Abstract

Over the past decades, industrial robotics has transitioned from fixed, single-purpose machines to flexible, collaborative mobile systems capable of navigating complex factory environments. Today's manufacturing demands, driven by labor scarcity, the need for rapid reconfiguration, and advances in AI and sensing, require robots to perform increasingly sophisticated, non-repetitive tasks alongside human workers. Designing and executing efficient multi-robot missions in such dynamic, human-in-the-loop settings presents multiple challenges: expressing high-level production requirements in a planner-friendly way, handling unexpected execution errors, scaling to large task allocations, and accounting for uncertainties in task durations and human behavior.

This thesis introduces an intuitive task modeling formalism and a suite of algorithmic methods that address these challenges end-to-end. First, we propose a domain-expert-friendly syntax for defining single-robot production missions, automatically generating problem definitions compatible with diverse off-the-shelf planners. To support rapid recovery from errors, we present task roadmaps, a novel planning algorithm that reuses the original search tree to accelerate replanning when execution deviates. We extend the formalism to a multi-robot kitting use case with alternative task locations and introduce a scalable, clustering-based approach to maintain computational tractability.

Recognizing the inherent uncertainties of human-robot collaboration, we further develop a collaborative stochastic task planning framework that integrates human risk preferences and models variability in task and routing durations. Finally, we tackle a collaborative production scenario with complex cross-schedule dependencies, proposing a stochastic scheduling method that generates optimized, deadlock-free plans while balancing efficiency with human well-being.

Extensive simulations and experiments grounded in real-world applications demonstrate that our methods significantly improve planning efficiency, robustness, and adaptability in dynamic industrial settings, paving the way toward more resilient, human-centric robotic automation.

Sammanfattning

Under de senaste decennierna har industrirobotiken utvecklats från stationära maskiner specialiserade för enstaka typer av uppgifter till flexibla, kollaborativa och mobila system som kan navigera i komplexa fabriksmiljöer. Dagens tillverkningskrav, drivna av bristen på arbetskraft, behovet av snabba omställningar samt framsteg inom AI och sensorteknik, fordrar att robotar utför alltmer sofistikerade, icke-repetitiva uppgifter tillsammans med mänsklig arbetskraft. Att effektivt utforma och genomföra produktionsuppdrag som omfattar samarbete mellan robotar och människor i den typen av dynamiska miljöer medför flera utmaningar: att specificera produktionsprocessen på en användarvänlig abstraktionsnivå som underlättar resursplaneringen, att hantera oväntade fel under exekvering, att skala upp antalet arbetsmoment som ska fördelas och att hantera osäkerheter i tidsåtgång och mänskligt beteende.

Denna avhandling introducerar ett intuitivt sätt att definiera och organisera arbetsmoment samt en uppsättning algoritmiska metoder som hanterar utmaningarna genom hela kedjan. Först föreslår vi en användarvänlig modellering för att definiera produktionsuppdrag för en robot, med automatisk generering av planeringsproblem kompatibla med olika kommersiella planeringsverktyg. För att möjliggöra en snabb återhämtning vid fel under drift presenterar vi task roadmaps, en ny planeringsalgoritm som återanvänder det ursprungliga sökträdet för att accelerera en omplanering när exekveringen avviker. Vi utökar modelleringen till en multi-robot kitting-applikation med alternativa upphämtningsplatser och introducerar en klustringsbaserad algoritm som är beräkningsmässigt hanterbar för uppskalade problem.

Med hänsyn till de inneboende osäkerheterna i människa–robot-samarbete utvecklar vi dessutom ett kollaborativt stokastiskt ramverk för planering av robot-uppgifter som integrerar mänskliga riskpreferenser och modellerar en varierande tidsåtgång för arbetsmoment och förflyttningar. Slutligen behandlar vi ett kollaborativt produktionsscenario med komplexa korsschemaberoenden och föreslår en stokastisk schemaläggningsmetod som genererar optimerade, låsningsfria planer där effektivitet balanseras med mänskligt välbefinnande.

Omfattande simuleringar och experiment baserade på realistiska applikationer visar att våra metoder väsentligen förbättrar effektivitet, robusthet och anpassningsbarhet av robot-planering i dynamiska industriella miljöer, vilket banar väg för en mer hållbar, människocentrerad robotautomatisering.

Acknowledgements

After completing my master student thesis at Uppsala University and ABB, I spent twenty-four interesting years with product development of industrial robots at ABB. One day my wife Sara advised that ABB had announced participation in ARRAY, a new research school at Mälardalen University which was led by Professor Thomas Nolte. This was the spark that ignited an interesting journey of research. My first thanks go to Sara for all support along the way.

I want to thank ABB, my manager Niklas Durinder and Thomas Nolte for supporting and encouraging this research idea from the very start. A special thanks to my main supervisor Professor Alessandro Papadopoulos, whose input in the research process has been invaluable. In addition, I want to thank my supervisors Thomas Nolte, Giacomo Spampinato and Branko Miloradović. I have had a fantastic, complementary, and stable team of supervisors who made this journey possible in a friendly and supportive atmosphere by gradually teaching me how to think as a researcher, discussing ideas and problems, providing guidance, and helping me find ways out in times of trouble.

Thanks to my fellow PhD students from ARRAY and other research projects for the fun discussions and sharing of experiences. Thanks to my colleagues and friends at MDU and ABB. Thanks to the CORE team for hosting me and making MDU my second workplace. In my everyday research life, I have spent a lot of time "embedded" with my ABB colleagues in team Tau. This has been an invaluable social environment, making the sometimes-lonely research work a lot more enjoyable. Thank you all present and recent Tau members for your inclusive team spirit. Thanks to Anders Wall for internal feedback on all papers and to all anonymous reviewers from the research community whose input helped to shape this thesis.

Finally, I want to thank my parents for their unconditional support. My father, who was very enthusiastic about my PhD journey, sadly left us too early but remains a big inspiration.

Anders Lager, Västerås, Fall of 2025

List of Publications

Papers included in this thesis[1]

Paper A: Anders Lager, Alessandro V. Papadopoulos, Giacomo Spampinato and Thomas Nolte. *A Task Modelling Formalism for Industrial Mobile Robot Applications.* In 20th International Conference on Advanced Robotics (ICAR), 2021.

Paper B: Anders Lager, Giacomo Spampinato, Alessandro V. Papadopoulos and Thomas Nolte. *Task Roadmaps - Speeding up Task Replanning.* In Frontiers in Robotics and AI, section Robotic Control Systems, 2022.

Paper C: Anders Lager, Branko Miloradović, Alessandro V. Papadopoulos, Giacomo Spampinato and Thomas Nolte. *A Scalable Heuristic for Mission Planning of Mobile Robot Teams.* In 22nd World Congress of the International Federation of Automatic Control (IFAC), 2023.

Paper D: Anders Lager, Branko Miloradović, Giacomo Spampinato, Thomas Nolte and Alessandro V. Papadopoulos. *Risk Aware Planning of Collaborative Mobile Robot Applications with Uncertain Task Durations.* In 33rd IEEE International Conference on Robot and Human Interactive Communication (RO-MAN), 2024.

Paper E: Anders Lager, Branko Miloradović, Giacomo Spampinato, Thomas Nolte and Alessandro V. Papadopoulos. *Stochastic Scheduling for Human-Robot Collaboration in Dynamic Manufacturing Environments.* Accepted by 34th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN), 2025.

[1]The papers included have been reformatted to comply with the thesis layout.

Other relevant publications[2]

Anders Lager, Alessandro V. Papadopoulos, Giacomo Spampinato and Thomas Nolte. *Towards Reactive Robot Applications in Dynamic Environments.* In 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), 2019.

Anders Lager, Alessandro V. Papadopoulos, Giacomo Spampinato and Thomas Nolte. *IoT and Fog Analytics for Industrial Robot Applications.* In 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), 2020.

[2]Not included in this thesis.

Contents

I

Thesis

Chapter 1

Introduction

1.1 Overview of Thesis

The thesis contains two parts.

Part I presents the research and summarizes the contributions. Chapter 1 gives an introduction motivating and specifying the research challenge. Chapter 2 presents the research context. Chapter 3 breaks down the research challenge into research questions and Chapter 4 explains the applied research process and methods. The research questions are addressed with contributions summarized in Chapter 5 and the thesis is concluded in Chapter 6.

Part II contains the collection of included papers, which provide the details of the contributions and their evaluation. It lists papers A, B, C, D, and E in chronological order.

1.2 Background and Motivation

Since the beginning of the 1950s, robots have been deployed to automate various industrial applications, for example, material handling, machine tending, glueing, painting, laser cutting, arcwelding, and spotwelding. Over the years, industrial robots have played a pivotal role in the automation and acceleration of various production processes thanks to their adaptability, high precision, repeatability, and ability to work 24/7. From the third generation of industrial robots, 1978 - 1999 [1], an extended ability for interaction with users, process equipment, and the environment was established. High-level programming languages increased the flexibility and standardized communication interfaces, for example, fieldbuses and ethernet, enabled interaction with various process equipment. Process control in coordination with highly accurate mo-

tion planning provided means for the successful deployment of sophisticated applications requiring high precision. Environmental deviations, for example, the location of a weld joint line, could be monitored with sensors and compensated for in a reactive way with real-time path adjustments. This historically successful approach to control robots is still widely used in industry today. Despite the flexibility of the third robot generation, these robots are designed to work in a fairly static environment within protective fences. All external events affecting a robot's planned actions are quite predictable and the robot's responses to them are pre-programmed.

In contrast to the third generation robots, today's industrial robots are further enhanced by dramatic increases in computing capacity, Artificial Intelligence (AI), and connectivity to fog and cloud nodes [2]. These robots can process high-volume data in real time, for example, point clouds from depth cameras. The development of AI algorithms has enabled learning of robot behaviors, sophisticated logical reasoning, and an ability to devise long-sighted plans to achieve assigned goals. Of growing importance is the trend of *mass customization* [3], leading to a higher variability in robot tasks and smaller batch sizes, which requires new ways to leverage the flexibility of robots without compromising the output quality of assigned tasks. A robot's operating environment can be open and more accessible for humans, reducing the footprint and enabling new ways of human-robot collaboration. The growth of collaborative applications has been supported by the development of safety standards [4], providing guidance for risk assessments in the robot installation phase. With the trend of mass customization, the skills and adaptivity of human workers are an important complement to handle unexpected problems related to the potentially unique conditions that may appear in each processing step. Human tasks may involve planning, supervision, problem-solving, or assembly. They may cover up for robot limitations in skills or capacity, for example, non-automated tasks or hard-to-automate tasks requiring a high level of creativity, dexterity, or manipulation skills. The different skills of robots and humans may be combined to solve complex tasks in a flexible and efficient way [5, 6]. However, collaborative robot applications expose a more dynamic environment that may interfere with robot actions in unexpected ways.

A wheeled platform equipped with a manipulator arm, known as a *mobile manipulator*, extends the operational range and enables the handling of more flexible and versatile tasks. This has opened up new types of industrial applications, for example, service tasks in medical labs, see Figure 1.1, or logistic operations in warehouses. Mobile robots may also form fleets to perform multi-robot missions. Besides increasing the performance of missions, a robot fleet can increase the variety of tasks that can be handled through cooperation

Figure 1.1: A mobile manipulator performing laboratory tasks in a hospital environment.

or heterogeneous (diverse) robot abilities. Additionally, the redundancy of a fleet may be used to safeguard a continuous ability to operate when robot errors occur. For the sake of simplicity, a mobile manipulator is hereafter referred to as a *robot* or a *mobile robot*, although these terms represent a wider scope.

In an industrial context, a mobile robot's operating environment is expected to be fairly structured, but the exact conditions, for example, locations of equipment or temporary obstacles, will vary in a dynamic working place. A robot is further expected to encounter unexpected actions and events initiated or caused by co-located humans, robots, or other actors. Such environmental uncertainties, in combination with shorter product cycles, increase the risk of errors and may cause a significant variability in task durations. While human participation in collaborative robot applications introduces uncertainties, for instance increased variation in task durations, it also provides advanced skills and problem-solving capabilities that strengthen system robustness. For

convenience, the term *agent* can represent a robot as well as a human in this thesis.

Complex routing problems, dependencies between tasks, and coordinated actions among robots and humans increase the need for long-sighted planning to maintain efficient operation. To devise efficient and realistic plans, uncertainties need to be taken into account.

1.3 Research Challenge

A transformation of technologies in industry is needed to handle increasingly complex robot applications while coping with the uncertainties of dynamic environments. The research challenge addressed in this thesis is:

How can we design an end-to-end, domain-expert-friendly planning framework that enables industrial users to intuitively specify and organize mobile-robot tasks, while generating efficient, scalable, dynamically adaptable, and risk-aware plans for both single- and multi-robot systems operating in uncertain, human-collaborative environments?

Chapter 2

Research Context

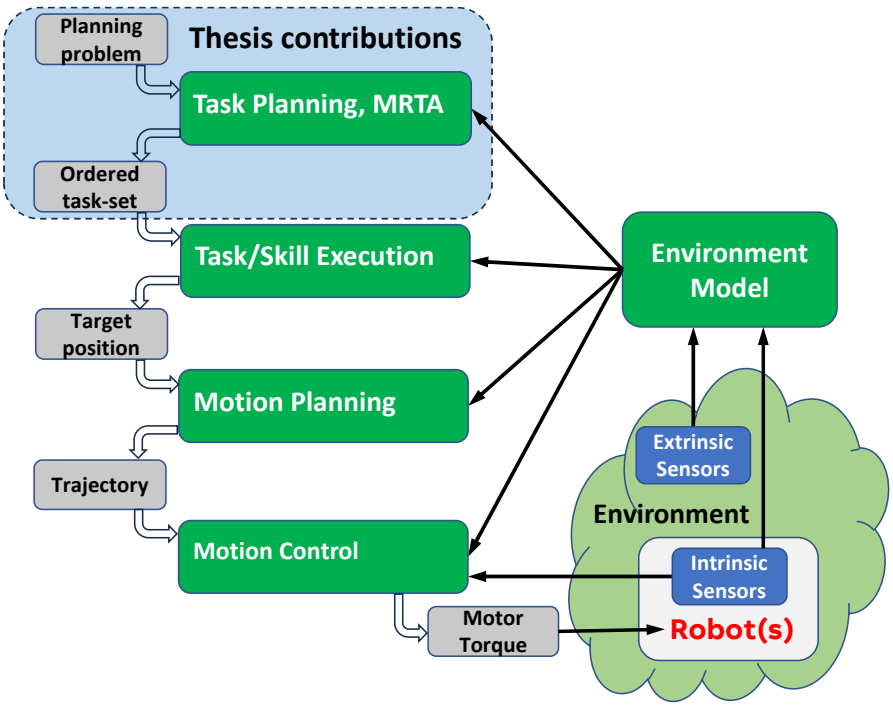

Figure 2.1: Scope of the thesis in the view of a robot system architecture.

Complex robotic production scenarios require/benefit from intelligent planning, execution and control with autonomous decision making. In Figure 2.1, a hierarchical architecture for a robot system is exemplified, having a top-down information flow with feedback loops from extrinsic

sensors observing the environment and intrinsic sensors monitoring the actuation of robots. At the top level a planning problem, given by a desired production scenario or mission, is sent to a task planning component that computes sequences of tasks, their allocation to robots and their mutual dependencies. At the second level, a task execution component processes tasks in the given order and generates target positions to guide robot movements. At the third level, a motion planner generates trajectories to reach target positions while considering obstacles in the environment. At the fourth level, a motion planner regulates motor torques to move the robots accordingly.

