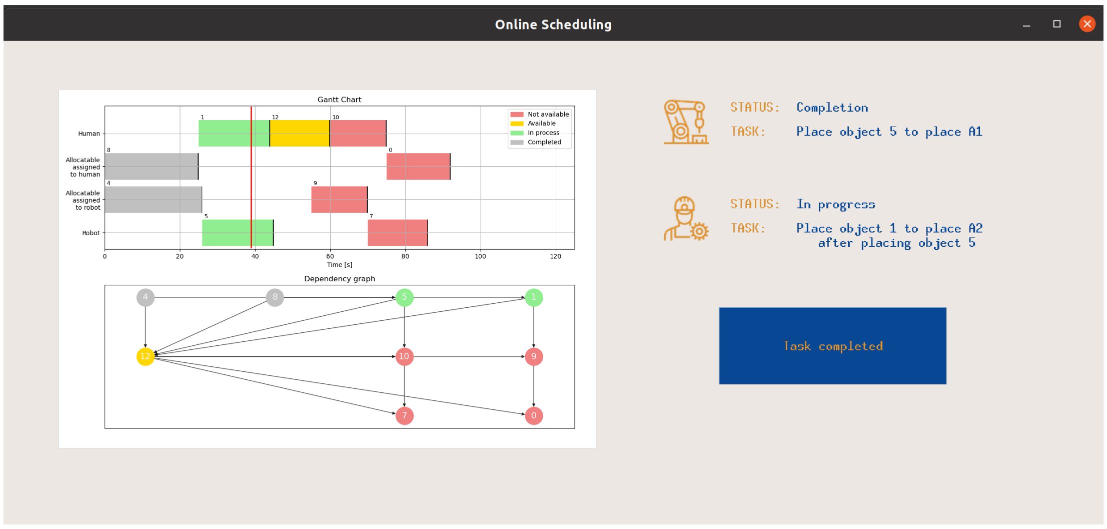


Figure 5.4 Behaviour three for robot motion

FAILURE, the fallback node executes the second child node, which is the execution node that moves the robot to a safe position. If there is no human, the fallback node returns SUCCESS, and its parent node proceeds to the next child node. If there is a task, the tree checks for the presence of the required object in the workspace. If no object is detected, the parent sequential node is received FAILURE, and searching stops in this node till the next tick. If the object is not in the target position, the system checks if the robot's gripper currently holds it. If it is not, the robot must pick it up. Once the robot has an object, the corresponding branch is executed in BT for placing it. If any conditions in the task specification need to be met before executing the task, such as the presence of another object, those conditions are checked first. The task is marked as completed if an object is in the target position.

Figure 5.4 displays only the basic robot actions. The real-world process includes smaller tasks, such as opening and closing the gripper and checking its state.

The final node in the root sequence handles the scenario where the robot has no pending tasks. In such cases, the robot is programmed to return to its designated home position to wait for the next task, ensuring it stays clear of human interference. However, suppose a new task is assigned to the robot in the next tick after completing the previous one. In that case, the robot skips the home position and proceeds directly to the new task without delay.



STATUS: Completion
TASK: Place object 5 to place A1

STATUS: In progress
TASK: Place object 1 to place A2 after placing object 5

Figure 5.5 GUI in the real setup. It includes a dynamic Gantt chart to display the task's current state and schedule. The dependency graph visualizes task-specific dependencies, while task details for the robot and user are shown on the right side. Human feedback on task completion or rejection is collected through buttons.

5.3.3 User interaction via a Graphical User Interface

A visual interface was developed to enable human interaction, and the main window is depicted in Figure 5.5. The GUI was programmed with the Tkinter library [5]. At the start of a new job, the main window displays a continuously updated schedule based on the current situation. A Gantt Chart with a tasks and task dependency graph is on the left-hand side. It informs the user of the robot's intentions and provides a general process overview. The red bar on the Gantt Chart shows the current time in the schedule dividing the observed history and the predicted future. Tasks are assigned colors based on their status (unavailable, available, in progress, and completed).The robot and the human tasks and status are shown on the right-hand side. The robot's status bar shows the phase of the task (preparation, execution, completion), pending conditions (waiting), the end of the task (success) or pending next task (available). The person's status bar shows task execution (in progress), task end (success), and waiting for the next task (available). The information displayed on the user interface, including the schedule visualization, is updated every second.

A pop-up window containing the task description appears if a non-allocated task request is received. After confirming acceptance, the window disappears. Upon completing the task, the user must confirm it by clicking the 'Task completed' button. In the event of an allocated task or a change in the schedule, a pop-up window appears with the option to accept or reject the task.

[5] https://docs.python.org/3/library/tk.html

Chapter 6

Experiments and Results

This chapter comprehensively overviews the conducted experiments and their corresponding outcomes. The experiments utilizing probabilistic simulation demonstrate efficiency, including faster execution time and predictability, as well as the schedule's and rescheduling's robustness inducted by uncertainties. Additionally, validating the proposed method in a real-world setting involves the development of a demo version without conducting systematic experiments.

6.1 Simulation

The primary objectives of conducting experiments using a simulation setup are twofold: first, to gauge the makespan and then compare it with the baseline task allocation algorithm, and second, to evaluate the system's capability to predict makespan. To evaluate against these criteria, 12600 experiments were performed in simulations with different initial conditions, described below.

6.1.1 Experimental setup

Each experiment with the proposed method involved running a simulation with varying parameters, such as the percentage of each agent's tasks and allocatable tasks, probability of error, and probability of human choice. All parameters are described in detail later in this section. The experimental process consisted of several steps. First, a schedule was created, specifying the assignment of tasks to agents. Then, the simulation simulated the execution of tasks by the agents, taking into account factors such as task duration, task dependencies, and human decision-making. If needed, rescheduling was evaluated and performed accordingly. The simulation continued until all tasks were completed.

In experiments, the scheduling utilizes the same task duration sampling algorithm as the simulation described in Chapter 5. Both the schedule and simulation draw task duration values from the same distributions. This approach is based on the concept of oracle estimation, where the assumed values are treated as the true values to estimate system behavior. In the context of these experiments, it involves using the distribution that closely matches the simulation distribution to generate the schedule.

The initial and final schedules were saved throughout each experiment to calculate the makespan, representing the maximum time required to complete all tasks. The makespan is measured in seconds. Its values were also recorded at each instance of rescheduling, providing insights into the accuracy of predictions made during the scheduling process. Additionally, data were collected on the number of task rejections by humans, allowing for an estimation of the impact of soft constraints.

To assess and compare the effectiveness of the proposed method, the baseline algorithm was also tested with the added feature of incorporating human choice [7]. The same set of experiments as for the proposed method was performed, and data were collected, such as the final makespan and the number of tasks and proposal rejections.

To standardize the experiments, a set of parameters was established to determine the factors influencing the complexity and reactivity of the schedule.

- **Scenario** The experiment includes six scenarios described in Table 3.2, with 16 tasks in each scenario. Scenario partitioning provides a systematic approach to testing the method under different kinds of dependency between tasks, which shows its robustness to the complexity of the work. Weighted allocations determined the number of non-allocatable tasks to each agent and allocatable tasks. This indicates the effectiveness of scheduling with allocatable tasks as opposed to scenarios without allocatable tasks due to the shorter makespan.