The thesis contributions belong to the top level in this architecture, which is related to Task Planning and Multi Robot Task Allocation (MRTA). Additionally, they cover related aspects of humans-in-the-loop. In this chapter these and other related topics and research areas are presented, starting with Task Representations in Section 2.1, Task Planning in Section 2.2, Multi Robot Task Allocation in Section 2.3, Collaborative Robot Applications in Section 2.4, Uncertainties in Robot Applications in Section 2.5, Reactive Task Planning in Section 2.6 and Proactive Task Planning in Section 2.7.

2.1 Task Representations

For humans, it is convenient to reason about work descriptions for a robot at an abstract level. They may specify *goals* to be reached, for example, "A circuit board shall be on the table", while leaving out the details of how to accomplish them. Alternatively, they may specify *tasks* to be performed, for example, "Assemble a circuit board", which define what is to be done while leaving out the details of how to accomplish it. For robots, tasks must be specified at a more detailed, machine-interpretable level to enable the control of robot movements and the use of required equipment, such as a robot-held tool. In this thesis, we refer to a *task* as an abstract action relevant to industrial or service applications, for example pick, place, fetch, deliver, inspect, screw, or glue, etc. A task may be performed by a single agent or a group of agents.

When organizing the planning or execution of tasks, a model is typically created with a *modeling formalism* that provides relevant elements, for example tasks, objects, agents and goals. Such a model may indicate relations between different elements, for example, "robot1" is holding "gripper", or constraints, such as, "task A" must be completed before the start of "task B". Several modeling formalisms exist for different purposes [7], for example, *programming*, *policies* and *planning*. Another important aspect to consider is the intended users of a modeling formalism. More specifically, what are the users'

needs and areas of knowledge? For industrial applications, a user is often a *domain expert* with in-depth knowledge about the process and the tasks the robots shall perform. However, domain experts are not necessarily experts in robot programming and other representations for planning and control of robot applications.

2.1.1 Programming

With standard programming languages such as C++ or Python, or with specialized robot programming languages such as ABB's RAPID [8], a desired control behavior can be manually defined using structured programming. This typically involves sequences of statements and constructs to control program flow, including *if–else* and *do–while*. In block-based programming [9], a library of visual blocks representing useful constructs or routines, for example a pick task, can be organized into a program structure. The building blocks hide a substantial part of the underlying programming details to avoid exposing the full complexity, for example, the detailed control of the gripper in the pick task. Block-based programming is intended to simplify and reduce the threshold for programming, which may be advantageous if an operator needs to swiftly adapt the program to handle a new product. Closely related to blocks representing tasks is the concept of a *skill* [10], which is a reusable implementation of a commonly used robot task. The skill encapsulates the detailed motor and sensing operations needed to run the skill. An instantiation of a skill in a robot program will typically require updating a set of skill parameters.

2.1.2 Policies

Some modeling approaches simplify the specification of complex control behaviors, for example, State Machines [11], PetriNets [12], and Behavior Trees [13]. In general, these approaches provide a reactive control *policy*, indicating which action shall be taken for a given observed state of the environment. Typically, a new action is triggered by a change of state. Actions may be performed by programmed routines, thereby combining modeling formalisms for policy and programming. In another common approach, a policy can be generated from a Linear Temporal Logic (LTL) specification indicating desired robot behaviors over time with logical statements [14]. Policies may also be learned to implement low level control behaviors for complex tasks/skills, for example, picking up a garment with a multi-fingered, robot-held gripper, where the policy output corresponds to actuated motor torques. This can be accomplished with Reinforcement Learning, where a

policy is learned from repeated experiments in the real world or a simulation environment [15]. The resulting policy-providing model, which can be considered a black box without interpretable logic, may be combined with higher-level modeling formalisms for policies or planning.

2.1.3 Planning

The purpose focused in this thesis is *automated planning*, also referred to as AI Planning, where a plan is derived to reach a goal by selecting and organizing tasks. Such a plan is typically more long-sighted than the look-ahead of a policy. AI planning and related modeling formalisms, for example PDDL, are covered in the next section. To integrate planning with plan execution, modeling formalisms for programming, policies and automated planning may be combined. In [16], a skill framework was combined with PDDL to enable automated planning by converting skills to PDDL actions suitable for planning.

2.2 Task Planning

2.2.1 AI planning

Planning is a problem where tasks[1] are selected and organized by anticipating their outcomes to reach a goal. For humans, explicit planning requires some effort and can be avoided if the goal is simple or can be reached with a well-known procedure. Planning can also be avoided if the goal can be efficiently approached in an exploratory manner. Planning may pay off if the goal is complex, requires close cooperation, or if a bad selection of tasks entails high risk or cost.

AI planning [17] is a computational study of the planning process. It provides computational methods, modeling tools, and algorithms for selecting and organizing tasks[1] into a plan. A plan, if executed from the current state, which represents the observed situation of the robots and their environment, will achieve the specified goal(s). A *feasible* (valid) goal reaching plan is not necessarily the most efficient plan. To rank different plans, a plan cost can be estimated, for example, the time required to perform a plan or the total energy consumption required. A plan is *optimal* if it minimizes the cost.

If the planning approach is *domain specific*, representations in the modeling formalism are adapted for a specific type of problem, for example motion planning. On the other hand, *domain independent* planning provides representations and methods of more general nature that can be applied to planning

[1] Often referred to as *actions*.

problems of many different types, for example, planning of robots, satellites, or patient appointments in healthcare. While domain dependent representations limit the problem types that can be solved, they are typically more efficient and therefore an important complement. The study of AI Planning is mostly concerned with domain independent planning.

Closely related to planning is *scheduling* which sometimes represents the same thing. In this thesis, the term scheduling is used about organizing tasks that are selected beforehand, for example by a domain expert, whereas planning also identifies these tasks. Like planning, scheduling can involve determining the order of tasks and choosing which agents to assign them to.

2.2.2 Precedence constraints

Precedence Constraints (PCs) represent ordering restrictions between tasks that must be satisfied by a plan. A PC is a common relation between tasks, for example, in a sequential manufacturing process. There are four main types of PCs [18] that may be required to hold for a pair of tasks, A and B:

1. **Start-after-completion**. A cannot start until B has completed. This is the most common type of PC in the planning literature. Besides specifying the order of A and B, it prevents them from concurrent execution.

2. **Start-after-start**. A cannot start until B has started. Neither prevents nor requires concurrent execution.

3. **Complete-after-start**. A cannot complete until B has started. Neither prevents nor requires concurrent execution.

4. **Complete-after-completion**. A cannot complete until B has completed. Neither prevents nor requires concurrent execution.

PCs may be specified with a modeling formalism. In paper E, all four types are present, while papers A to D consider intra-schedule start-after-completion PCs.

2.2.3 Defining a planning problem

A computational approach to planning typically requires a problem formulation that describes the problem in logical terms. This typically includes constraints, for example PCs, that must be satisfied by a feasible solution. Given the problem formulation, an algorithm may search for a good feasible solution.

A planning problem may be represented in a purely mathematical form as an optimization problem. Optimization problems include a set of decision

variables representing a parameterized plan. An algorithm is used to identify the decision variables so that an objective function, describing the cost of a plan, is minimized while a number of constraints, describing requirements for a plan to be feasible, are satisfied. One type of optimization problem is a Mixed Integer Linear Program (MILP), characterized by a mix of discrete and continuous decision variables with a linear objective function and linear constraints. A mathematical representation is indispensable for a researcher and included in all papers of this thesis to precisely define targeted planning problems in a compact format. However, mathematical expressions are rarely a convenient way for domain experts to formulate or communicate mobile robot planning problems, as illustrated in Section 2.1.

Problem Domain Definition Language (PDDL) [19] provides a modeling formalism for planning problems that is developed within the AI Planning research community. PDDL originates from classical planning [20], where the state of the environment is represented with objects and binary *facts* indicating the existence (or non-existence) of different relations between objects. To alter the environment, an action involving a set of objects can be applied if the action's preconditions on a set of binary facts are fulfilled. The *effect* of applying an action is the creation of a new set of facts and the removal of another set of facts. A planning problem is defined by an initial state and a desired goal state. A plan is a sequence of actions that gradually alter the environment from the initial state to the goal state. PDDL has evolved over time to cover different aspects of planning problems, for example temporal planning [21, 22] where actions have a duration and facts can be numerical, and probabilistic effects [23].

In industry, tasks to be performed are typically well organized by domain experts. Therefore, it makes sense to explicitly represent which tasks need to be accomplished rather than specifying a goal state. A goal state specification leaves room for a planner to reach the goal in "clever ways" with unexpected actions that may be undesirable. For a domain expert, it can be convenient to provide an overall work description at a suitable abstraction level where the detailed planning is handled by automated planning, thereby combining the skills of a domain expert with the efficient reasoning capacity of planner algorithms. One approach to the modeling task of task planning problems, Hierarchical task Networks (HTN) [24] takes this approach. In HTN, the goal is to perform an *abstract* task, which is a task description on a higher abstraction level, for example, a mission. A robot can only be assigned *primitive* tasks, which are task descriptions on the lowest abstraction level. An abstract task is associated with alternative *methods* that can be used to decompose the task into a partially or totally ordered set of sub-tasks. A partially ordered set of tasks

does not impose mutual task orderings unless required. The decomposition is made recursively until there are only primitive tasks. A final ordering may be planned to optimize the efficiency or it may be resolved during runtime. A decomposition method has preconditions on the state that must be fulfilled to be applied. Similar to PDDL actions, tasks have preconditions on facts and effects that modify facts.

Both PDDL and HTN are powerful modeling approaches that simplify the definition of planning problems compared to a purely mathematical approach. However, there is a significant threshold for a domain expert to define a planning problem with PDDL or HTN. Especially for PDDL, it is hard to get an intuition of the expected robot behavior from a planning problem.

One graph-based approach of simplifying a robot work description for a task planning problem is Expression Tree, which recently was proposed and investigated for a single-robot Traveling Salesperson Problem problem [25] followed by a multi-robot Vehicle Routing Problem (VRP) [26]. This modeling approach has AND nodes, OR nodes, and sequence nodes with functional similarities to the earlier modeling approach, Robot Task Scheduling Graph, proposed in paper A and extended in paper C to model a VRP problem.

2.2.4 Motion planning and control

Motion planning is used to generate the path or trajectory of a robot when moving between different locations in the environments, for example, within a task or between different tasks. A robot trajectory should be safe and avoid unintended collisions. An optimized trajectory could be fast or have a limited energy consumption. *Motion control* deals with the following of planned trajectories using control loops that actuate motors while considering sensor feedback indicating actual trajectories. This thesis does not contribute to motion planning or motion control, but these topics are intertwined with task planning. A task planner affects which movements shall be planned by the motion planner as indicated in Figure 2.1. But task planning may also be affected by motion planning. For example, estimated paths for moving between task locations may affect the selection and sequencing of tasks. In the research field Task and Motion Planning (TAMP), task planning and motion planning are combined to consider both problems in parallel [27], but typically at the price of a highly increased problem complexity. TAMP may enable more flexible plans, for example, for problems with highly constrained movements where task selection and ordering matters for the generation of feasible motion plans.

With a mobile manipulator, motion planning needs to be combined for the mobile base and the arm to perform planned tasks in the operational environ-

ment [28]. To follow a motion plan towards a planned location, the robot needs to keep track of its own location. A traditional robot with a stationary base uses intrinsic sensors, for example, resolvers or encoders, to locate itself accurately in the workspace. With a wheeled mobile base, a robot additionally needs extrinsic sensors, such as laser scanners or depth cameras, to locate itself with accuracy. Localization can be achieved by matching sensed features in the environment with a map of the workspace. Initially, the map may be created by moving around the robot to explore the workspace with the extrinsic sensors. Thereafter, the map can be maintained during operation with Simultaneous Localization And Mapping (SLAM) [29]. For example, if a new stationary object has appeared in the workspace, the robot may still locate itself from visible mapped features and the new object will over time become a part of the map if it remains in the same position long enough.

2.3 Multi Robot Task Allocation

Planning or scheduling multi-agent missions are referred as Multi Robot Task Allocation (MRTA) problems. MRTA problems can be categorized with four different problem dimensions [30, 31] to indicate their complexity:

1. Task concurrency. *Single-task robots* (ST) must handle tasks sequentially, while *multi-task robots* (MR) are able to perform multiple tasks in parallel.

2. Task type. *Single-robot tasks* (SR) are handled by a single agent, while *multi-robot tasks* (MR) require simultaneous cooperation with multiple agents.

3. Assignment type. With *instant allocation* (IA) a task is not planned until agents are available, while *time extended assignment* (TA) generates a more long-sighted plan that allocates multiple tasks for agents.

4. Interrelatedness between tasks. In the simplest case, there are *no dependencies* (ND) between tasks. With *intra-schedule dependencies* (ID) tasks performed by the same agent have dependencies, for example precedence constraints. With *cross-schedule dependencies* (XD) there are dependencies between tasks allocated to different agents. With *complex dependencies* a task can be recursively decomposed into a set of dependent *primitive* tasks which cannot be further decomposed. For example, "Clean the kitchen" can be decomposed into "Vacuum the floor", "Mop the floor", "Clean the sink", "Clean the stove", etc.

> "Vacuum the floor" must be ready before starting "Mop the floor" and the primitive cleaning tasks may be allocated to different agents. There may exist alternative decompositions for a task, representing different ways to perform them, for example with another set of tasks.

The MRTA category for a planning problem can be specified by listing the four characteristics. For example, in paper C an MRTA problem of category ST-SR-TA-ID is investigated while in paper E a more complex MRTA problem of category ST-MR-TA-XD is targeted. The other papers do not cover MRTA problems, as they either plan a single robot (paper A and B) or use a static allocation of tasks to agents (paper D).

The main variants of solving MRTA problems are *centralized* and *decentralized* approaches [32]. In centralized approaches, there is one point of control for allocating tasks to robots while decentralized approaches organize the allocation of tasks in a distributed manner, for example with auction-based approaches. While decentralized approaches often are fast and scalable, centralized approaches may generate more efficient or even optimal plans. In this thesis, centralized approaches are focused.

2.4 Collaborative Robot Applications

In collaborative robot applications, tasks are performed by both robots and humans in a shared or separated workspace. Different cooperation modes are possible [33]. While *Cooperating*, robots and humans share the same workspace and goals but are occupied with different tasks. Additionally, they may concurrently work on the same task in a *Collaborative* mode. If on the other hand they do not interact, this can be referred as *Coexistence* [34]. If *Synchronized*, a robot and a human perform tasks in a common part of the workspace, but in a sequence not overlapping in time, for example a human handing over a work piece for later processing by a robot. A common theme for most works on planning and execution of collaborative robot applications is *uncertainties* related to the interaction between robots and humans and how they can be managed.

2.5 Uncertainties in Robot Applications

The industrial environment where a robot operates is assumed to be *semi-structured* and *dynamic*. A semi-structured environment means the environment mainly contains objects that are known or can be identified, for example a screw or a circuit board. However, observations may also contain features

that cannot be categorized, for example the tools of a service technician performing temporary work. A dynamic environment means unexpected actions from humans or other actors may take place that interfere with the actions of the robot. Many of the uncertainties in the operational environment cause routing and task durations of agents to become uncertain, which affects the efficiency of a mission plan. In this section, we identify a relevant subset of uncertainties discussed in the literature.

2.5.1 General uncertainties

From the time of the very first industrial robots, the positioning of parts to be manipulated, often in repetitive cycles, has had a variation from the expected location. Traditionally, such variations have been minimized with fixtures and accurate sensor measurements. However, the need for flexibility to handle such deviations increases, for example to handle the picking of objects from an unsorted pile (bin picking) with partly observable objects. Many works have focused on part position adaptivity [35]. Adaptivity may not be enough to avoid errors from occurring requiring automatic recovery actions, for example retries, or human assistance to restart operations [36]. Parts may also be missing or hidden. The detection of incorrectly processed parts with inspection tasks may require recovery actions that affect the through-put of a manufacturing process. The arrival or cancellation of tasks or missions may have a stochastic behavior requiring re-planning of ongoing activities. Planned movements or routes of robots may be affected by temporary obstacles. Multiple agents may cause temporary congestion at critical passages [35]. Sometimes, unexpected environmental conditions may require explicit exploration of an area, for example to update the map or to find a missing object [37]. Tasks may fail due to function failures of robots, tools, and other equipment [38]. With the trend of mass customization, the required robot capabilities to perform a task may depend on uncertain characteristics of objects to be manipulated, for example size, shape, weight, softness etc. Heterogeneous robots with different capabilities may be used to cover a larger variability of tasks, but uncertainties in required capabilities poses risk of allocating the wrong agents [39]. Some uncertainties in resource consumption [40], for example a robot's battery energy consumption for executing a task in a dynamic environment, needs to be managed in real-world scenarios.

2.5.2 Uncertainties in collaborative applications

Collaborative applications add additional uncertainties related to the interaction between humans and robots. Regardless of the human-robot cooperation

mode, the proximity of humans may require stopping, speed reduction or re-planning of robots to avoid a collision or limit its effects. The availability of human agents and their task execution capacity are more uncertain due to human factors such as attention and fatigue. Human selected task sequences may vary from time to time and between different humans [35]. Human capabilities or skills may increase significantly over time from a growing experience [41].

2.6 Reactive Task Planning

While executing a plan a mobile robot needs to be reactive to unexpected conditions or events, which requires a degree of autonomous behavior. The architecture example in Figure 2.1 has four hierarchical levels in a top-down order given by task planning, task execution, motion planning and motion control. Re-planning may be initiated at one of these levels without involving upper levels, for example to adjust a trajectory, modify a task or re-plan a task sequence [42]. However, re-planning at a lower level, for example adjusting a trajectory, may affect a higher-level plan, for example by altering the plan duration. Re-planning is typically faster and less complex at a lower level, requiring less computational effort and less sophisticated corrective actions. However, re-planning on a higher level may solve more complex planning problems and increase the operational efficiency. In the following section, re-planning at the task planning level is discussed.

2.6.1 Reactive task re-planning approaches

Reactive task re-planning approaches, for example paper B, is mainly focused on detecting and resolving planning problems on the fly. Typically, knowledge of uncertainties are not considered in the planning phase.

At some point along the execution of a plan, the observed state may not match the expected state well enough to be able to continue with the current plan, and a re-planning may be required before the execution can continue. Another purpose of re-planning can be to optimize the plan for new state conditions in an opportunistic way, where the old plan is still valid [43, 44].

The robot's re-planning time needs to be limited since it has a direct impact on the productivity, while the generation of an initial plan may be less critical. The most straight forward approach to re-planning is to re-generate the plan from scratch, where the initial conditions have been updated with the observed state of the environment, including unexpected conditions and the effects of the robot's performed work [45, 46]. However, the complexity of a planning problem is typically NP-hard, causing an exponential growth of planning time

with the problem size. In a runtime scenario with frequent re-plannings from scratch, the planning time may become a bottleneck for efficient operation.

To speed up re-planning, some alternative strategies have been suggested where [47] used a rule-based rearrangement of operations to repair the plan and [48] proposed generating an initial plan with partially ordered tasks, combined with a runtime algorithm generating alternative completely ordered plans and selecting the one with the highest probability for success. For multi-agent missions, the concept of an adaptable task to manage different situational conditions in uncertain environments was proposed by [49], using trigger functions for a proper runtime selection of task adaption.

To speed up both planning and re-planning, heuristic algorithms can be used to search for acceptable but suboptimal solutions, which is an approach used in paper C. A heuristic algorithm typically explores a limited part of the solution space guided by a promising strategy, a heuristics, to find feasible and sometimes near-optimal solutions in a limited planning time. Meta-heuristic algorithms, used in paper E, are more general suboptimal search strategies that can be applied to a wide range of planning problems. They are designed to explore the solution space effectively with elements of randomness, for example Genetic Algorithms [50], Simulated Annealing [51] or Particle Swarm Optimization [52]. The optimality and convergence of metaheuristic algorithm typically depends on the tuning of some hyperparameters affecting the search strategy.

2.7 Proactive Task Planning

Proactive task planning approaches consider knowledge of uncertainties in the planning phase to provide more efficient plans. Proactive task planning does not replace reactive task planning, which may coexist in a robot system architecture.

2.7.1 Related approaches

When performing an industrial task, it is supposed to change the state of the environment in a desired way. For example, after a robot picks up an object, it is expected to hold the object. However, in a dynamic environment tasks may fail and the object may instead fall to the ground, which is an example of uncertain task effects. Planning problems with uncertain effects can be modeled with Markov Decision Processes (MDP) [53]. A probability is estimated for every state s_{t+1} that may result from an action a_t taken in an initial state s_t at time t. Instead of generating a plan, a policy $\Pi(s_t)$ is computed, indicating

what is the next action to take given the observed state. The policy is computed to maximize an accumulation of rewards from future actions. Due to the computational burden of evaluating alternative effects at each decision point, the method does not scale well for long-sighted planning problems. Observations may also be uncertain, for example an inaccurate observation may indicate that the robot is holding the object, although it is on the floor. Partially Observable Markov Decision Processes (POMDP) [54] is an extension of MDP where the uncertainty of the observed state is modeled. In an industrial scenario, these two types of uncertainties are prevalent and may cause errors while executing planned tasks.

Industrial tasks are typically well defined and cannot be canceled if they fail. Instead of modeling uncertain task effects or state observations, it can be reasonable to model the uncertainty of *task durations* to account for a dynamic environment which may cause re-planning of robot paths, retries of tasks or assistance from a human supervisor to solve problems. Temporal uncertainties are addressed in Probabilistic Simple Temporal Networks (PSTN) [55], which is used to model scheduling problems with temporal constraints, for example a deadline for an activity to be finished. Start and end times of tasks are modeled as random variables with probability distributions. To solve a PSTN is to compute a schedule for all tasks where the risk for violating temporal constraints is minimized or bounded. Modeling task durations as random variables with Gaussian probability distributions for task planning problems was suggested by [56], where a set of approximate stochastic operators was used to add, multiply, maximize, etc. random variables in a computationally efficient way. In another approach, triangular fuzzy sets were used to model uncertain durations of human tasks and robot tasks in a collaborative robot application [57].

The uncertainty of resource consumption, for example battery consumption, was recently addressed in a multi-agent scenario featuring critical tasks and less critical tasks associated with both optimistic and pessimistic worse case costs. A Mixed Criticality approach from the real-time community was proposed, yielding proactive optimized plans providing guarantees to handle all critical tasks in worst case scenarios [58].

2.7.2 Random variables

In this section, some definitions of random variables from probability theory are introduced in an intuitive way, omitting details of the mathematical definitions provided in paper D and E. In these papers, random variables are used to model and represent uncertainty of task and routing durations. Moreover, operands for random variables, sum and $\max$, are combined to compute prob-

abilistic makespans of planned missions incorporating sequential as well as concurrent tasks.

2.7.2.1 Probability distribution

A random variable X may for example be modeled to represent the time required to move a mobile robot between two specific locations in a factory. This movement time is uncertain for various reasons discussed in Section 2.5. The *outcome* of X is a random value that can take any value matching possible outcomes of the movement time. However, the probability for an outcome within some time intervals is higher than others. This *probability distribution* of outcomes can be modeled with a Probability Density Function (PDF), $f_X(t) \geq 0$, exemplified in the upper diagram of Figure 2.2, where the horizontal axis represents (outcomes of) time. The total area under the PDF curve is 1 and the probability for an outcome of time less than t is represented by the area below the PDF curve up to t. This probability can be computed by integrating the PDF:

$$F_X(t) = \mathbb{P}[X \leq t] = \int_{-\infty}^{t} f_X(\omega)\,\mathrm{d}\omega \tag{2.1}$$

The function $F_X(t)$ is referred as the Cumulative Density Function (CDF) of X and is visualized in the lower diagram of Figure 2.2, where CDF values, for example 0.4, match the left-hand side area below the PDF curve.

For Definition 2.1, we assume X is a *continuous* random variable. To model any distribution, for example from data collected by repeated observations of actual or simulated outcomes, it is often convenient to let X be modeled as a *discrete* random variable where the outcome is a set of discrete values rather than a continuous range. For example, the outcomes may be modeled with a set of equidistant discrete values, for example $\Omega = \{0.0, 1.0, \ldots, N\}$ where the values represent equidistant time intervals $\{[-0.5, 0.5), [0.5, 1.5), \ldots\ [N - 0.5, N + 0.5)\}$. The probability distribution is modeled with a discrete PDF, sometimes referred as a probability mass function. The value of $f_X(t)$ represents the probability for an outcome within the time interval containing t. The CDF becomes a sum of $f_X(t)$ values:

$$F_X(t) = \mathbb{P}[X \leq t] = \sum_{\{\omega \in \Omega | \omega \leq t\}} f_X(w) \tag{2.2}$$

If a higher modeling resolution is desired, the interval length can be reduced to a proper value while increasing the size of Ω to represent the same range of time.

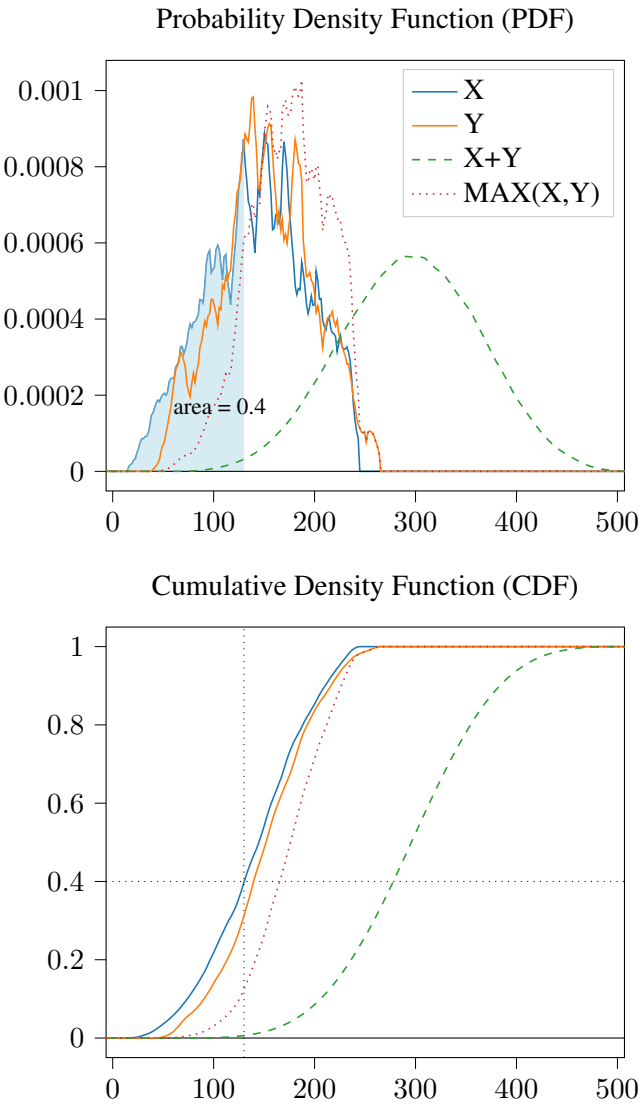

Figure 2.2: Distributions of random variables.

2.7.2.2 Percentile

p_k is the k-th percentile of a probabilistic distribution $f_X(t)$ and is defined as:

$$p_k = \inf\{t : F_X(t) \geq k\}, \quad 0 < k < 1.$$

In Figure 2.2, $p_{40} \simeq 130$ is indicated in the CDF and PDF diagrams for X. p_{50} is the median value, which is the center point of a distribution.

2.7.2.3 Independence

Two random variables are independent if there is no correlation between their outcomes. This relation can be illustrated with an example:

Assume a robot shall move between two locations with duration X and thereafter pick an object with duration Y. If X and Y are independent random variables, the observation of X must not alter the probability distribution of Y. For example, a fast movement should not increase the probability for a fast pick. Further assume Z represents the total duration of moving and picking, that is $Z = X + Y$. This is an explicit correlation between Z and X (or Z and Y) indicating their dependence, and a fast movement increases the probability for a fast total time.

2.7.2.4 Stochastic dominance

Stochastic dominance is a relation where one random variable is larger than (dominates) another. Y dominates X if $\forall t$, $F_Y(t) \leq F_X(t)$ which is denoted $Y \geq_{st} X$. This relation is exemplified in Figure 2.2, where the CDF curve of Y never is above X. Stochastic dominance does not occur if the CDF curves intersect.

2.7.2.5 Sum

The sum of two independent random variables, $X + Y$, also referred as convolution, can be used to compute the total duration of two sequential tasks. The sum becomes a distribution exemplified by the PDF and CDF diagrams in Figure 2.2. The discrete version of the sum is defined by:

$$f_Z(t) = \mathbb{P}[Z = t] = \sum_{\omega \in \Omega} f_X(\omega) f_Y(t - \omega) \tag{2.3}$$

2.7.2.6 Maximum

The maximum of two independent random variables, $\max(X, Y)$, can be used to compute the total duration of two concurrent tasks, X and Y. The maximum becomes a distribution exemplified by the PDF and CDF diagrams in Figure 2.2. It has a CDF defined by:

$$F_Z(t) = F_X(t)F_Y(t) \tag{2.4}$$

Chapter 3

Research Questions

The general research challenge,

How can we design an end-to-end, domain-expert-friendly planning framework that enables industrial users to intuitively specify and organize mobile-robot tasks, while generating efficient, scalable, dynamically adaptable, and risk-aware plans for both single- and multi-robot systems operating in uncertain, human-collaborative environments?

is addressed with four specific research questions (RQs). These are listed in the following, including a brief description of how they are tackled by the thesis:

RQ 1: **How can we design a graph-based modeling formalism that allows domain experts to specify and organize industrial mobile-robot tasks intuitively, and automatically translate models into both MILP and PDDL planning representations?**

The thesis introduces the *Robot Task Scheduling Graph* (*RTSG*) formalism: a compact, visual, graph-based language for domain experts to specify alternative task sequences without needing deep programming knowledge. RTSG models consist of start/goal nodes, AND- and OR-forks, and lock/join constructs to capture sequencing and synchronization. A conversion pipeline then automatically translates any RTSG model into (1) a Mixed Integer Linear Program and (2) a Planning Domain Definition Language specification. A benchmark on a mobile kitting application shows that the two resulting formulations are equivalent in solution quality for representative solvers, and that RTSG yields intuitive models.