- **Weights** To cover a wider range of possible cases, weights were set up. It is responsible for the percentage of each agent's tasks and allocatable tasks in the job. Scenarios 1, 3, and 5 in Table 3.2 with non-allocatable tasks were tested with these weights:

  $$w_{1,3,5} = [(0.3, 0.7), (0.5, 0.5), (0.7, 0.3)] \tag{8}$$

  The first case has a distribution where 30% of all tasks are assigned to a human and 70% to a robot. Each scenario runs using all three weights. Scenarios 2, 4, and 6 in Table 3.2 with allocatable tasks have three weight components:

  $$w_{2,4,6} = [(0.4, 0.4, 0.2), (0.3, 0.3, 0.4), (0.2, 0.2, 0.6)] \tag{9}$$

  For the first case $(0.4, 0.4, 0.2)$, 80% of the tasks are allocated to the human and the robot (40% each). The remaining 20% is free to be allocated by the proposed method during the scheduling. The same interpretation applies to the rest of all cases of weight.

- **Fail probability** Various experiments involved different error probabilities for each task, including 0.1%, 0.5%, 1%, and 5%. The occurrence of errors led to a threefold increase in task time, resulting in delays for dependent tasks and the overall schedule. The inclusion of significant delays in the experimental data was beneficial, as it provided insights into the impact of deviations from the schedule, both minor and substantial.

- **Probability of human choice** The final variable in the simulation is the human's choice. The two options (general probability of rejection and probability of rejection for each task) are explained in detail in Section 4.1.2. For the first option, the experiment involves testing three different probability values for a person accepting. These probabilities are 80%, 50%, and 20%. This probability is assigned separately to each task with a different value in the second option.

Altogether, a total of 10 800 experiments were conducted, accounting for various combinations of parameters and scenarios. An additional set of experiments was performed, specifically focusing on including soft constraints in the schedule. These experiments exclusively examined scenarios involving allocatable tasks and the probability of accepting each task individually, amounting to a total of 1 800 experiments.

All tests were conducted on a computer with the following parameters:

- OS: Windows 10 Pro

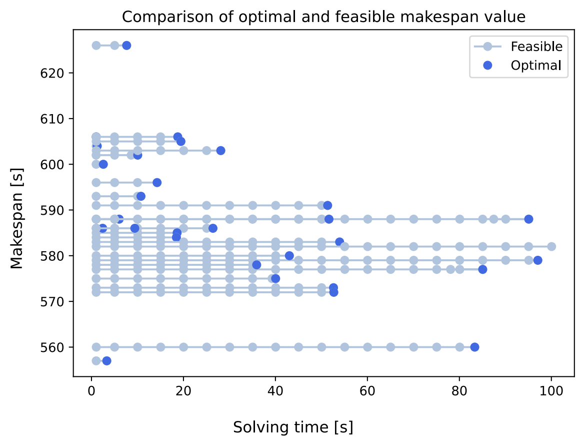


Figure 6.1 This figure depicts the measurements of the makespan of the job over the solving time. The initial search time limit was set to 1 second and increased in increments of 5 seconds. Notably, in 97% of the cases, the solver confirmed the optimality of the obtained solutions in the first second.

- CPU: Intel(R) Xeon(R) E3-1240 v5
  - Architecture: x64
  - Frequency: 3.50 GHz
  - Total Cores: 4
  - Total Threads: 8
  - L3 Cache: 8 MB
- Memory: 32 GB

A NumPy[1] library module is utilized for sampling probability distributions, and a random generator from the same library is used with a random seed set based on the iteration number.

6.1.2 Results

The experiments yielded a consistent outcome that demonstrated the accuracy of the constraints and reliability of the computations as the schedule and rescheduling were successfully generated in all iterations. The correctness of the schedule was tested manually with a sample of 30 experiments. The found schedule and rescheduling corresponded to the given constraints.

The solver itself evaluates the solution's optimality and indicates whether the solution is optimal or feasible once it is found. The distinction between feasible and optimal solutions was examined. Scenario 4 was executed without soft constraints and repeated ten times with varying weights. This scenario is particularly well-suited for testing, as it typically requires an average of more than 10 seconds to find the optimal solution. The increased search time can be attributed to the extensive solution search space resulting from the absence of constraints between tasks. The search process was constrained to a maximum duration of 100 seconds. Initially, the search was limited to 1 second and then increased in increments of 5 seconds. The makespan value achieved within each respective solving time limit was recorded for analysis. In 97% of the cases, the solver confirmed that the solution obtained in the first second was optimal. Figure 6.1 illustrates the measurements of the makespan of the job over solving time. It indicates that a feasible solution, which finds in the first second of solving, is optimal. The

[1]https://numpy.org/doc/stable/reference/random/generator.html

makespan of the job remains constant. This suggests that solutions obtained within the first seconds can be considered optimal.

Makespan

Makespan refers to the total duration or time required to complete a job. It measures the total time taken from the start of the first task to the completion of the last task. Makespan is often used as a performance metric in scheduling and optimization problems to evaluate the efficiency of different scheduling algorithms or strategies.

Table 6.1 presents the experiments' results to compare the methods makespan. The table is divided into two parts. The first includes data from experiments with scenarios 1, 2, and 3 (with non-allocatable tasks), where each scenario was tested with varying error probabilities (0.1%, 0.5%, 1%, and 5%). For each method, the median and variance of the makespan were calculated. The median is preferred over the mean in similar situations due to its resistance to outliers and ability to represent the central value in skewed or non-normal distributions accurately. It provides a more robust measure of central tendency. The next part of the table presents the results of scenarios 4, 5, and 6 (with allocatable tasks), divided into two groups - one where a human can accept or reject an assigned task or rescheduling proposal and the other without choice. Human choice in all these experiments was modeled using the overall probability of acceptance/rejection.

Scenarios		Fail probability	Schedule method		Baseline method	
			$\tilde{x}[s]$	σ^2	$\tilde{x}[s]$	σ^2
0, 1, 2 (with non-allocatable tasks)		0.1%	**600**	**3109.65**	623	7534.78
		0.5%	**600**	**3150.95**	623	7534.78
		1%	**600**	**3152.01**	623	7534.78
		5%	**764**	**8469.26**	780	15419.88
3, 4, 5 (with allocatable tasks)	with choice	0.1%	**662**	**5952.23**	671	6919.99
		0.5%	**663**	**6191.8**	671	6919.99
		1%	**662**	**6260.87**	671	6919.99
		5%	**809**	20.133.8	833	**19416.27**
	without choice	0.1%	**598**	**1601.27**	619	1655.83
		0.5%	**597**	**1518.01**	619	1655.83
		1%	**597**	**1550.8**	619	1655.83
		5%	**732**	**5421.3**	771	7454.64

Table 6.1 This table displays the outcomes of the experiments that compared the median $\tilde{x}$ and variance σ^2 of the schedule method with the base method.

The results of measurements with 1% and 5% error probability given in the Table 6.1 are also visualized in Figure 6.2. The visualization focuses on these specific error cases because measurements with error probabilities of 0.1%, 0.5%, and 1% yield similar results. Across all experiments, the proposed schedule method consistently exhibited a lower makespan median than the baseline method. The advantages of task allocation (scenarios 4, 5, and 6) are evident, as their proper distribution reduces makespan. Notably, the bar chart in Figure 6.2 demonstrates that the makespan values in the third row are considerably shifted towards the left compared to the first row, indicating improved reducing makespan.

Human choice directly influences the duration of the job. The median of the schedule method is not significantly lower than the baseline median in cases of error up to 1%. However, with a higher probability of error, the baseline method becomes inconsistent for the minimum makespan, which is evident in all situations.

The key difference between these methods is their decision-making process, as shown in Figure 6.3. The upper part depicts the solution of the baseline method, while the bottom part depicts the solution of the scheduling method. At the point when the robot completes task C, the only available task is task E. However, task D cannot start because it depends on task B, which has not yet begun execution. Both methods aim to find a task for the robot or leave

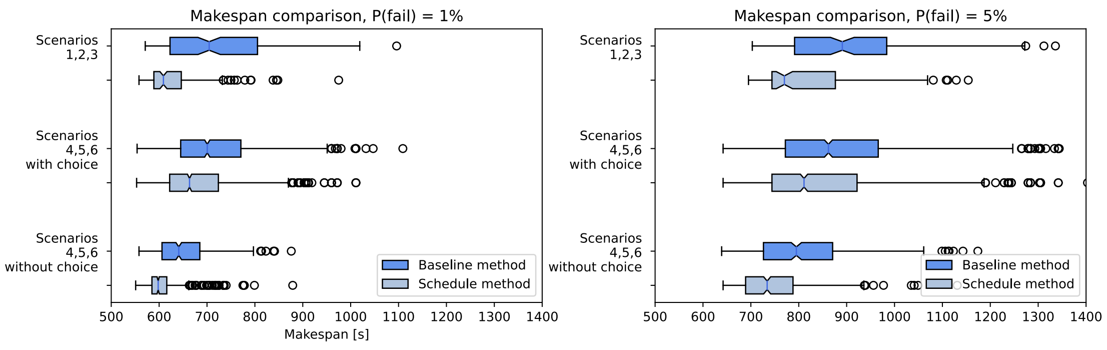


Figure 6.2 The provided graph depicts the varying makespan distribution across different scenarios. The results on the right side of the graph pertain to an experiment conducted with task phase error probabilities of 1% and 5%. It is observable that the proposed schedule method exhibits significantly better performance (lower makespan) than the baseline method across all scenarios. This demonstrates the efficacy of our approach in effectively leveraging task flexibility while also adeptly managing situations characterized by high levels of uncertainty.

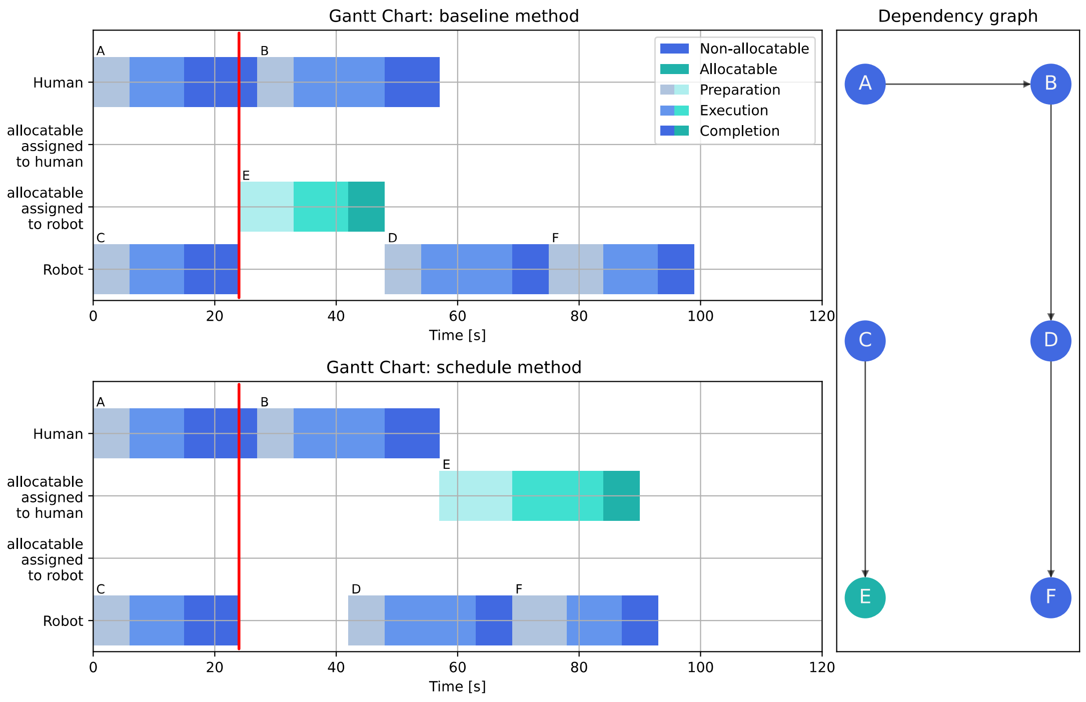


Figure 6.3 Comparison of decision-making processes in the baseline and scheduling methods. In time $24s$, the methods must decide whether to assign task E to the robot. The base method decides to assign; the schedule method does not. The makespan of the baseline method is $99s$, while the makespan of the scheduling method is $90s$, highlighting the impact of the evaluation of task allocation.

it idle while waiting for the next available task. The baseline method evaluates the execution time of the allocated task performed by both agents, which is 33s of time for a human and 24s for a robot. Therefore, it assigns the task to the robot since it is more profitable. On the other hand, the scheduling method estimates the makespan if the task is assigned to the robot. After rescheduling the task to another agent, the makespan increases beyond what was predicted earlier, making the rescheduling disadvantageous. As a result, the baseline method has a makespan of 99s, while the scheduling method has a makespan of 90s. This example is used to illustrate the principle being described. Although the difference in the makespan may seem small compared to the average time of tasks, it can have a significant impact, particularly if there are a series of incorrect decisions.

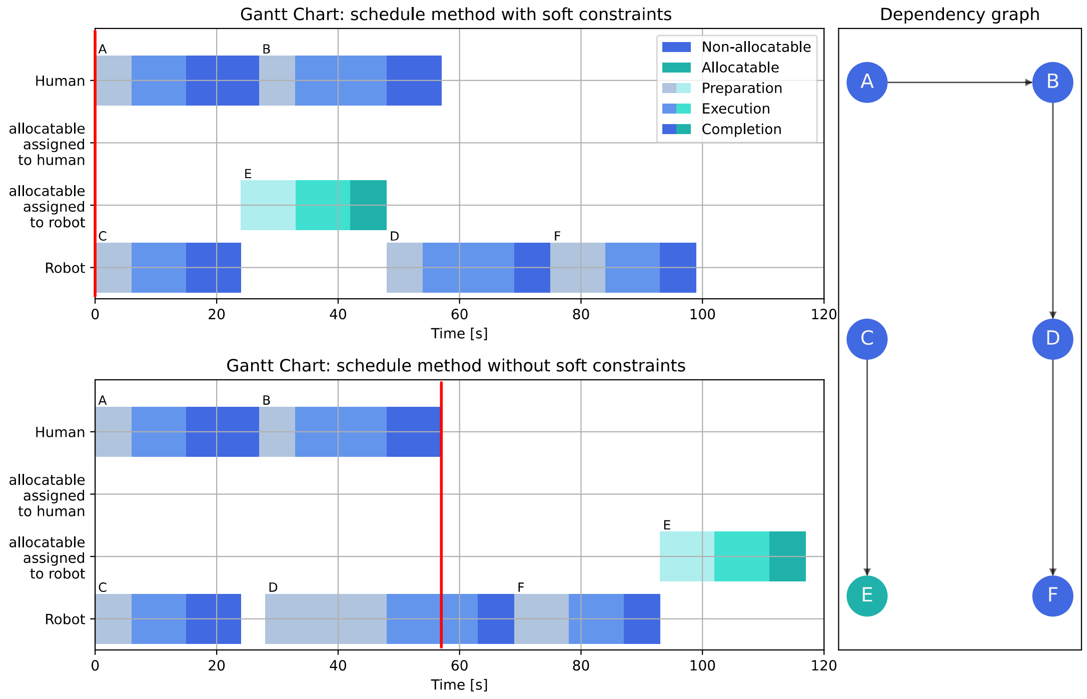


Figure 6.4 The comparison presented in the figure is between the schedule method with and without soft constraints. The upper part of the figure displays a schedule where there is an 80% chance of human rejection of task E, and this probability is taken into account while creating the schedule. In contrast, the lower part of the figure shows the schedule without considering the person's choice, and task E is assigned to the person who rejects it. This results in reassignment to the robot, which cause an offer of makespan.