Furthermore, an experiment correlating path-length cost models with simulated makespan validates that RTSG's cost abstraction faithfully

guides plan efficiency.

RQ 2: **What algorithms and data structures enable low-latency initial planning and incremental replanning of mobile-robot tasks when the environment changes?**
To answer this, the thesis proposes *Task Roadmaps* (*TRM*), inspired by probabilistic roadmaps in motion planning. A Branch-and-Bound (B&B) algorithm first explores an RTSG-model-derived search space offline, building a reuseable "roadmap" of partial plans. When replanning is needed (e.g., a path is blocked or a task fails), the B&B-TRM query phase identifies the current progress node and reuses existing subtrees instead of expanding. In simulation with a mobile manipulator kitting scenario, B&B-TRM replanning is orders of magnitude faster than solving from scratch with either a MILP solver or a PDDL planner.

RQ 3: **How can we efficiently plan a fleet of robots for a multi-agent mission, ensuring high-quality solutions within bounded computation time?**
The thesis develops a cluster-and-balance heuristic:

- Supervised clustering of tasks into k groups (one per robot) using a Variable Neighborhood Search-based adaptation of k-medoids, which respects separation/precedence constraints.
- Route planning within each cluster via a TSP solver for each robot.
- Conflict resolution if multiple clusters pick the same robot (recompute the cheapest cluster).
- Alternative-task pruning to remove redundant branches.
- Load balancing by iteratively moving tasks (with highest gain/loss ratio) between robot sequences to reduce makespan variance.

On problems up to 200 tasks and 5 robots, this heuristic finds plans within a few seconds that are near-optimal compared to Gurobi's MILP solver (which times out or runs for minutes), with planning-time-to-makespan ratios under 10%. The heuristic's makespan deviation from optimal shrinks as task counts grow, while the MILP solver becomes infeasible for online use.

RQ 4: **How can we incorporate stochastic task-duration models and human risk preferences into a planning framework to generate, evaluate, and select collaborative robot-human plans that balances**

makespan efficiency and uncertainty?
Two layered contributions tackle this.

First, in Paper D, we focus on the plan generation under uncertainty. We propose an extension of the RTSG model to include human tasks and &JS (join-sync) nodes that block robot actions until humans complete their work. The paper introduces stochastic durations of tasks, with robot and human task times modeled as independent random variables with generic, refinable distributions. The paper then proposes a solver based on a Branch-and-Bound search over stochastic RTSG that generates all non-dominated plans (in terms of first-order stochastic dominance) by safely pruning plans whose CDFs are dominated by others.

Second, Paper E extends the above to multiple robots and humans, handling cross-schedule dependencies. The paper introduces a GRASP heuristic, a deadlock detection & repair algorithm, and merge/prune operators to bound the complexity of random-variable combinations, and it provides analytical bounds (upper/lower) on makespan distributions under dependence assumptions, ensuring robust schedule executability with quantified risk.

Together, these ensure the planning framework not only produces efficient nominal plans but also equips domain experts with risk-aware choices grounded in probabilistic performance guarantees.

Chapter 4

Research Process and Methods

In this thesis a constructive research approach is used, which is a solution-oriented way to tackle the research challenge. In a *research process*, solutions are devised to improve some *aspects*, for example simplicity, performance or robustness of *artifacts*, for example a task planning methodology or algorithm. The improvements are verified with suitable *research methods*.

4.1 Research Process

Research questions are defined to guide the research process, see figure 4.1, which is a repeated cycle of sequential steps:

1. **Develop a research question:** Devise an initial research question. Find and analyze state-of-the-art with scientific literature studies. Are there interesting aspects of the research question that have not been properly addressed before? Are there promising solution ideas that have not been properly explored before? Refine the research question to target a promising solution idea.

2. **Develop a solution:** Devise and develop a solution to the research question. A solution must include novel elements that improve aspects of alternative approaches. This is often the most time-consuming step, requiring iterative development and testing of solution components, the development of a realistic validation framework and the integration of alternative approaches to be compared.

3. **Validate the solution:** Check correctness of claimed solution aspects. Compare with relevant alternative approaches.

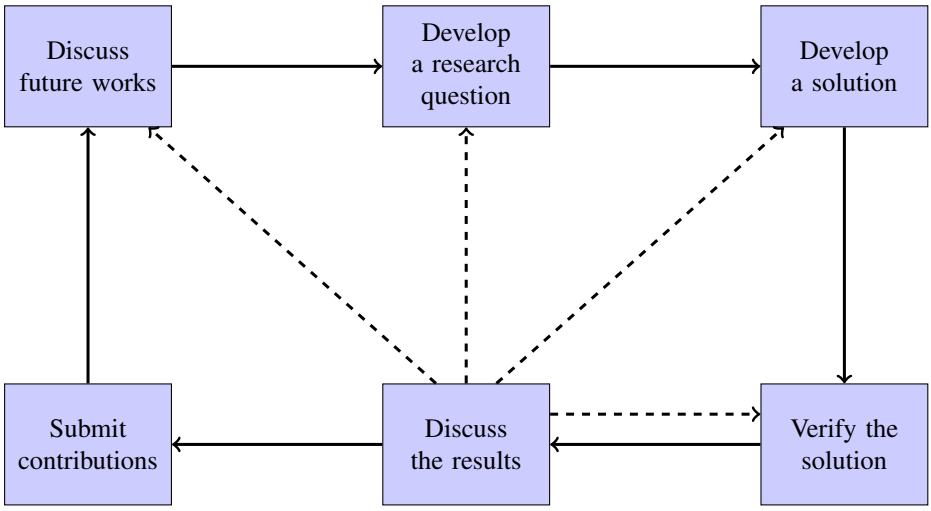

Figure 4.1: The main steps of the research process.

4. **Discuss the results:** Did our devised solution address the research question effectively? If the answer is no, it may be possible to improve or change the solution. On other occasions, the research question needs to be refined or changed. Sometimes, a developed solution may be more promising for another research question. Are the results valid? If not, the solution may have to be validated in a new way. The next step depends on the discussion and will lead to a new iteration in the research process. Eventually, the next step is to submit contributions.

5. **Submit contributions:** Document contributions and findings in the context of related works. Papers are submitted to conferences or journals to undergo a peer review process. Ultimately, submitted papers become published and extend the knowledge of the research community

6. **Discuss future work:** Discuss future work in the light of previous results and peer reviews. Did our results generate ideas for new solutions and/or research questions? This discussion will feed the next cycle of the process with potential updates of the active set of research questions and solution ideas.

4.2 Research Methods

Literature surveys form the basis to find the state-of-the-art in relevant research areas. On top of that, controlled experiments and theoretical proofs are the re-

search methods used in this thesis. All papers include controlled experiments where a suggested approach is compared with state-of-the-art approaches. Paper D and E additionally provide theoretical proofs, thereby contributing to establish a theoretical ground for stochastic planning and scheduling.

4.2.1 Experiments

A general goal of an experiment, that matches most experiments in this thesis, is: *Analyze the planning of tasks for industrial mobile robots in collaborative industrial applications by comparing a proposed solution with the state-of-the-art in terms of planning efficiency and plan optimality/robustness.*

Many experiments are focused on validating the outcome of planning algorithms running in isolation, but with realistic input data from industrially relevant use cases. Other experiments, in paper A and B, are executed as part of simulations running on top of the software platform Robot Operating System (ROS)[59], with ROS navigation stack[60] for mobile navigation and Gazebo[61] for modeling and simulation of robots and the application environment. Using simulations reduces the cost and time to develop and verify solutions compared to real systems. However, simulations avoid some important challenges a real system set-up will encounter, for example object recognition and classification from sensor input. No experiments are run on real robots, which is an important future step, for example, to validate system behavior in terms of performance, robustness, resilience and human perception.

Chapter 5

Thesis Contributions

In this section, the contributions are described, and their connection to the research questions as well as to the included papers are indicated.

5.1 Contributions

C1: **Robot Task Scheduling Graph (RTSG) Formalism.** An intuitive, graph-based language for domain experts to encode mobile-robot workflows using start/goal nodes, AND/OR forks, and lock/join constructs—automatically compiled into both MILP and PDDL planning problems. A benchmark on a kitting use case shows that RTSG models yield equivalent-quality plans in both representations, and simulation confirms that its cost abstraction accurately predicts makespan.

C2: **Task Roadmaps for Incremental Replanning.** A two-phase Branch-and-Bound approach (B&B-TRM) that precomputes and stores a reusable search graph ("roadmap") of partial plans. When replanning is needed, it reuses existing branches instead of expanding—achieving replanning times orders of magnitude faster than scratch-built MILP or PDDL solvers while preserving optimality.

C3: **Cluster-and-Balance Heuristic for MRTA.** A scalable, hybrid heuristic that (1) clusters tasks into robot-specific groups via a constraint-aware k-medoids variant, (2) solves intra-cluster routes as TSP instances, (3) resolves robot conflicts, (4) prunes redundant alternatives, and (5) iteratively shifts tasks to balance makespan. On

problems up to 200 tasks and 5 robots, it computes near-optimal schedules in seconds versus minutes for MILP.

C4: **Risk-Aware Single-Robot Planning with Humans.** An RTSG extension for human-robot collaboration that tags human tasks and &JS sync nodes, models all durations as random variables, and uses a stochastic Branch-and-Bound to enumerate plans under first-order stochastic dominance. From this Pareto front, a planner selects a plan via a user-specified risk level or CDF-based comparison, backed by a safe pruning theorem and Monte Carlo validation.

C5: **Stochastic Scheduling Framework for Multi-Agent Teams.** Building on C4, this framework handles multiple robots and humans by (a) defining cross-schedule precedence operators, (b) introducing GRASP and genetic heuristics, (c) implementing a deadlock detection & repair routine, and (d) deriving analytic upper/lower bounds on makespan distributions under uncertainty. It generates executable, risk-bounded schedules in realistic production scenarios.

The contributions are mapped to the research questions in Table 5.1.

5.1.1 C1: Robot Task Scheduling Graph (RTSG) Formalism

The first contribution is the result of RQ 1: *How can we design a graph-based modeling formalism that allows domain experts to specify and organize industrial mobile-robot tasks intuitively, and automatically translate models into both MILP and PDDL planning representations?* Defining a planning problem, with PDDL or other relevant approaches in the literature, for a mobile robot operating in an industrial application is a complex task for a domain expert. This contribution proposes an intuitive graph-based task modeling formalism, Robot Task Scheduling Graph (RTSG), that is used to model the workflow of a robot application using start/goal nodes, AND/OR forks, lock/join constructs and where edges represent precedence constraints. Importantly, RTSG leverage automated planning to find efficient plans. We demonstrate how an RTSG model can be automatically converted into planning problems with two different representations, MILP and PDDL. The approach is investigated in a benchmark study where a kitting application is modeled with RTSG. The study compares one MILP solver and two PDDL planners, and the results support a hypothesis that the two conversions are equivalent. A complementary simulation study corroborates the correlation between estimated and simulated makespan of a generated plan.

5.1.2 C2: Task Roadmaps for Incremental Replanning

The second contribution is the results of RQ 2: *What algorithms and data structures enable low-latency initial planning and incremental replanning of mobile-robot tasks when the environment changes?* This contribution explores the idea that replanning often is more time critical than making an initial plan. The search for efficient plans in a planning algorithm typically involves the expansion of a search space in the form of a tree or a graph. We propose the concept of Task Roadmaps, which is to keep the search space expanded during initial planning and reuse it when replanning is needed. The concept is demonstrated with a novel Branch-And-Bound algorithm, B&B-TRM, able to plan and replan planning problems defined with an RTSG model. A benchmark study of a realistic, simulated kitting application compares the replanning performance of the proposed B&B-TRM algorithm with a MILP solver, a PDDL planner and initial planning with B&B. The study indicates that B&B-TRM is able to generate plans of equivalent optimality within a fraction of the planning time of the other approaches. Complementary experiments investigate the limitations of the approach for scaled up problem instances.

5.1.3 C3: Cluster-and-Balance Heuristic for MRTA

The third contribution is the result of RQ 3: *How can we efficiently plan a fleet of robots for a multi-agent mission, ensuring high-quality solutions within bounded computation time?* This contribution is three-fold. First, a new industrially relevant variant of the Vehicle Routing Problem (VRP) is modeled with a MILP formulation. The proposed variant is the first to include OR-type precedence constraints with alternative predecessors. Secondly, the problem is represented with a Robot Task Scheduling Graph with proposed extensions and interpretations for the modeling of multi-robot applications. The third and the main contribution is the proposal of a heuristic algorithm able to generate suboptimal plans of high quality with a limited planning time small as well as scaled up problem instances. The algorithm is empirically evaluated against a MILP solver in terms of solution optimality and planning time.

5.1.4 C4: Risk-Aware Single-Robot Planning with Humans

The fourth contribution is the result of RQ 4: *How can we incorporate stochastic task-duration models and human risk preferences into a planning framework to generate, evaluate, and select collaborative robot-human plans that balances makespan efficiency and uncertainty?* With this contribution, we leverage modeling of uncertain task durations as independent random variables

with unrestricted distributions, which can be refined from future observations. These stochastic task durations are used to pro-actively compute and select the best plan for a robot assisting human workers, while considering the risk-willingness of a human planner. The main contribution is a methodology for stochastic task planning: Given an RTSG work description, a B&B algorithm (extended from paper B) computes the best plans without a stochastic ordering. From this Pareto front candidate set, a final plan is selected while considering the risk willingness of a human planner. To support this methodology, theoretical contributions provide a safe pruning strategy with a guarantee to never stop the exploration of a better plan, and a novel selection criterion, stochastic set dominance, for full-length plans.

5.1.5 C5: Stochastic Scheduling Framework for Multi-Agent Teams

The fifth contribution is the result of targeting RQ 3 and RQ 4 in combination. The contribution is a novel stochastic scheduling framework targeting real-world collaborative, multi-agent manufacturing applications with several types of cross-schedule dependencies, multi-agent tasks and which considers individual human constraints for preferred activities and desired idle time. It reuses the modeling of uncertain task durations (from C4) as independent random variables. The outcome is an optimized pro-active schedule for a desired risk-level, with a realistic makespan estimate given as a probability distribution. Further, the framework provides theoretically safe bounds on such makespan distributions which arises from dependencies between aggregated random variables in the composition of the makespan. A novel deadlock detection and repair algorithm ensures executability of schedules, preventing cyclic dependencies that may arise between a set of assigned tasks caused by an undesired combination of agent task sequences and precedence constraints. A novel GRASP heuristics, which is benchmarked against a genetic algorithm, provides optimized schedules in a limited time.

Table 5.1: Mapping between contributions and research questions.