This instance demonstrates the advantages of scheduling, but the example depicted in Figure 6.4 also highlights the limitations of the scheduling approach when a person's choice is involved. In the previous scenario, if the person had rejected task E, the completion time would have been longer than that of the baseline method. This inaccuracy may account for the higher variance at a high error probability of 5% as presented in Table 6.1.

The previously mentioned error can be prevented by incorporating soft constraints based on human preferences during schedule creation. For instance, if the chance of task E being rejected is 80%, the initial schedule would be the same as obtained through the baseline method, as demonstrated in Figure 6.3. On the other hand, if the probability of rejection is small enough, the schedule assigns it to the human. Even if the task is declined, the schedule cannot reduce

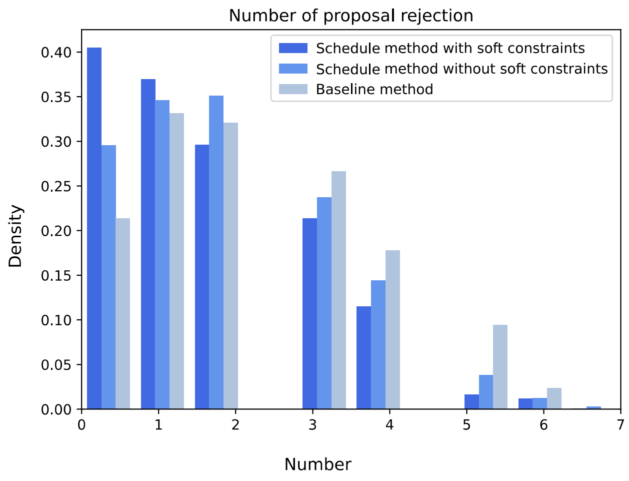


Figure 6.5 This histogram shows the density of rejection rates from a task or rescheduling proposal, depending on the method. The depicted data are a sample from experiments conducted with a 1% probability of error through scenarios 4, 5 and 6.

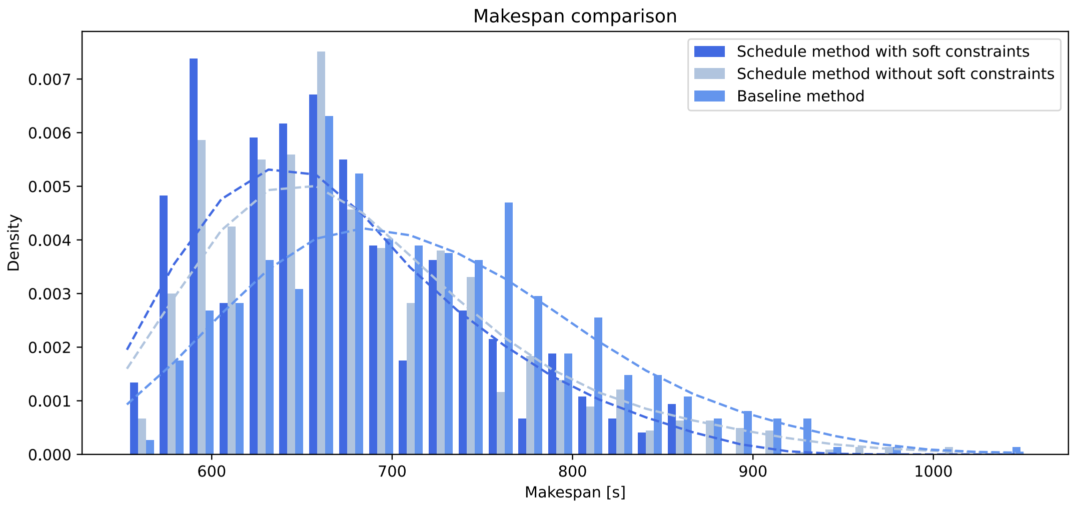


Figure 6.6 The histogram displays the contrast in job completion time between the base method and the schedule method with and without soft constraints. The depicted data are a sample from experiments conducted with a 1% probability of error through scenarios 4, 5 and 6.

makespan. Still, the number of failures can be reduced by utilizing soft constraints, resulting in lower failure rates as demonstrated in histogram on Figure 6.5. Including soft constraints positively impacts the makespan it takes to execute a task, as demonstrated in Figure 6.6. The scheduling approach that does not incorporate soft constraints has a higher median and variance; the same works for the baseline method.

This reduction in failures also contributes to better predictability of makespan, which is discussed in the following section.

Predictability

The reactive schedule is characterized by its ability to adapt to changes in underlying assumptions, which is a significant advantage. In the case of this work, the triggers for schedule changes are either a person's rejection to complete a task or a delay in task completion render-

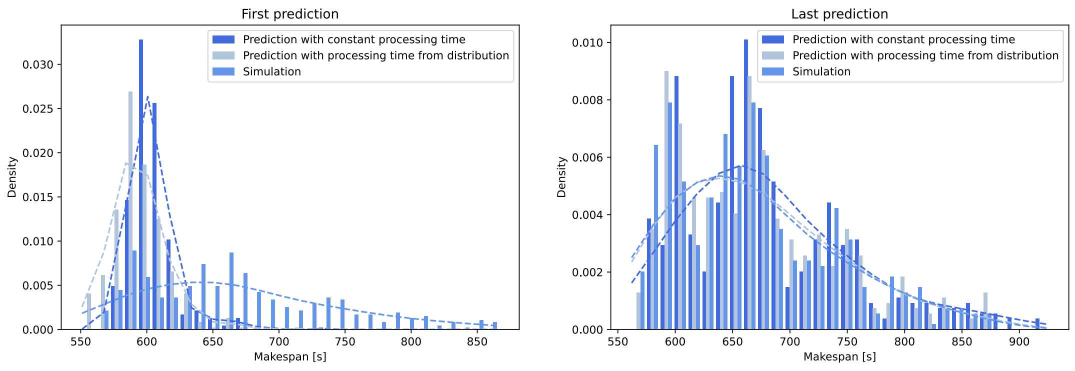


Figure 6.7 The histogram shows the difference between the first and the last prediction of makespan on the right and left sides, respectively. These histograms suggest a better ability to predict job duration when using probability distribution in scheduling than fixed values. The depicted data are a sample from experiments conducted with a 1% probability of error through scenarios 4, 5 and 6.

ing the original schedule unreliable for predicting makespan. This unreliability is illustrated by the left histogram in Figure 6.7, where the assumed values differ significantly from the simulation. However, updating the schedule during job execution brings it closer to reality. The right histogram in the same figure shows that the most recent schedule generated is almost identical to the simulation, demonstrating the benefit of the reactive schedule in the prediction context.

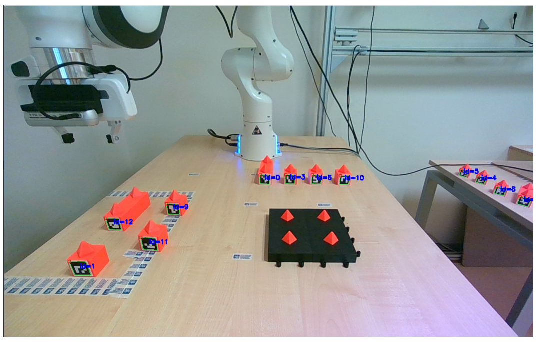
The advantage of employing a probability distribution for task process time in scheduling instead of a fixed value is demonstrated in Figure 6.7. While this aspect may not substantially impact the scheduling efficiency, it holds great significance in aligning the scheduled makespan with the actual one. Comparing scheduled makespan with a constant value, the variance is lower in the initial schedule than in scheduling the case of sampling from a probability distribution. However, when considering the last prediction, a notable disparity emerges between these two predictions and the simulation. The outcomes obtained from a schedule incorporating probability closely align with the simulation results compared to a schedule utilizing a constant value. This substantiates the notion that using a probability distribution in scheduling leads to more accurate predictions of the makespan of a job.

6.2 Real robot experiments

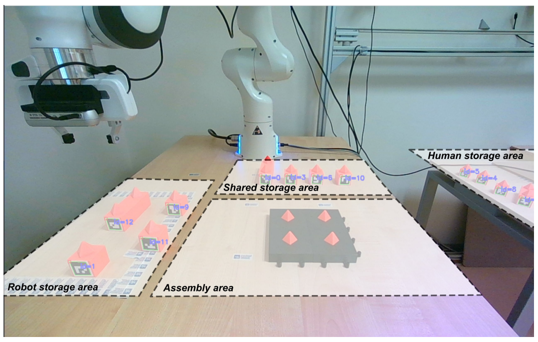
Integrating an online scheduling-based task distributor with a physical Franka Emika Panda robot demonstrates the validity of the proposed approach on several test use cases from the literature [45]. The demo (see video at `https://drive.google.com/drive/folders/1A7rOCF-76yVHVAD3dNUWscwVSHBHpdNQ?usp=sharing`)effectively highlights both the reactiveness of the schedule and the robot's control achieved through utilizing a behavior tree.

6.2.1 Use case

The experiment on a real robot was designed to test HRC scenarios using a simple implementation that does not require additional efforts to adjust the robot's accuracy, such as using 3D-printed assembly [45]. The setup depicted on Figure 6.8 includes an assembly area represented by a black construction plate, a part storage area for the robot on the left, a shared part storage area adjacent to the robot base, and a part storage area designated for the worker on the right.

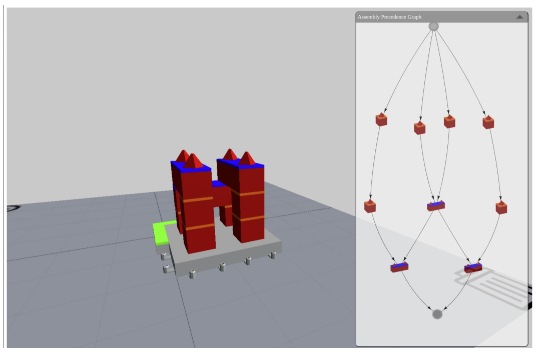


a) The initial state with example task based on the HRC task generator [45]

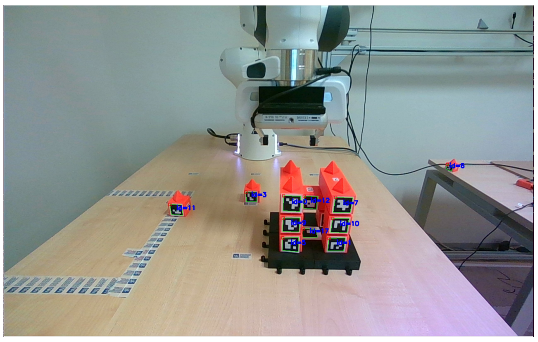


b) Setting up a workspace for assigning objects to the allocatable and non-allocatable task type.

Figure 6.8 The real robot setup with the delimitation of workspace.



a) The goal of the shared assembly task modeling in Benchmark Builder [45]. On the right, the precedence graph is shown

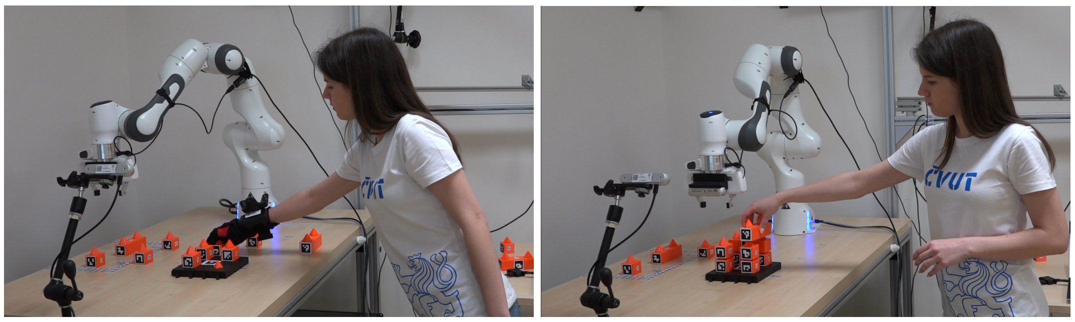


b) The final state of HRC demo task on real robot setup.

Figure 6.9 The real robot setup with a demonstration of the goal and final state of the shared assembly task.

The assembly task presented in Figure 6.9 involves a collaborative effort between a human worker and a robot. Task durations for the robot and the human were measured by conducting multiple repetitions, totaling 20 measurements for each task. Due to the similar complexity of the tasks and the close proximity of the objects, the average duration for task execution by the robot was 21 seconds. Conversely, the average duration for task completion by a human was 14 seconds. These recorded durations were then used to create a probability distribution model, which forms the foundation for task scheduling. It is not strictly necessary to know a precise distribution. The schedule remains relevant and valid, as demonstrated in the results in Section 6.1.2, even with an approximately constant value or an imprecise estimation of the distribution of task execution time. The control logic governing the task assignment dynamically distributes specific tasks based on the schedule outlined in Figure 6.9. This scheduling approach ensures that each human or robot participant receives the suitable task appropriately, optimizing their collaboration and coordination during the assembly task.

The worker and the robot have access to the assembly and shared part storage areas simultaneously. Safety measures are ensured through the collaborative robot mode, which stops the motors upon detecting any external force on the robot's joints. However, gloves can be utilized for direct manipulation within the robot's target position if needed. Suppose the system detects a glove above the table. In that case, the robot receives a command to move away from that area, enabling manipulation in the overall workspace without the physical presence of the



a) Alternate use of the workspace. By incorporating glove-based safety measures, the robot detects its presence within the workspace. It promptly moves away to a secure location, where it remains until the glove is no longer within the workspace.

b) Simultaneous use of the workspace. Providing safety solely through the collaborative mode of the robot during work without utilizing gloves and without requiring the robot to be relocated to a secure area.

Figure 6.10 Examples of using assembly area.

robot.

6.2.2 Robot and schedule restiveness

In the context of robotics, reactiveness refers to the ability of a robot to perceive and respond quickly to changes in its environment or external stimuli. Two tests were conducted to evaluate the system's restiveness.