	RQ 1	RQ 2	RQ 3	RQ 4
C1	X			
C2		X		
C3			X	
C4				X
C5			X	X

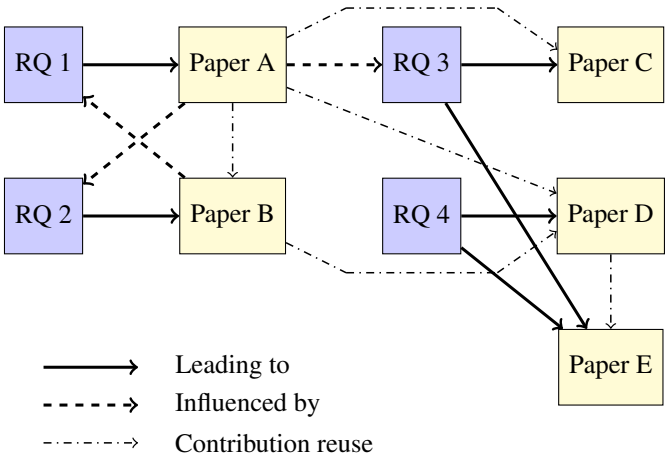

Figure 5.1: Dependencies between research questions and papers.

5.2 Included Papers

In this section, the papers included in the thesis are outlined. These papers are listed and mapped to the contributions in Table 5.2.

Table 5.2: Mapping between papers and contributions.

	C1	C2	C3	C4	C5
Paper A	X				
Paper B		X			
Paper C			X		
Paper D				X	
Paper E					X

An overview of the dependencies between research questions and papers is provided in Figure 5.1. Additionally, it displays the influence between papers, indicating the reuse of contributions. These dependencies and their causes, which are discussed below, reflect the exploratory nature of the research process – resembling a mobile robot navigating and mapping an unknown environment where the outcome is unknown beforehand and revealed on the go.

RQ 1 had a first version and a final version with major differences, and so did the resulting paper A. The first version of RQ 1 was abandoned as a consequence of constructive reviewer feedback. RQ 2, leading to a first and a final version of paper B, was influenced by a planning algorithm introduced in the first version of paper A but removed from its final version. The final RQ

1 was inspired by a review of the first paper B which highlighted the applied RTSG modeling formalism, introduced as a sub-contribution in the first version of paper A, as a very interesting idea. The final RQ 1 entailed a major rework of paper A into the final version, with the RTSG task modeling formalism and its conversions to MILP and PDDL as the main contribution. Paper B was extended into a journal paper in the final version. RQ 3, leading to paper C, was inspired by a review of the final paper A which suggested an extension into multi-agent planning. RQ 4, leading to the stochastic planning approach in paper D, was mainly influenced by the desire to address the challenge of uncertainties in collaborative robot applications. Paper E addressed RQ 4 but also RQ 3, extending the stochastic modeling framework of paper D into multi-agent scheduling.

The RTSG modeling formalism from paper A was reused by paper B and further extended by paper C and D. Paper D extended the B&B algorithm introduced in paper B by adding a stochastic modeling framework. Paper E did not reuse B&B but extended the stochastic modeling framework. Similar to paper A, the initial work of paper E explored the idea of a novel task modeling formalism, but as a complement (to RTSG) for manufacturing processes. However, this part was eventually turned into a motivating example for the proposed multi-agent stochastic scheduling framework.

5.2.1 Paper A

Title: A Task Modelling Formalism for Industrial Mobile Robot Applications.
Authors: Anders Lager, Alessandro V. Papadopoulos, Giacomo Spampinato and Thomas Nolte.
Status: In 20th International Conference on Advanced Robotics (ICAR), 2021.
Abstract: Industrial mobile robots are increasingly introduced in factories and warehouses. These environments are becoming more dynamic with human co-workers and other uncertainties that may interfere with the robot's actions. To uphold efficient operation, the robots should be able to autonomously plan and replan the order of their tasks. On the other hand, the robot's actions should be predictable in an industrial process. We believe the deployment and operation of robots become more robust if the experts of the industrial processes are able to understand and modify the robot's behaviour. To this end, we present an intuitive novel task modelling formalism, Robot Task Scheduling Graph (RTSG). RTSG provides building blocks for the explicit definition of alternative task sequences in a compact graph format. We present how such a graph is automatically converted to a

task planning problem in two different forms, i.e., a Mixed Integer Linear Program (MILP) and a Planning Domain Definition Language specification (PDDL). Converted RTSG models of a mobile kitting application are used to experimentally compare the performance of one MILP planner and two PDDL planners. Besides providing this comparison, the experiments confirm the equivalence of the converted MILP and PDDL problem formulations. Finally, a simulation experiment verifies the assumed correlation between a cost model, based on path lengths, and the makespan.

My role I developed the solution, compared the approach with state-of-the-art, derived the theory and performed the experiments. Alessandro came up with the initial idea to investigate a graph-based task modeling approach. All co-authors supervised the findings of this work, discussed the results and contributed to the development of the final manuscript.

5.2.2 Paper B

Title: Task Roadmaps - Speeding up Task Replanning.
Authors: Anders Lager, Giacomo Spampinato, Alessandro V. Papadopoulos and Thomas Nolte.
Status: In Frontiers in Robotics and AI, section Robotic Control Systems, 2022.
Abstract: Modern industrial robots are increasingly deployed in dynamic environments, where unpredictable events are expected to impact the robot's operation. Under these conditions, runtime task replanning is required to avoid failures and unnecessary stops, while keeping up productivity. Task replanning is a long-sighted complement to path replanning, which is mostly concerned with avoiding unexpected obstacles that can lead to potentially unsafe situations. This paper focuses on task replanning as a way to dynamically adjust the robot behaviour to the continuously evolving environment in which it is deployed. Analogously to probabilistic roadmaps used in path planning, we propose the concept of *Task roadmaps* as a method to replan tasks by leveraging an offline generated search space. A graph-based model of the robot application is converted to a task scheduling problem to be solved by a proposed Branch and Bound (B&B) approach and two benchmark approaches: Mixed Integer Linear Programming (MILP) and Planning Domain Definition Language (PDDL). The B&B approach is proposed to compute the task roadmap, which is then reused to replan for unforeseeable

events. The optimality and efficiency of this replanning approach are demonstrated in a simulation-based experiment with a mobile manipulator in a kitting application. In this study, the proposed B&B Task Roadmap replanning approach is significantly faster than a MILP solver and a PDDL based planner.

My role I conceived the presented idea, developed the theory, and performed the experiments. Alessandro and Giacomo encouraged me to investigate how to improve replanning scenarios for the B&B algorithm. Giacomo conceived the idea to investigate the synergies of the replanning concept with Probabilistic Roadmaps. All co-authors supervised the findings of this work, discussed the results and contributed to the development of the final manuscript.

5.2.3 Paper C

Title: A Scalable Heuristic for Mission Planning of Mobile Robot Teams.
Authors: Anders Lager, Branko Miloradović, Alessandro V. Papadopoulos, Giacomo Spampinato and Thomas Nolte.
Status: In 22nd World Congress of the International Federation of Automatic Control (IFAC), 2023.
Abstract: In this work, we investigate a task planning problem for assigning and planning a mobile robot team to jointly perform a kitting application with alternative task locations. To this end, the application is modeled as a Robot Task Scheduling Graph and the planning problem is modeled as a Mixed Integer Linear Program (MILP). We propose a heuristic approach to solve the problem with a practically useful performance in terms of scalability and computation time. The experimental evaluation shows that our heuristic approach is able to find efficient plans, in comparison with both optimal and non-optimal MILP solutions, in a fraction of the planning time.

My role I suggested the approach, modeled and developed the solution and performed all experiments. Branko provided modeling input on the makespan objective and suggested investigating the usage of TSP solvers for single route computations. Alessandro encouraged me to investigate the modeling aspects with RTSG. All authors supervised the work, discussed the results, and contributed to the final manuscript.

5.2.4 Paper D

Title: Risk Aware Planning of Collaborative Mobile Robot Applications with Uncertain Task Durations.
Authors: Anders Lager, Branko Miloradović, Giacomo Spampinato, Thomas Nolte and Alessandro V. Papadopoulos.
Status: In 33rd IEEE International Conference on Robot and Human Interactive Communication (RO-MAN), 2024.
Abstract: The efficiency of collaborative mobile robot applications is influenced by the inherent uncertainty introduced by humans' presence and active participation. This uncertainty stems from the dynamic nature of the working environment, various external factors, and human performance variability. The observed makespan of an executed plan will deviate from any deterministic estimate. This raises questions about whether a calculated plan is optimal given uncertainties, potentially risking failure to complete the plan within the estimated timeframe. This research addresses a collaborative task planning problem for a mobile robot serving multiple humans through tasks such as providing parts and fetching assemblies. To account for uncertainties in the durations needed for a single robot and multiple humans to perform different tasks, a probabilistic modeling approach is employed, treating task durations as random variables. The developed task planning algorithm considers the modeled uncertainties while searching for the most efficient plans. The outcome is a set of the best plans, where no plan is better than the other in terms of stochastic dominance. Our proposed methodology offers a systematic framework for making informed decisions regarding selecting a plan from this set, considering the desired risk level specific to the given operational context.

My role I proposed the research question (RQ 4) for this novel direction of the thesis, made the literature review and proposed the targeted knowledge gap, that is a stochastic modeling of task durations for task planning. I proposed the use case scenario and developed the modeling and the planning approach. I developed the theory and performed all experiments. Alessandro suggested to investigate convolutions, stochastic dominance and assisted in deriving the MAX operator. All authors supervised the work, discussed the results and contributed to the final manuscript.

5.2.5 Paper E

Title: Stochastic Scheduling for Human-Robot Collaboration in Dynamic Manufacturing Environments.
Authors: Anders Lager, Branko Miloradović, Giacomo Spampinato, Thomas Nolte and Alessandro V. Papadopoulos.
Status: Accepted by 34th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN), 2025.
Abstract: Collaborative human-robot teams enhance efficiency and adaptability in manufacturing, but task scheduling in mixed-agent systems remains challenging due to the uncertainty of task execution times and the need for synchronization of agent actions. Existing task allocation models often rely on deterministic assumptions, limiting their effectiveness in dynamic environments. We propose a stochastic scheduling framework that models uncertainty through probabilistic makespan estimates, using convolutions and stochastic max operators for realistic performance evaluation. Our approach employs meta-heuristic optimization to generate executable schedules aligned with human preferences and system constraints. It features a novel deadlock detection and repair mechanism to manage cross-schedule dependencies and prevent execution failures. This framework offers a robust, scalable solution for real-world human-robot scheduling in uncertain, interdependent task environments.

My role I developed the motivating manufacturing scenario and the general theory. Specifically, I found the inherent problem of dependencies between operands of stochastic operators, which prevented a straight-forward analytical makespan computation. To overcome, I developed the novel makespan computation approach and suggested it would represent a safe stochastic upper bound if we could prove a basic mathematical property of the stochastic max operator. This proof was derived by Alessandro, and he additionally provided a lower bound for the max operator. Supported by these results, I developed the proof of the upper and lower bounds of the novel makespan computation approach and incorporated these bounds into the cost function to provide conservative estimates of both makespan and human idle times. Alessandro suggested to investigate how to consider preferences of human agents. Branko was helpful in the development of the genetic algorithm and suggested to investigate a GRASP based solution of the planning problem. I did all programming and developed all experiments. All authors supervised the work, discussed the results and contributed to the final manuscript.

5.3 Other Publications

Publications listed here are not included in the licentiate thesis:

- Anders Lager, Alessandro V. Papadopoulos, Giacomo Spampinato and Thomas Nolte. *Towards Reactive Robot Applications in Dynamic Environments.* In 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), 2019.
- Anders Lager, Alessandro V. Papadopoulos, Giacomo Spampinato and Thomas Nolte. *IoT and Fog Analytics for Industrial Robot Applications.* In 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), 2020.

Chapter 6

Conclusions and Future Work

In this thesis the general challenge,

How can we design an end-to-end, domain-expert-friendly planning framework that enables industrial users to intuitively specify and organize mobile-robot tasks, while generating efficient, scalable, dynamically adaptable, and risk-aware plans for both single- and multi-robot systems operating in uncertain, human-collaborative environments?

is addressed with four research questions (RQ1-RQ4) and five resulting contributions (C1-C5).

6.1 Summary of Contribution

RQ1 is identified from the desire to assist domain experts in the creation of intuitive robot work descriptions while leveraging the efficiency of automated planning: *How can we design a graph-based modeling formalism that allows domain experts to specify and organize industrial mobile-robot tasks intuitively, and automatically translate models into both MILP and PDDL planning representations?* The resulting contribution (C1) is the Robot Task Scheduling Graph (RTSG) Formalism and its conversions to MILP and RTSG. Extensive experiments confirm that the proposed conversions yield equivalent plan optimality with state-of-the-art solvers/planners.

RQ2 targets the need for efficient planning and replanning of highly complex task planning problems by leveraging the data structures of algorithmic search spaces: *What algorithms and data structures enable low-latency initial planning and incremental replanning of mobile-robot tasks when the environment changes?* The resulting contribution (C2) is Task Roadmaps for Incremental Replanning. Experiments indicate the re-planning time of the

proposed algorithm, B&B-TRM, is significantly faster than B&B, PDDL and MILP planners/solvers.

RQ3 extends planning scenarios to include multi-agent applications: *How can we efficiently plan a fleet of robots for a multi-agent mission, ensuring high-quality solutions within bounded computation time?* One contribution (C3) is a Cluster-and-Balance Heuristic for MRTA. It is used to solve a novel type of VRP problem, modeled with RTSG as a multi-agent kitting application. Benchmark Experiments with a proposed MILP problem formulation verify near-optimality of plans and superior planning times with good scalability.

RQ4 is identified from the desire to manage task planning in dynamic environments with humans in the loop of task execution and planning: *How can we incorporate stochastic task-duration models and human risk preferences into a planning framework to generate, evaluate, and select collaborative robot-human plans that balances makespan efficiency and uncertainty?* One contribution (C4) is Risk-Aware Single-Robot Planning with Humans. It is used to search for and select the best proactive plans for any risk willingness level. To capture the task execution uncertainty, task durations are modeled as unrestricted random variables in a workflow modeled with RTSG for a single robot assisting multiple humans. The algorithmic search approach includes a theoretically safe pruning strategy based on stochastic dominance and is benchmarked against a deterministic version.

Finally, RQ3 and RQ4 are addressed in combination, leading to the contribution Stochastic Scheduling Framework for Multi-Agent Teams (C5). Extending the stochastic modeling of task durations from C4, it provides a proactive, risk-aware and deadlock-free scheduling approach for a real-world multi-agent and collaborative production scenario with various types of cross-schedule task dependencies. Derived stochastic makespans provide conservative estimates within proven theoretical bounds. Additionally, human ergonomics (preferred idle time) and preferred human activities are considered by the scheduling approach.

6.2 Future Work

Future work may investigate the following RQs, from technical aspects to a broader question:

- How input distributions can be efficiently learned and improved over time?
- How to efficiently combine the strengths of proactive and reactive planning?

- What are the key challenges of human-robot and human-system interactions for a sustainable, human-centric and efficient real-world, industrial, collaborative multi-agent system ?

Bibliography

[1] Alessandro Gasparetto and Lorenzo Scalera. A brief history of industrial robotics in the 20th century. *Advances in Historical Studies*, 8(1):24–35, 2019.