The first test was object pursuit, where the robot was tasked with picking up and placing a cube on top of another one using only the ID of the objects as the specification. The robot could follow the cube's position even as it moved and detect attachment during all task pickup, taking necessary actions to perform it. Another cube was set as the target destination, and even when its position on the table was changed, the robot could complete the task. In the absence of the cube, the robot waited for the execution of the conditions in the home position. The robot's movements during testing were smooth and corresponded to the specified behavior.

The second test was safety. Virtual reality gloves were used to impose safety constraints, and when a human hand appeared in the workspace, the robot moved away to a pre-defined safe place. The reaction to the safety zone violation was instantaneous, and the robot remembered its state after returning to the task and continued to perform it. These tests demonstrate the system's ability to react dynamically to changing conditions and perform tasks safely and efficiently.

The schedule demonstrated reactivity as expected, not only in cases of significant delays but also in optimizing the execution time of the work whenever possible. An illustrative example of this is depicted in Figure 6.11, where the last task 0 was reassigned to the robot based on the rescheduling evaluation, aiming to minimize the makespan considering the actual delay caused by the human.

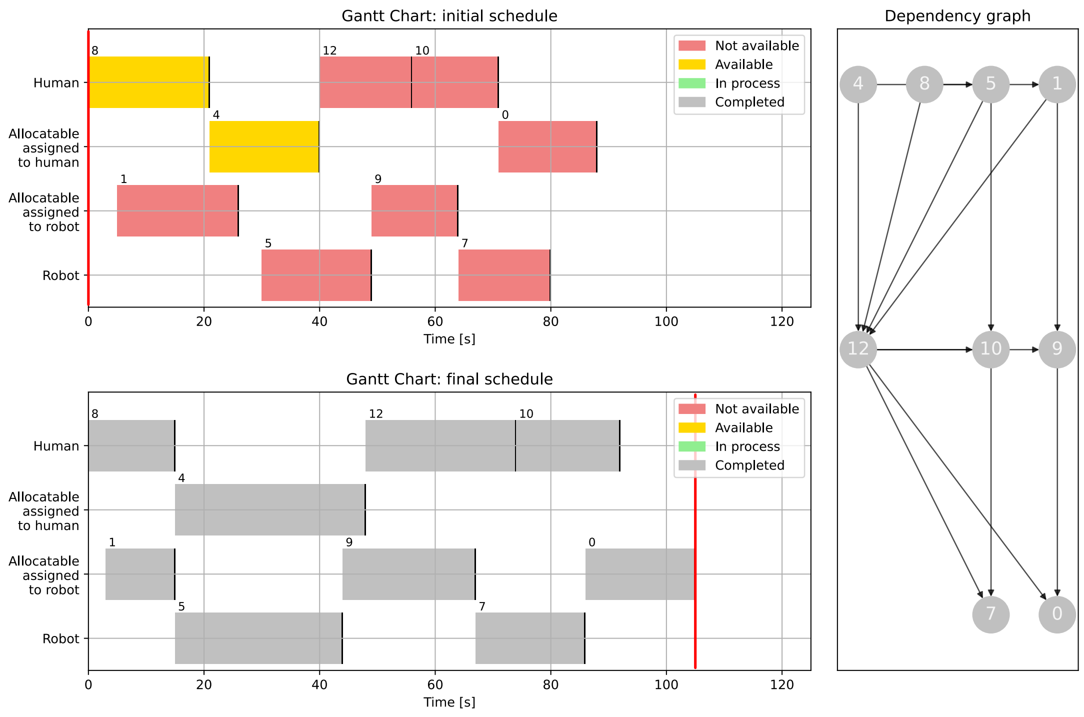


Figure 6.11 The initial and final schedules of the demonstration with a real robot. It presents the optimization of task allocation achieved through rescheduling considering the actual delay. Specifically, in this instance, the last task 0 was redistributed from the originally assigned human agent to the robot, aiming at makespan minimization.

Chapter 7

Conclusion

This work has addressed the challenges associated with the dynamic nature of the Human-Robot Collaboration (HRC) environment. The main aim was to develop efficient scheduling approaches in HRC settings where humans and robots work towards a common goal. By considering the limited controllability and inherent uncertainty of the environment, the thesis emphasizes the need for scheduling and acting that considers uncertainty and allows for continuous monitoring and updating of foundational assumptions.

The task assignment of the collaboration was structured to minimize inaccuracies during execution. This involved establishing a job hierarchy and analyzing task dependencies. Further, uncertainties related to human involvement were identified and modeled based on action controllability and observability. The next step was to incorporate this model data into the scheduling process.

Scheduling problem statements and solutions were provided by Constraint Programming (CP). The solution considers task requirements and system uncertainties such as task duration or human choice. Allowing individuals to select their tasks represents a significant innovation in this study. This approach is driven by ethical considerations and the aim to alleviate monotony in the production process. Monotony negatively impacts the mental well-being of workers, as does the stress associated with collaborating with robots. When designing the task distribution algorithm, emphasis was placed on creating a clear and intuitive algorithm that minimizes deviations from the planned schedule. A task allocation algorithm based on the schedule was proposed, with adjustments and rescheduling performed to react to plan changes. Probabilistic simulations were conducted to evaluate the efficiency and reliability of the approach compared to a baseline method.

The task allocation algorithm proposed in this study was implemented in a physical robotic workspace using the Franka Emika Panda robot. The integration presented a demonstration of HRC in executing a specific use case. The algorithm was complemented by implementing a Behavior Tree (BT), enabling reactive control of the robot and facilitating quick and safe adaptations to environmental changes. Object detection was also implemented to evaluate the workspace, while a graphical user interface GUI was developed for human instructions.

This work presents a promising approach for HRC management that considers human well-being while achieving high production autonomy in Industry 4.0, which was further acknowledged with the submission of a workshop paper as an additional result.

7.1 Future work

This research lays the foundation for the application of HRC in various fields. It offers a range of improvements and expansions in different areas that can be implemented:

- **Scheduling**
  - In certain scenarios, the robot or human may require tool changes or the use of additional instruments to perform coherent manipulations. The proposed model allows an extension to the scheduling functionality, considering the time required for tool change. This enables the evaluation of task sequences based on the optimal number of tool exchanges.
  - Investigate the relationship between schedule robustness and the arrangement of allocatable tasks during job progression. The underlying principle is that assigning tasks closer to the end of the job enhances flexibility and reduces job makespan.
- **Reactive control**
  - While the behavior tree has demonstrated impressive reactiveness, it is currently limited to a predetermined set of actions, specifically Pick&Place, for the scope of this thesis. To enhance the capabilities of the behavior tree, the integration of learning algorithms is being explored, which is a current area of focus. By incorporating these algorithms, the behavior tree can be expanded to adapt and learn new sequences of actions, offering further potential for improvement.
  - The system's efficiency is influenced by human feedback, such as confirmation of job completion. However, it is feasible to eliminate the need for waiting for an explicit acknowledgment of job completion. This can be achieved by assessing the assembly progress or monitoring the gloves' movements. By tracking and analyzing human movements, the system can ensure safety and evaluate the progress of the human task.
- **Prediction and uncertainty**
  - In this thesis, the initial assumption was that statistical information about the tasks and agents was provided. However, a more realistic approach is to begin with a rough prior and refine it based on incoming observations. This expanded approach enhances the applicability of the proposed method.
  - In order to enhance the impact of soft constraints on predicting human choices and dynamically adapting schedules, it is crucial to investigate the causal relationship of human decision-making. Developing a model of human behavior based on this research data will enable further reduction of task rejections. This will not only facilitate a more comfortable workspace but also facilitate the integration of robot-human collaboration into the production process by minimizing uncertainties.
- **Human-Robot Interaction**
  - To assess the human perception of collaboration in the system, user studies with a larger number of participants should be conducted to evaluate how humans perceive collaboration. These studies will involve structured experiments with volunteers to gather data on stress levels during robot interactions, the intuitiveness of decision-making and movements, and other relevant factors that guide system improvement.
  - Direct interviews with individuals actively working on production lines provide valuable insights into challenges, limitations, and areas for improvement in human-robot interaction, covering topics such as communication, safety, workflow efficiency, and task allocation. This qualitative approach enhances understanding and enables targeted enhancements for improved interaction and system performance.

Bibliography

[1] Alfred D. Chandler. *The Visible Hand: The Managerial Revolution in American Business*. Harvard University Press, 1977. ISBN: 9780674940529. URL: http://www.jstor.org/stable/j.ctvjghwrj (visited on 01/21/2023).

[2] Andrea Teresa Espinoza Pérez et al. "Mass customized/personalized manufacturing in Industry 4.0 and blockchain: Research challenges, main problems, and the design of an information architecture". In: *Information Fusion* 79 (2022), pp. 44–57. ISSN: 1566-2535. DOI: https://doi.org/10.1016/j.inffus.2021.09.021.

[3] Mohammed-Amine Abdous et al. "Assembly line balancing problem with ergonomics: a new fatigue and recovery model". In: *International Journal of Production Research* 61.3 (2023), pp. 693–706. DOI: 10.1080/00207543.2021.2015081. eprint: https://doi.org/10.1080/00207543.2021.2015081. URL: https://doi.org/10.1080/00207543.2021.2015081.

[4] Thierry Yonga Chuengwa et al. "Research Perspectives in Collaborative Assembly: A Review". In: *Robotics* 12.2 (2023). ISSN: 2218-6581. DOI: 10.3390/robotics12020037. URL: https://www.mdpi.com/2218-6581/12/2/37.

[5] Andrea Pupa et al. "A Resilient and Effective Task Scheduling Approach for Industrial Human-Robot Collaboration". In: *Sensors* 22.13 (2022). ISSN: 1424-8220. DOI: 10.3390/s22134901. URL: https://www.mdpi.com/1424-8220/22/13/4901.

[6] Nikolaos Nikolakis et al. "Dynamic scheduling of shared human-robot manufacturing operations". In: *Procedia CIRP* 72 (2018). 51st CIRP Conference on Manufacturing Systems, pp. 9–14. ISSN: 2212-8271. DOI: https://doi.org/10.1016/j.procir.2018.04.007. URL: https://www.sciencedirect.com/science/article/pii/S2212827118305225.

[7] Christoph Petzoldt et al. "Implementation and Evaluation of Dynamic Task Allocation for Human-Robot Collaboration in Assembly". In: *Applied Sciences* 12.24 (2022). ISSN: 2076-3417. DOI: 10.3390/app122412645. URL: https://www.mdpi.com/2076-3417/12/24/12645.

[8] Zukui Li and Marianthi Ierapetritou. "Process scheduling under uncertainty: Review and challenges". In: *Computers & Chemical Engineering* 32.4 (2008). Festschrift devoted to Rex Reklaitis on his 65th Birthday, pp. 715–727. ISSN: 0098-1354. DOI: https://doi.org/10.1016/j.compchemeng.2007.03.001. URL: https://www.sciencedirect.com/science/article/pii/S0098135407000580.

[9] Soraya Izghouti et al. "A Genetic Algorithm for tasks allocation and sequencing in a human robot assembly system". In: *2022 19th International Multi-Conference on Systems, Signals & Devices (SSD)*. 2022, pp. 2186–2191. DOI: 10.1109/SSD54932.2022.9955749.

[10] Yee Yeng Liau and Kwangyeol Ryu. “Task Allocation in Human-Robot Collaboration (HRC) Based on Task Characteristics and Agent Capability for Mold Assembly”. In: *Procedia Manufacturing* 51 (2020). 30th International Conference on Flexible Automation and Intelligent Manufacturing (FAIM2021), pp. 179–186. ISSN: 2351-9789. DOI: https://doi.org/10.1016/j.promfg.2020.10.026. URL: https://www.sciencedirect.com/science/article/pii/S2351978920318813.

[11] Yee Yeng Liau and Kwangyeol Ryu. “Genetic algorithm-based task allocation in multiple modes of human–robot collaboration systems with two cobots”. In: *The International Journal of Advanced Manufacturing Technology* 119.11 (Apr. 2022), pp. 7291–7309. ISSN: 1433-3015. DOI: 10.1007/s00170-022-08670-x. URL: https://doi.org/10.1007/s00170-022-08670-x.

[12] Riccardo Maderna et al. “Flexible scheduling and tactile communication for human–robot collaboration”. In: *Robotics and Computer-Integrated Manufacturing* 73 (2022), p. 102233. ISSN: 0736-5845. DOI: https://doi.org/10.1016/j.rcim.2021.102233. URL: https://www.sciencedirect.com/science/article/pii/S0736584521001150.

[13] Andrea Casalino et al. “Optimal Scheduling of Human–Robot Collaborative Assembly Operations With Time Petri Nets”. In: *IEEE Transactions on Automation Science and Engineering* 18.1 (2021), pp. 70–84. DOI: 10.1109/TASE.2019.2932150.

[14] Marco Faroni et al. “A Layered Control Approach to Human-Aware Task and Motion Planning for Human-Robot Collaboration”. In: *2020 29th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*. 2020, pp. 1204–1210. DOI: 10.1109/RO-MAN47096.2020.9223483.

[15] Shufei Li et al. “Proactive human–robot collaboration: Mutual-cognitive, predictable, and self-organising perspectives”. In: *Robotics and Computer-Integrated Manufacturing* 81 (2023), p. 102510. ISSN: 0736-5845. DOI: https://doi.org/10.1016/j.rcim.2022.102510. URL: https://www.sciencedirect.com/science/article/pii/S0736584522001922.

[16] Meng-Lun Lee et al. “Task allocation and planning for product disassembly with human–robot collaboration”. In: *Robotics and Computer-Integrated Manufacturing* 76 (2022), p. 102306. ISSN: 0736-5845. DOI: https://doi.org/10.1016/j.rcim.2021.102306. URL: https://www.sciencedirect.com/science/article/pii/S0736584521001861.

[17] Nigora Gafur et al. “Dynamic path planning and reactive scheduling for a robotic manipulator using nonlinear model predictive control”. In: *2022 30th Mediterranean Conference on Control and Automation (MED)*. 2022, pp. 604–611. DOI: 10.1109/MED54222.2022.9837147.

[18] Jan Kristof Behrens. *Integrated task and motion scheduling for flexible manufacturing*. 2022. DOI: 10.26092/elib/1801.

[19] Cristiane Ferreira, Gonçalo Figueira, and Pedro Amorim. “Scheduling Human-Robot Teams in collaborative working cells”. In: *International Journal of Production Economics* 235 (2021), p. 108094. ISSN: 0925-5273. DOI: https://doi.org/10.1016/j.ijpe.2021.108094. URL: https://www.sciencedirect.com/science/article/pii/S0925527321000700.