[2] Anders Lager, Alessandro Papadopoulos, and Thomas Nolte. Iot and fog analytics for industrial robot applications. In *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, volume 1, pages 1297–1300. IEEE, 2020.

[3] Mitchell M Tseng and Jianxin Jiao. Mass customization. *Handbook of industrial engineering*, 3:684–709, 2001.

[4] Int. Organization for Standardization Standard ISO/TS 15066:2016. Robots and robotic devices-collaborative robots, 2016.

[5] Arash Ajoudani, Andrea Maria Zanchettin, Serena Ivaldi, Alin Albu-Schäffer, Kazuhiro Kosuge, and Oussama Khatib. Progress and prospects of the human–robot collaboration. *Autonomous Robots*, 42(5):957–975, 2018.

[6] Andie Zhang. Collaborative robots–enabling smes to automate in post-pandemic world. In *ISR Europe 2022; 54th International Symposium on Robotics*, pages 1–7. VDE, 2022.

[7] H. Nakawala, P. J. S. Goncalves, P. Fiorini, G. Ferringo, and E. D. Momi. Approaches for action sequence representation in robotics: A review. In *IROS*, pages 5666–5671, 2018.

[8] Zichen Wang, Jingyi Wang, Fu Song, Kun Wang, Hongyi Pu, and Peng Cheng. K-rapid: A formal executable semantics of the rapid robot programming language. In *Proceedings of the 10th ACM Cyber-Physical System Security Workshop*, pages 64–76, 2024.

[9] David Weintrop, Afsoon Afzal, Jean Salac, Patrick Francis, Boyang Li, David C Shepherd, and Diana Franklin. Evaluating coblox: A comparative study of robotics programming environments for adult novices. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, pages 1–12, 2018.

[10] Ole Madsen, Simon Bøgh, Casper Schou, Rasmus Skovgaard Andersen, Jens Skov Damgaard, Mikkel Rath Pedersen, and Volker Krüger. Inte-

gration of mobile manipulators in an industrial production. *Industrial Robot: An International Journal*, 42(1):11–18, 2015.

[11] Jonathan Bohren and Steve Cousins. The smach high-level executive [ros news]. *IEEE Robotics & Automation Magazine*, 17(4):18–20, 2010.

[12] Andrea Casalino, Andrea Maria Zanchettin, Luigi Piroddi, and Paolo Rocco. Optimal scheduling of human–robot collaborative assembly operations with time petri nets. *IEEE Transactions on Automation Science and Engineering*, 18(1):70–84, 2019.

[13] Petter Ögren and Christopher I. Sprague. Behavior trees in robot control systems. *Annual Review of Control, Robotics, and Autonomous Systems*, 5(Volume 5, 2022):81–107, 2022.

[14] Marius Kloetzer and Cristian Mahulea. Path planning for robotic teams based on ltl specifications and petri net models. *Discrete Event Dynamic Systems*, 30(1):55–79, 2020.

[15] Fengming Li, Qi Jiang, Sisi Zhang, Meng Wei, and Rui Song. Robot skill acquisition in assembly process using deep reinforcement learning. *Neurocomputing*, 345:92–102, 2019.

[16] Matthew Crosby, Francesco Rovida, Mikkel Rath Pedersen, Ronald P. A. Petrick, and Volker Krüger. Planning for robots with skills. In *Workshop on Planning and Robotics (PlanRob)*, pages 49–57, 2016.

[17] James A Hendler, Austin Tate, and Mark Drummond. Ai planning: Systems and techniques. *AI magazine*, 11(2):61–61, 1990.

[18] Michele Lombardi and Michela Milano. Optimal methods for resource allocation and scheduling: a cross-disciplinary survey. *Constraints*, 2012.

[19] Maria Fox and Derek Long. Pddl2. 1: An extension to pddl for expressing temporal planning domains. *Journal of artificial intelligence research*, 20:61–124, 2003.

[20] Richard E Fikes and Nils J Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3-4):189–208, 1971.

[21] Andrew Coles, Amanda Coles, Allan Clark, and Stephen Gilmore. Cost-sensitive concurrent planning under duration uncertainty for service-level agreements. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 21, pages 34–41, 2011.

[22] Patrick Eyerich, Robert Mattmüller, and Gabriele Röger. Using the context-enhanced additive heuristic for temporal and numeric planning. In *ICAPS*, 2009.

[23] Håkan LS Younes and Michael L Littman. Ppddl1. 0: An extension to pddl for expressing planning domains with probabilistic effects. *Techn. Rep. CMU-CS-04-162*, 2:99, 2004.

[24] Dana Nau, Tsz-Chiu Au, Okhtay Ilghami, Ugur Kuter, J William Murdock, Dan Wu, and Fusun Yaman. Shop2: An htn planning system. *J. Artif. Intell. Res. (JAIR)*, 20:379–404, 2003.

[25] Rahul Kala. Mission planning on preference-based expression trees using heuristics-assisted evolutionary computation. *Applied Soft Computing*, 136:110090, 2023.

[26] Rahul Kala. Operational probability aware mission planning on expression trees using evolutionary computation. *Evolutionary Intelligence*, 18(4):81, 2025.

[27] Caelan Reed Garrett, Rohan Chitnis, Rachel Holladay, Beomjoon Kim, Tom Silver, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Integrated task and motion planning. *Annual review of control, robotics, and autonomous systems*, 4(1):265–293, 2021.

[28] Yoshio Yamamoto and Xiaoping Yun. Coordinating locomotion and manipulation of a mobile manipulator. In *[1992] Proceedings of the 31st IEEE Conference on Decision and Control*, pages 2643–2648. IEEE, 1992.

[29] Hugh Durrant-Whyte and Tim Bailey. Simultaneous localization and mapping: part i. *IEEE robotics & automation magazine*, 13(2):99–110, 2006.

[30] Brian P Gerkey and Maja J Matarić. A formal analysis and taxonomy of task allocation in multi-robot systems. *The Int. Journal of Robotics Research*, 23(9):939–954, sep 2004.

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

[32] Alaa Khamis, Ahmed Hussein, and Ahmed Elmogy. *Multi-robot Task Allocation: A Review of the State-of-the-Art*, pages 31–51. Springer International Publishing, 2015.

[33] Stefan Thiemermann. *Direkte Mensch-Roboter-Kooperation in der Kleinteilemontage mit einem SCARA-Roboter*. PhD thesis, Jost-Jetter Verlag, 2005.

[34] Eloise Matheson, Riccardo Minto, Emanuele GG Zampieri, Maurizio Faccio, and Giulio Rosati. Human–robot collaboration in manufacturing applications: A review. *Robotics*, 8(4):100, 2019.

[35] Xiaodan Wang, Rossitza Setchi, and Abdullah Mohammed. Modelling uncertainties in human-robot industrial collaborations. *Procedia Computer Science*, 207:3652–3661, 2022.

[36] Anders Billesø Beck, Anders Due Schwartz, Andreas Rune Fugl, Martin Naumann, and Björn Kahl. Skill-based exception handling and error recovery for collaborative industrial robots. In *FinE-R@ IROS*, pages 5–10, 2015.

[37] Min Chen, Emilio Frazzoli, David Hsu, and Wee Sun Lee. Pomdp-lite for robust robot planning under uncertainty. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5427–5433. IEEE, 2016.

[38] Stephen B Stancliff, John Dolan, and Ashitey Trebi-Ollennu. Planning to fail—reliability needs to be considered a priori in multirobot task allocation. In *2009 IEEE International Conference on Systems, Man and Cybernetics*, pages 2362–2367. IEEE, 2009.

[39] Jinwoo Park, Andrew Messing, Harish Ravichandar, and Seth Hutchinson. Risk-tolerant task allocation and scheduling in heterogeneous multi-robot teams. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5372–5379. IEEE, 2023.

[40] Jonathan Gough, Maria Fox, and Derek Long. Plan execution under resource consumption uncertainty. In *Proceedings of the Workshop on Connecting Planning Theory with Practice at ICAPS*, volume 4, pages 24–29, 2004.

[41] Shaobo Zhang, Yi Chen, Jun Zhang, and Yunyi Jia. Real-time adaptive assembly scheduling in human-multi-robot collaboration according to human capability. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3860–3866. IEEE, 2020.

[42] Anders Lager, Giacomo Spampinato, Alessandro V Papadopoulos, and Thomas Nolte. Towards reactive robot applications in dynamic environments. In *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1603–1606. IEEE, 2019.

[43] G. Kazhoyan, A. Niedzwiecki, and M. Beetz. Towards plan transformations for real-world mobile fetch and place. In *IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 11011–11017, 2020.

[44] D. Hadfield-Menell, L. P. Kaelbling, and T. Lozano-Pérez. Optimization in the now: Dynamic peephole optimization for hierarchical planning. In *2013 IEEE Int. Conf. on Robotics and Automation*, pages 4560–4567, 2013.

[45] Martin Weser, Dominik Off, and Jianwei Zhang. Htn robot planning in partially observable dynamic environments. In *2010 IEEE International Conference on Robotics and Automation*, pages 1505–1510. IEEE, 2010.

[46] Branko Miloradović. *Multi-agent mission planning*. Malardalen University (Sweden), 2022.

[47] Ping Lou, Quan Liu, Zude Zhou, Huaiqing Wang, and Sherry Sun. Multi-agent-based proactive–reactive scheduling for a job shop. *Int. Journal of Advanced Manufacturing Technology - INT J ADV MANUF TECHNOL*, 59, 03 2012.

[48] Oscar Lima, Michael Cashmore, Daniele Magazzeni, Andrea Micheli, and Rodrigo Ventura. Robust plan execution with unexpected observations, 2020.

[49] Gianluca Filippone, Juan Antonio Piñera García, Marco Autili, and Patrizio Pelliccione. Handling uncertainty in the specification of autonomous multi-robot systems through mission adaptation. In *Proceedings of the 19th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 25–36, 2024.

[50] Javier G Martin, José Ramón Domínguez Frejo, Ramón A García, and Eduardo F Camacho. Multi-robot task allocation problem with multiple nonlinear criteria using branch and bound and genetic algorithms. *Intelligent Service Robotics*, 14(5):707–727, 2021.

[51] Hanfu Wang and Weidong Chen. Simulated annealing algorithms for the heterogeneous robots task scheduling problem in heterogeneous robotic

order fulfillment systems. In *International conference on intelligent autonomous systems*, pages 276–287. Springer, 2021.

[52] Changyun Wei, Ze Ji, and Boliang Cai. Particle swarm optimization for cooperative multi-robot task allocation: a multi-objective approach. *IEEE Robotics and Automation Letters*, 5(2):2530–2537, 2020.

[53] Mausam Natarajan and Andrey Kolobov. *Planning with Markov decision processes: An AI perspective*. Springer Nature, 2022.

[54] Hanna Kurniawati. Partially observable markov decision processes and robotics. *Annual Review of Control, Robotics, and Autonomous Systems*, 5:253–277, 2022.

[55] Michael Saint-Guillain, Tiago Vaquero, Steve Chien, Jagriti Agrawal, and Jordan Abrahams. Probabilistic temporal networks with ordinary distributions: Theory, robustness and expected utility. *Journal of Artificial Intelligence Research*, 71:1091–1136, 2021.

[56] Andrew W Palmer, Andrew J Hill, and Steven J Scheding. Modelling resource contention in multi-robot task allocation problems with uncertain timing. In *IEEE Int. Conf. ICRA*, pages 3693–3700, 2018.

[57] Andrea Casalino and Angelo Geraci. Allowing a real collaboration between humans and robots. *Special Topics in Information Technology*, page 139, 2021.

[58] Franco Cordeiro, Samuel Tardieu, and Laurent Pautet. Rescue: Multi-robot planning under resource uncertainty and objective criticality. In *37th Euromicro Conference on Real-Time Systems (ECRTS 2025)*, pages 5–1. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2025.

[59] Morgan Quigley, Brian Gerkey, Ken Conley, Josh Faust, Tully Foote, Jeremy Leibs, Eric Berger, Rob Wheeler, and Andrew Ng. Ros: an open-source robot operating system. In *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA) Workshop on Open Source Robotics*, Kobe, Japan, may 2009.

[60] Rodrigo Longhi Guimarães, André Schneider de Oliveira, João Alberto Fabro, Thiago Becker, and Vinícius Amilgar Brenner. Ros navigation: Concepts and tutorial. *Robot Operating System (ROS) The Complete Reference (Volume 1)*, pages 121–160, 2016.

[61] N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, volume 3, pages 2149–2154 vol.3, 2004.

II

Included Papers

Chapter 7

Paper A
A Task Modelling Formalism for Industrial Mobile Robot Applications

Anders Lager, Alessandro V. Papadopoulos, Giacomo Spampinato and Thomas Nolte. In 20th International Conference on Advanced Robotics (ICAR), 2021.

Abstract

Industrial mobile robots are increasingly introduced in factories and warehouses. These environments are becoming more dynamic with human coworkers and other uncertainties that may interfere with the robot's actions. To uphold efficient operation, the robots should be able to autonomously plan and replan the order of their tasks. On the other hand, the robot's actions should be predictable in an industrial process. We believe the deployment and operation of robots become more robust if the experts of the industrial processes are able to understand and modify the robot's behaviour. To this end, we present an intuitive novel task modelling formalism, Robot Task Scheduling Graph (RTSG). RTSG provides building blocks for the explicit definition of alternative task sequences in a compact graph format. We present how such a graph is automatically converted to a task planning problem in two different forms, i.e., a Mixed Integer Linear Program (MILP) and a Planning Domain Definition Language specification (PDDL). Converted RTSG models of a mobile kitting application are used to experimentally compare the performance of one MILP planner and two PDDL planners. Besides providing this comparison, the experiments confirm the equivalence of the converted MILP and PDDL problem formulations. Finally, a simulation experiment verifies the assumed correlation between a cost model, based on path lengths, and the makespan.

7.1 Introduction

To support human labour with repetitive, non-ergonomic and simple tasks, the need is ever increasing for having mobile robots able to perform versatile industrial robot tasks like kitting and machine tending.

For efficient operation in a dynamic, collaborative working space where unexpected situations are expected to occur, the ability to plan and replan tasks autonomously in a robust way becomes a key success factor. The maturity of AI Planning has seen tremendous progress over the last decades and several modelling formalisms with high expressiveness have been demonstrated successfully in industrial robot applications and other domains [1, 2]. However, these modelling formalisms are often complex and do not take advantage of the domain expert's intuition and skills in understanding what is a valid task sequence. We consider a domain expert as someone who has expert knowledge in the tasks that the robot shall perform in a certain industrial context—not an expert in robot programming. We strongly believe that enabling the competence of domain experts is crucial to reduce the threshold for the successful commissioning of competitive industrial robot applications on a larger scale.

In industrial robot applications, it is important to avoid unexpected action sequences that reach a defined goal state, at the price of potentially unexpected and undesired side effects. Finding a *feasible* plan is often easy since working procedures typically are well organized. The challenge often lies in finding an efficient plan.

In this paper, we present a novel modelling formalism, Robot Task Scheduling Graph (RTSG), that addresses these problems while leveraging AI planning. One goal with RTSG is to combine the knowledge and experience of domain experts with the efficiency of automated planning/scheduling. The modelling formalism provides building blocks for describing variable sequences of robot actions to reach high-level goals. As RTSG is graph-based, it enables an intuitive visual overview.