[20] Burak Gökgür, Brahim Hnich, and Selin Özpeynirci. “Parallel machine scheduling with tool loading: a constraint programming approach”. In: *International Journal of Production Research* 56.16 (2018), pp. 5541–5557. DOI: 10.1080/00207543.2017.1421781. eprint: https://doi.org/10.1080/00207543.2017.1421781. URL: https://doi.org/10.1080/00207543.2017.1421781.

[21] Xinyu Li et al. “Dynamic task reallocation in human-robot collaborative workshop based on online biotic fatigue detection”. In: *2022 IEEE 18th International Conference on Automation Science and Engineering (CASE)*. 2022, pp. 116–122. DOI: 10.1109/CASE49997.2022.9926591.

[22] Chiu-Hsiang Lin et al. “Human-robot collaboration empowered by hidden semi-Markov model for operator behaviour prediction in a smart assembly system”. In: *Journal of Manufacturing Systems* 62 (2022), pp. 317–333. ISSN: 0278-6125. DOI: https://doi.org/10.1016/j.jmsy.2021.12.001. URL: https://www.sciencedirect.com/science/article/pii/S0278612521002508.

[23] Zhanrui Liao, Longsheng Jiang, and Yue Wang. “A quantitative measure of regret in decision-making for human-robot collaborative search tasks”. In: *2017 American Control Conference (ACC)*. 2017, pp. 1524–1529. DOI: 10.23919/ACC.2017.7963169.

[24] Robert Glaubius et al. “Scheduling Design with Unknown Execution Time Distributions or Modes”. In: 2009.

[25] Michele Colledanchise and Petter Ögren. *Behavior Trees in Robotics and AI*. CRC Press, July 2018. DOI: 10.1201/9780429489105. URL: https://doi.org/10.1201%2F9780429489105.

[26] Kevin French et al. “Learning Behavior Trees From Demonstration”. In: *2019 International Conference on Robotics and Automation (ICRA)*. 2019, pp. 7791–7797. DOI: 10.1109/ICRA.2019.8794104.

[27] Matteo Iovino et al. “A survey of Behavior Trees in robotics and AI”. In: *Robotics and Autonomous Systems* 154 (2022), p. 104096. ISSN: 0921-8890. DOI: https://doi.org/10.1016/j.robot.2022.104096. URL: https://www.sciencedirect.com/science/article/pii/S0921889022000513.

[28] Lu Lu et al. “Mental stress and safety awareness during human-robot collaboration - Review”. In: *Applied Ergonomics* 105 (2022), p. 103832. ISSN: 0003-6870. DOI: https://doi.org/10.1016/j.apergo.2022.103832. URL: https://www.sciencedirect.com/science/article/pii/S0003687022001557.

[29] Tim Jeske, Sebastian Terstegen, and Catharina Stahn. “Opportunities of Digitalization and Artificial Intelligence for Occupational Safety and Health in Production Industry”. In: *Digital Human Modeling and Applications in Health, Safety, Ergonomics and Risk Management. Human Body, Motion and Behavior*. Ed. by Vincent G. Duffy. Cham: Springer International Publishing, 2021, pp. 43–57. ISBN: 978-3-030-77817-0.

[30] Arsha Ali, Dawn Tilbury, and Lionel Robert. “Considerations for Task Allocation in Human-Robot Teams”. In: (Oct. 2022). DOI: 10.48550/arXiv.2210.03259.

[31] Sara J. Czaja and Sankaran N. Nair. “Human Factors Engineering and Systems Design”. In: *Handbook of Human Factors and Ergonomics*. John Wiley & Sons, Ltd, 2012. Chap. 2, pp. 38–56. ISBN: 9781118131350. DOI: https://doi.org/10.1002/9781118131350.ch2.

[32] Mark R. Lehto and Bradley T. Cook. "Occupational Health and Safety Management". In: *Handbook of Human Factors and Ergonomics*. John Wiley & Sons, Ltd, 2012. Chap. 25, pp. 699–733. ISBN: 9781118131350. DOI: https://doi.org/10.1002/9781118131350.ch25.

[33] Peter A. Hancock et al. "A Meta-Analysis of Factors Affecting Trust in Human-Robot Interaction". In: *Human Factors* 53.5 (2011). PMID: 22046724, pp. 517–527. DOI: 10.1177/0018720811417254.

[34] Wonjoon Kim et al. "Factors affecting trust in high-vulnerability human-robot interaction contexts: A structural equation modelling approach". In: *Applied Ergonomics* 85 (2020), p. 103056. ISSN: 0003-6870. DOI: https://doi.org/10.1016/j.apergo.2020.103056.

[35] Zahra Rezaei Khavas, S. Reza Ahmadzadeh, and Paul Robinette. "Modeling Trust in Human-Robot Interaction: A Survey". In: *Social Robotics*. Ed. by Alan R. Wagner et al. Cham: Springer International Publishing, 2020, pp. 529–541.

[36] Robert Michael Zlot. "An Auction-Based Approach to Complex Task Allocation for Multirobot Teams". PhD thesis. Pittsburgh, PA: Carnegie Mellon University, Dec. 2006.

[37] Omar Al-Buraiki and Pierre Payeur. "Probabilistic Task Assignment for Specialized Multi-Agent Robotic Systems". In: June 2019, pp. 1–7. DOI: 10.1109/ROSE.2019.8790420.

[38] Fabian Ranz, Vera Hummel, and Wilfried Sihn. "Capability-based Task Allocation in Human-robot Collaboration". In: *Procedia Manufacturing* 9 (2017). 7th Conference on Learning Factories, CLF 2017, pp. 182–189. ISSN: 2351-9789. DOI: https://doi.org/10.1016/j.promfg.2017.04.011.

[39] G. Ayorkor Korsah, Anthony Stentz, and M. Bernardine Dias. "A comprehensive taxonomy for multi-robot task allocation". In: *The International Journal of Robotics Research* 32.12 (2013), pp. 1495–1512. DOI: 10.1177/0278364913496484.

[40] Alessandro Roncone, Olivier Mangin, and Brian Scassellati. "Transparent role assignment and task allocation in human robot collaboration". In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. 2017, pp. 1014–1021. DOI: 10.1109/ICRA.2017.7989122.

[41] Andrea Maria Zanchettin et al. "Prediction of Human Activity Patterns for Human–Robot Collaborative Assembly Tasks". In: *IEEE Transactions on Industrial Informatics* 15.7 (2019), pp. 3934–3942. DOI: 10.1109/TII.2018.2882741.

[42] Michael F. Gorman John J. Kanet Sanjay L. Ahire. *Constraint Programming for Scheduling*. CRC Press, 2004. ISBN: 9781584883975. URL: https://core.ac.uk/download/pdf/232825873.pdf (visited on 04/30/2023).

[43] Sebastian Schuetz and Viswanath Venkatesh. "The Rise of Human Machines: How Cognitive Computing Systems Challenge Assumptions of User-System Interaction". In: *Journal of the Association for Information Systems* 21.2 (2020), pp. 460–482.

[44] Jason W. Burton, Mari-Klara Stein, and Tina Blegind Jensen. "A systematic review of algorithm aversion in augmented decision making". In: *Journal of Behavioral Decision Making* 33.2 (2020), pp. 220–239. DOI: https://doi.org/10.1002/bdm.2155.

[45] Dominik Riedelbauch and Jonathan Hümmer. “A Benchmark Toolkit for Collaborative Human-Robot Interaction”. In: *2022 31st IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*. Aug. 2022, pp. 806–813. DOI: `10.1109/RO-MAN53752.2022.9900790`.