We do not claim RTSG to be the most expressive modelling approach but we argue it is sufficiently expressive for a semi-structured industrial mobile robot application. We present how an RTSG model can be *automatically* converted to a task scheduling problem in two different forms: Mixed Integer Linear Programming (MILP) and Planning Domain Definition Language (PDDL) [3]. This enables RTSG to be used with MILP solvers as well as PDDL-based planners. Improving the efficiency of these planners is a relevant problem, but it is beyond the scope of this paper that focuses on the capabilities of the modelling formalism.

An experimental comparison of the performance for two PDDL-based

planners and one MILP solver is presented. These experiments are applied to modelled use cases for a mobile kitting application, measuring the planning time and the efficiency of generated task sequences. Additionally, the results indicate an equivalence of the MILP and the PDDL representations of the RTSG scheduling problem. Finally, in a simulation study, we show that a transition cost model based on path lengths is a valid approach to minimize the resulting makespan.

The rest of the paper is organized as follows. Section 7.2 presents the related work. Section 7.3 gives an intuitive description of the RTSG modelling formalism. Section 7.4 describes a conversion from RTSG to MILP. Section 7.5 describes a conversion from RTSG to PDDL. Section 7.6 presents the experimental results, while Section 7.7 concludes the paper.

7.2 Related work

RTSG fills a gap between existing modelling formalisms of robot action sequences by combining a desirable set of properties:

- Intuitive modelling approach for a domain expert.
- Intended for use with an automated planner/scheduler to generate efficient task sequences.
- Leverages domain experts intuition, skills and knowledge on a suitable variability of task sequences.

A recent literature review investigated different approaches for representations of action sequences used for robot task planning and execution in a dynamic environment [4]. A selection of these representation approaches is given in Table 7.1. The selection covers all representations available in ROS that have been used with industrial robot applications. Additionally, robot skills, block-based programming and Behaviour Trees have been added. The first column, indicating the intuitiveness of the formalism, is subjective and not backed with empirical evidence. The indicated existence of properties for the second and the third column is indicated by the referenced works.

PDDL [5] is an expressive modelling formalism to set up general planning problems (see Section 7.5.1). Modelling a planning problem is focused on creating objects in the world and give facts about them and their mutual relations while providing operators that may change these facts to reach a goal condition. However, the representation is not intuitive for a domain expert, and there is no explicit way to indicate preferences on action sequences.

Table 7.1: Properties of modelling formalisms for industrial robot applications

Modelling formalism	Intuitive for domain expert	Task sequence variability guided by domain expert	Automated planning/ scheduling
PDDL	–	–	✓
HTN	–	✓	✓
CRAM	–	✓	–
RTSG	✓	✓	✓
Robot Skills	✓	–	✓
State Machine	(✓)	✓	–
Petri Net	(✓)	✓	–
Behavior Trees	(✓)	✓	–
Block-based Programming	✓	–	–

Hierarchical Task Networks (HTN) is another general modelling formalism [6]. In HTN, modelling is about specifying partial orders of tasks. It supports compound tasks that can be decomposed by alternative methods into smaller subtasks in a desired partial order. This provides a way to indicate alternative task sequences. Primitive subtasks correspond to operators in PDDL. The specification of preconditions and variables for methods and operators, the specification of operator effects and the flexible binding of variables give great expressiveness. However, managing these general concepts can be challenging for a domain expert.

Similar to HTN, Cognitive Robotic Abstract Machine (CRAM) [7] that is based on Reactive Plan Language [8] has an action-centric modelling formalism. It is an expressive programming language supporting a hierarchical task structure. The general purpose of CRAM is to be a tool to write robust robot control programs. However, it does not provide support for automated planning and a programming language is not intuitive for a domain expert.

Robot skills build on the idea that the knowledge of an expert of robot programming can be encoded into the implementation of tasks in the form of reusable skills that can be used as simple building blocks when modelling a robot application. These skills can include preconditions and effects to support runtime execution but also automated planning [9]. However, this concept does not explicitly support using a domain expert's intuition of the task sequence variability.

Block-based programming is primarily intended to simplify programming

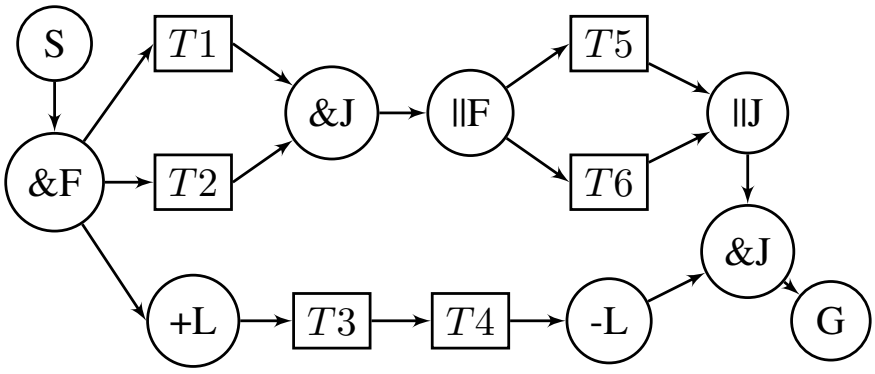

Figure 7.1: Robot Task Scheduling Graph.

by providing configurable building blocks, e.g., CoBlox [10].

Other general modelling formalism's can be used for modelling complex robot behaviour and guiding the execution e.g., State Machines [11], PetriNets [12, 13] and Behaviour Trees [14]. These powerful modelling techniques are not primarily intended for automated planning and the modelling complexity can be challenging.

In some approaches, no apparent high-level modelling formalism is used and the problem representation is purely mathematical, typically in the form of an optimization problem, e.g., [15]. This can give good results but is less intuitive and the problem formulation is harder to adapt to meet new requirements.

In assembly applications, directed graphs, AND/OR-graphs [16] (based on the assembly parts) or precedence graphs [17] (based on assembly operations) can be used to model the variability of assembly sequences that will fulfil a specified assembly. ASML is a later approach [18]. These techniques are used to search for a good design for manufacturability but also for finding an efficient assembly sequence. They are not intended for runtime task planning/replanning. RTSG presents an approach akin to assembly modelling formalisms, to guide automated scheduling of tasks.

7.3 RTSG Modelling formalism

With the building blocks informally described below, RTSG provides a modelling formalism for a domain expert that guides the selection of a task sequence to be decided by an automatic planner with respect to some optimization objective.

An RTSG model, as exemplified in Figure 7.1, is a directed acyclic graph having one start node (S) with a single outgoing edge representing an initial state. At the other end, there is one goal node (G) with a single incoming edge representing a desired goal state. In between, a set of robot tasks leading towards the goal are represented by rectangular nodes having a single incoming edge and a single outgoing edge. In addition, a collection of *logical* nodes impose different scheduling dependencies between tasks. Edges represent precedence constraints: If there is a directed path between two tasks, e.g., $T1$ and $T5$, the first task must precede the latter task in a plan where both tasks are scheduled. AND-Pairs split the graph with an AND-Fork node (&F) into parallel branches and rejoin them with an AND-Join node (&J). Tasks in different AND-Fork branches, e.g., $T1$, $T2$ and $T3$ may be scheduled in any mutual order since there is no directed path between them. OR-Pairs split and rejoin the graph in a similar way with an OR-Fork (||F) and an OR-Join (||J). However, the resulting parallel branches represent alternatives, i.e., only tasks in one of the branches, e.g., $T5$ or $T6$, will be scheduled. Lock-Pairs encapsulate a part of a single branch between an AddLock node (+L) and a RemoveLock node (-L). The set of tasks between a Lock-pair, e.g., $\{T3, T4\}$ will be scheduled as a coherent sub-array of the full task sequence, i.e., uninterrupted by other tasks. Pairs may encapsulate other pairs in a hierarchy, e.g., an OR-Pair may contain other OR-pairs that split alternative branches into sub-alternatives.

Complementing the graph, a task cost estimation should be provided. Apart from specifying the cost of performing different tasks, it should include transition costs between any pair of tasks allowed by the RTSG model to be scheduled in consecutive order.

7.4 Conversion from RTSG to MILP

Section 7.4.1 presents how an RTSG model is converted to a MILP scheduling problem with decision variables, optimization objective and constraints. Section 7.4.2 specifies how the scheduling problem is modified in a replanning scenario.

7.4.1 Conversion from RTSG to MILP

Notation

A is the set of all task nodes, S is the start node and G is the goal node. The following notation is used when combining them: $A^S = A \cup S$, $A^G = A \cup G$ and $\tilde{A} = A \cup S \cup G$. $O \subseteq A$ denotes all tasks encapsulated by OR-Pairs. $j \prec k$

indicates that j precedes k, where $j, k \in \tilde{A}$.

Variables and objective

Decision variables are given by $X_{j,k} \in \{0, 1\} \quad j, k \in \tilde{A}$.

$$X_{j,k} = \begin{cases} 1, & \text{if task } j \text{ is followed by task } k. \\ 0, & \text{otherwise.} \end{cases}$$

$K_{j,k} \in \mathbb{R}_{\geq 0}$ represents the cost for performing task k after task j:

$$K_{j,k} = \tau_{j,k} + \alpha_k \tag{7.1}$$

where $\tau_{j,k}$ is the transition cost and α_k is the action cost.

The objective is to minimize the cost function J:

$$J = \sum_{j \in A^S} \sum_{k \in A^G} X_{j,k} K_{j,k}. \tag{7.2}$$

General constraints

$$X_{j,j} = 0 \qquad \forall j \in \tilde{A} \tag{7.3}$$

$$\sum_{k \in A^G} X_{j,k} = 1 \qquad \forall j \in A^S \setminus O \tag{7.4}$$

$$\sum_{k \in A^G} X_{j,k} \leq 1 \qquad \forall j \in O \tag{7.5}$$

$$\sum_{j \in A^S} X_{j,k} = 1 \qquad \forall k \in A^G \setminus O \tag{7.6}$$

$$\sum_{j \in A^S} X_{j,k} \leq 1 \qquad \forall k \in O \tag{7.7}$$

$$\sum_{k \in A^G} X_{j,k} = \sum_{k \in A^S} X_{k,j} \qquad \forall j \in O \tag{7.8}$$

$$\sum_{j \in A^G} X_{j,S} = 0 \tag{7.9}$$

$$\sum_{k \in A^S} X_{G,k} = 0 \tag{7.10}$$

$$\sum_{j \in V} \sum_{k \in V} X_{j,k} \leq |V| - 1 \qquad \forall V \subseteq A, V \neq \emptyset \tag{7.11}$$

There can be no transition between the same task (7.3). Tasks outside OR-Pairs will occur once (7.4) and (7.6). Tasks inside OR-Pairs will occur at most once (7.5), (7.7), and (7.8). There is no transition to the start state (7.9) and there is no transition from the goal state (7.10). There can be no cyclic sub-tours between tasks (7.11).

Precedence constraints

Assuming $D \subseteq \tilde{A}$ is any ordered subset with elements D_i. Precedence constraints must hold for these subsets in general and especially if they become a sub-array of the task sequence:

$$\sum_{j=1}^{|D|-1} X_{D_j,D_{j+1}} \leq |D| - 2 \quad \forall D \subseteq \tilde{A},\, |D| \geq 2,\, D_{|D|} \prec D_1. \tag{7.12}$$

Lock constraints

$L \subseteq A$ are the set of all tasks encapsulated by a Lock-Pair.

The first tasks in L are defined as $L^F = \{a \in L \mid b \not\prec a \quad \forall b \in L\}$. The last tasks in L are defined as $L^L = \{a \in L \mid a \not\prec b \quad \forall b \in L\}$.

$$X_{j,k} = 0 \qquad \forall L, \forall j \in A^S \setminus L, \forall k \in L \setminus L^F \tag{7.13}$$

$$X_{j,k} = 0 \qquad \forall L, \forall j \in L \setminus L^L, \forall k \in A^G \setminus L \tag{7.14}$$

$$\sum_{j \in A^S \setminus L} \sum_{k \in L^F} X_{j,k} \leq 1 \quad \forall L \tag{7.15}$$

$$\sum_{j \in L^L} \sum_{k \in A^G \setminus L} X_{j,k} \leq 1 \quad \forall L \tag{7.16}$$

There can only be transitions to the first tasks from external tasks (7.13) and there can at most be one such transition (7.15). Similarly, there can only be transitions from the last tasks to external tasks (7.14) and there can at most be one such transition (7.16).

OR-Pair constraints

The constraints presented here can handle a nested structure of OR-Pairs. However, a potential simplification of such a structure, e.g., with algebraic rules, is out of the scope for this work. If an OR-pair is contained by an outer OR-Pair, it is denoted an *internal* OR-Pair and at most one of its alternative branches will be scheduled. The outermost OR-Pairs are denoted *external*

OR-Pairs and exactly one of their branches will be scheduled. OR-Pairs contain *OP nodes* that can be of two types: tasks and internal OR-Pairs.

$O_1, \ldots, O_v$ are sets of OP nodes contained by OR-Pair $1, \ldots, v$. $O_{p1}, \ldots, O_{pm}$ are the set of OP nodes contained by branches $1, \ldots, m$ of OR-Pair p. $O^T_{pq} = \{a \in O_{pq} \mid a \text{ is a task}\}$. $O^{OP}_{pq} = \{a \in O_{pq} \mid a \text{ is an internal OR-Pair}\}$

One *primary* OP node, $P_{pq} \in O_{pq}$, is arbitrary selected for each OR-Pair branch.

Three help operators (7.17), (7.18), and (7.19) are defined to support the definition of OR-Pair constraints. Note that operators F and H return a set while R returns a *set of sets*.

$$F(O_{pq}) = \begin{cases} \{a\} & \text{if } P_{pq} \text{ is task } a. \\ F(O_{r1}) \cup \ldots \cup F(O_{rm}) & \text{if } P_{pq} \text{ is OR-pair } O_r \end{cases} \tag{7.17}$$

$$H(O_p) = F(O_{p1}) \cup F(O_{p2}) \cup \ldots \cup F(O_{pm}) \tag{7.18}$$

$$R(O_{pq}) = O^T_{pq} \setminus P_{pq} \cup \bigcup_{i \in O^{OP}_{pq} \setminus P_{pq}} \{H(i)\} \tag{7.19}$$

Given these definitions, the OR-Pair constraints can be summarized:

$$\sum_{j \in A^G} \sum_{q=1}^{m} \sum_{s \in F(O_{pq})} X_{s,j} = 1 \qquad \forall \text{ external } O_p \tag{7.20}$$

$$\sum_{j \in A^S} \sum_{q=1}^{m} \sum_{s \in F(O_{pq})} X_{j,s} = 1 \qquad \forall \text{ external } O_p \tag{7.21}$$

$$\sum_{j \in A^G} \sum_{k \in r} X_{k,j} = \sum_{j \in A^G} \sum_{s \in F(O_{pq})} X_{s,j} \quad \forall O_{pq}, \forall r \in R(O_{pq}) \tag{7.22}$$

$$\sum_{j \in A^S} \sum_{k \in r} X_{j,k} = \sum_{j \in A^S} \sum_{s \in F(O_{pq})} X_{j,s} \quad \forall O_{pq}, \forall r \in R(O_{pq}) \tag{7.23}$$

One of the branches of external OR-Pairs will be scheduled (7.20), (7.21). If the primary OP node in an OR-Pair branch is scheduled, so will the remaining OP Nodes in the same branch (7.22), (7.23).

7.4.2 Replanning

At a point of replanning, the ordered set $C = \{C_1, \ldots, C_l\}$ represents completed tasks. Additional constraints for completed transitions (7.24) (7.25):

$$X_{S,C_1} = 1 \tag{7.24}$$

$$X_{C_i,C_{i+1}} = 1 \quad \forall i = 1, \ldots, l-1 \tag{7.25}$$

Furthermore, the cost matrix, $K_{j,k} \quad j \in A^S \setminus C, k \in A^G \setminus C$, is updated to consider a new starting location and an updated world state. Finally, the cost matrix is updated to consider completed tasks:

$$
\begin{aligned}
&K_{S,C_1} = 0 \\
&K_{C_i,C_{i+1}} = 0 \quad \forall i = 1, \ldots, l-1 \\
&K_{C_l,j} = K_{S,j} \quad \forall j \in A^G \setminus C
\end{aligned}
$$

7.5 Conversion from RTSG to PDDL

7.5.1 PDDL

PDDL is a modelling formalism originating from STRIPS [19] that has evolved from the planning competitions held by The International Conference on Autonomous Planning and Scheduling since 1998. In PDDL, a planning problem is described in terms of objects in the world (e.g., robot, gripper, box and location), an initial state and the desired goal state. The initial state and the goal state are specified as a list of facts. A fact is related to a set of objects and it is defined with a binary predicate (e.g., gripper is holding box). Actions (e.g., place box on location) can be applied to change facts. Actions have parameters (e.g., robot, gripper, box, location). They also have preconditions as a binary function of predicates (e.g., gripper is holding box and robot is at location). If the preconditions hold for some set of parameters, an action can be performed that will change the facts according to the action's list of effect predicates applied to the parameters (e.g., gripper is not holding box, the box is on location). Finding a plan is about finding a sequence of actions applied to the objects that step-by-step will change the facts from the initial state to the goal state. A simple objective for finding an *optimal* plan is to minimize the number of actions. The PDDL2.1 specification [5] introduced syntax for temporal and numerical planning. It also includes *metrics* that allows for specifying an objective. With these language extensions, it is possible to convert an RTSG model into a PDDL scheduling problem.

7.5.2 Conversion from RTSG to PDDL

With respect to RTSG, we identify the natural PDDL objects as the RTSG nodes. The reason for converting nodes to PDDL objects is that nodes have several relations and properties that can be defined as predicates or numerical functions, e.g., the edges that connect them or the transition cost between them. Two types of actions are needed. The first type is to run a task and

the second type is to fire a transition for a logical node. Running a task has a duration while firing a transition is instant. The purpose of transition actions is to guide the scheduling of tasks according to the constraints imposed by the RTSG. The occurrences of transition actions in a planned action sequence do not correspond to real robot actions. However, they can be used to improve the visualization of a runtime execution state and progress:

In Figure 7.3, the active task is orange, completed tasks are green while non-started tasks are grey. Completed logical nodes are light green while the remaining are white. This illustrates the execution progress as a gradual green propagation of the graph that will follow the outgoing edges of completed tasks/transitions.

Listing 7.1: PDDL domain

```
(define (domain RTSG)
    (:types
      node - object
      task logical andjoin2 - node
      startcond goalcond robtask - task
      andfork orfork orjoin - logical
      nofork - orfork)
    (:predicates
      (edge ?n1 ?n2 - node)
      (fired ?n - node)
      (latest-completed ?t - task)
      (andjoin2-inputs ?n1 ?n2 - node)
      (orfork-branch ?orf - orfork ?to - node)
      (branch-not-selected ?orf - orfork)
      (not-locked ?from ?to - task))
    (:functions
      (cost ?from ?to - task))
    (:durative-action RUN-TASK
      :parameters (?this ?prev - task ?input - node ?orf - orfork)
      :duration (= ?duration (cost ?prev ?this))
      :condition (and
          (at start (latest-completed ?prev))
          (at start (edge ?input ?this))
          (at start (fired ?input))
          (at start (orfork-branch ?orf ?this))
          (at start (branch-not-selected ?orf))
          (at start (not-locked ?prev ?this)))
      :effect (and
          (at start (not(latest-completed ?prev)))
          (at start (not(branch-not-selected ?orf)))
          (at end (latest-completed ?this))
          (at end (fired ?this))))
    (:durative-action FIRE-LOGICAL
      :parameters (?this - logical ?input - node ?orf - orfork)
      :duration (= ?duration 0)
      :condition (and
          (at start (edge ?input ?this))
          (at start (fired ?input))
          (at start (orfork-branch ?orf ?this))
          (at start (branch-not-selected ?orf)))
      :effect (and
          (at start (not(branch-not-selected ?orf)))
          (at end (fired ?this))))
    (:durative-action FIRE-ANDJOIN2
      :parameters (?this - andjoin2 ?input1 ?input2 - node ?orf -
          orfork)
      :duration (= ?duration 0)
      :condition (and
          (at start (edge ?input1 ?this))
          (at start (edge ?input2 ?this))
          (at start (fired ?input1))
          (at start (fired ?input2))
          (at start (andjoin2-inputs ?input1 ?input2))
          (at start (orfork-branch ?orf ?this))
          (at start (branch-not-selected ?orf)))
      :effect (and
          (at start (not(branch-not-selected ?orf)))
          (at end (fired ?this)))))
```

Listing 7.2: PDDL problem

```
(define (problem RTSG-config)
(:domain RTSG)
    (:objects
        s - startcond
        g - goalcond
        af1 - andfork
        aj1 aj2 - andjoin2
        of1 - orfork
        oj1 - orjoin
        t1 t2 t3 t4 t5 t6 - robtask
        nfs nfg ... nft4 - nofork   ; Dummy objects
    )
    (:init
        (fired s)
        (latest-completed s)
        (edge s af1)
        (edge af1 t1) ... (edge aj2 g)
        (not-locked s t1) ... (not-locked t6 g)
        (andjoin2-inputs t1 t2)
        (andjoin2-inputs oj1 t4)
        (orfork-branch of1 t5)
        (orfork-branch of1 t6)
        (branch-not-selected of1)
        (orfork-branch nfs s)       ; Dummy fact
        ...                         ; ...
        (orfork-branch nfg g)       ; Dummy fact
        (branch-not-selected nfs)   ; Dummy fact
        ...                         ; ...
        (branch-not-selected nft4)   ; Dummy fact
        (= (cost s t1) 100) ... (= (cost t6 g) 100))
    (:goal (fired g))
    (:metric minimize total-time)
)
```

Converted domain and problem files for the RTSG model in Figure 7.1 are shown in Listings 7.1 and 7.2. The syntax used from PDDL2.1 has been reduced to enable the POPF2 planner [20] that is supported by ROSPlan [21] making it an attractive choice for robotics research. POPF2 does not support some of the PDDL2.1 requirements, among them negative preconditions, disjunctive preconditions and conditional effects. This adds some complexity to the conversion by a need to use antonym predicates (e.g., “not-locked" instead of “locked"), dummy objects, redundant facts and redundant actions.

In the following, a walkthrough is made through the different sections of the converted PDDL domain and problem files in Listings 7.1 and 7.2. The contents of the domain sections are mostly fixed and independent of the RTSG model with only a few stated exceptions for AND-Join nodes. The problem sections are populated from the RTSG model as specified in the walkthrough. With this specification, the conversion from a general RTSG model to PDDL2.1 can be fully automated.

Domain sections:

Types

RTSG node types are arranged as different types in a hierarchy: A *node* is a PDDL *object*. A *task* is a *node* that affects the cost of the plan. There are three types of *tasks*: *robottask*, *startcond* and *goalcond*. The remaining types are used to define logical nodes of different types. The *nofork* type is used to create dummy objects that support the handling of alternative task sequences. The *andjoin2* type is used to create AND-Joins having two incoming edges. If the RTSG model has AND-Joins with more incoming edges, additional types are needed to cover them as well, e.g., *andjoin3*, *andjoin4* etc.

Predicates

The edges between two RTSG nodes are indicated with an *edge* predicate. A completed RTSG node is indicated with a *fired* predicate. The *latest-completed* predicate indicates if a task is the latest completed task. A group of all X nodes having an outgoing edge to the very same AND-Join are indicated with an *andjoinX-inputs* predicate. Nodes having an incoming edge from a specific OR-Fork are indicated with an *orfork-branch* predicate. The same predicate is also used to indicate other nodes, but these are created with a *nofork* in the problem sections. The *branch-not-selected* predicate indicates that there has been no selection of an alternative branch for an OR-Fork. Finally, the *not-locked* predicate indicates that a transition is possible between two tasks.

Functions

A *cost* function indicates the cost, as a numerical value, required to perform a task after finishing a previous task.

Durative actions for running tasks

There is one action that runs tasks. The parameters indicate which task to run, the previous task, the node that is connected to the incoming edge and an associated orfork (a dummy or a real). The action's duration time is set to the cost to run the task after the previous task. The preconditions require that an action already has been run for the node connected to the incoming edge. It also requires that a transition from the previous task is allowed (*not-locked*). The primary effect of the action is to indicate that the action for the task has run (*fired*). The combination of preconditions and effects avoids concurrent tasks and prevents the scheduling of tasks in more than one alternative OR-Pair branch. Note that $goalcond$ also is a $task$ and running it will reach the goal state, e.g., by moving to a certain location.

Durative actions for firing transitions

The remaining actions are used to fire transitions for logical nodes. The parameters indicate for which logical node the transition will occur, a node that is connected to an incoming edge and an associated orfork. The action's duration time is always zero. The preconditions require that an action already has been run for the node connected to the incoming edge. The primary effect of the action is to indicate that the action for the logical node has run. The combination of preconditions and effects prevents the scheduling of tasks in more than one alternative OR-Pair branch.

Similar but separate actions are used to fire transitions for AND-Join nodes. One such action is needed for every used number of incoming edges on AND-Join nodes in the RTSG model. There is only one difference between these actions: The preconditions require that actions have been run for *all* nodes connected to the incoming edges.

Problem sections:

Objects

One *node* (of corresponding *type*) is created for each node in the RTSG model except for AddLocks and RemoveLocks. One dummy *nofork* object is created

for each of these *nodes* that do not have an incoming edge from an OR-Fork node.

Init

The start node is indicated as *fired* and it is also indicated to be the *latest-completed task*. *Edge* facts are created between the *nodes* according to the RTSG model, but where AddLock and RemoveLock nodes are bypassed. *No-lock* facts are defined for all possible transitions between *tasks* with respect to the precedence constraints and Lock-Pair constraints imposed by the RTSG model. For all AND-Joins in the RTSG model, an *andjoinX-inputs* fact is created indicating all *nodes* that are connected to the incoming *edges*. For all *nodes* having an incoming *edge* from an *orfork*, an *orfork-branch* fact is created indicating this OR-Fork, and for all remaining *nodes*, an *orfork-branch* fact is created indicating the corresponding *nofork*. A *branch-not-selected* fact is created for all OR-Forks, indicating if an outgoing branch has not yet been selected. Finally, numerical *cost* facts are created to specify the cost of all possible transitions between *tasks*.

Goal and metric

The goal is to reach the condition that the goal node has been *fired*. The metric indicates that an optimal plan should minimize the total duration of the plan (*total-time*).

7.5.3 Replanning

In a replanning scenario, some modifications are required for the init section of the problem Listing 7.2. A *fired* predicate is added for all completed tasks and for all logical nodes that precede completed tasks. The *latest-completed* predicate is removed for the start node and added for the latest completed task. Transition costs from the latest completed task are updated to account for a new start location of the robot. Potential obstacles may affect some transition costs, e.g., if the robot needs to move another way between two tasks. If there are completed tasks in one OR-Fork branch, scheduling of tasks in alternative branches are avoided by *not* creating a *branch-not-selected* predicate for the corresponding *orfork*.

7.6 Results

7.6.1 Experimental setup

The targeted application is a mobile robot operating in a warehouse for picking customer orders in the form of kits, i.e., boxes filled with specified objects. The robot moves around the warehouse shelves and performs robot tasks as specified by an RTSG model. In the graph, a robot task represents an action where a specific object is handled at a specific location. The RTSG models for three different use cases (A, B and C) are shown in Figures 7.2, 7.3 and 7.4. All use cases start with the fetching of 2 empty kit boxes and allow them to be filled in parallel. In use cases A and B, the kit boxes are filled in two layers separated by an interlayer. Use case B has more precedence constraints than A while use case C has quite few precedence constraints.

Gazebo [22] was used to set up a simulated mobile robot in a simple warehouse world having shelves of different shapes, see Figure 7.5. Dijkstra's algorithm was used to generate two-dimensional collision-free paths between handling locations at the different shelves. The path lengths were used to define transition costs.

The experiments were run with Ubuntu 18.04.5 on an Intel i5-4570 quad-core processor with 7.6 GB RAM.

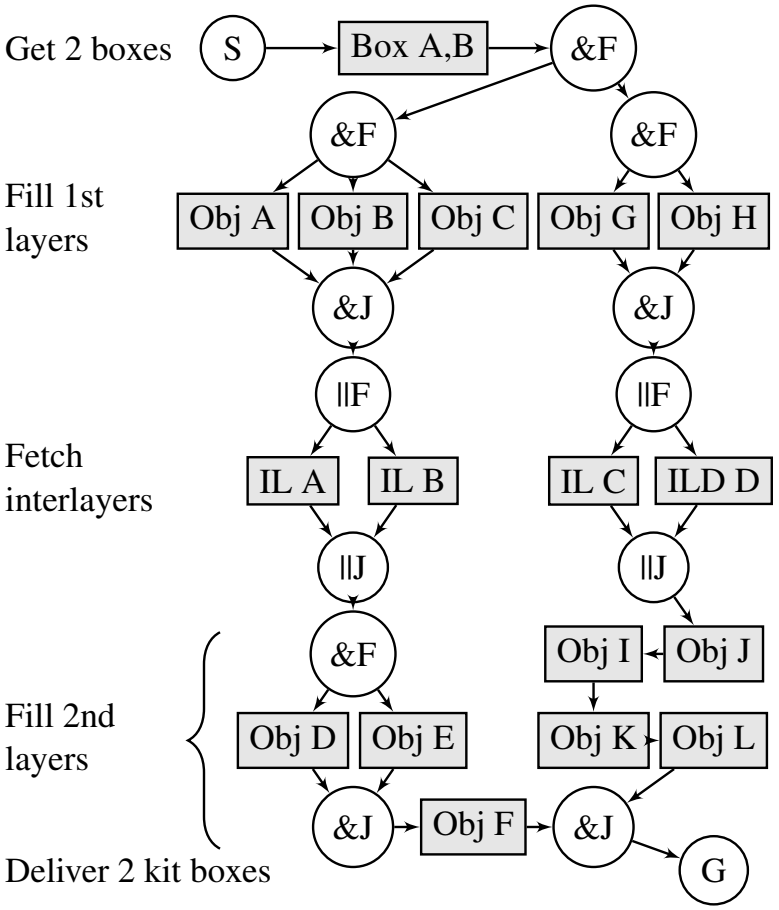

Figure 7.2: RTSG model for use case A.

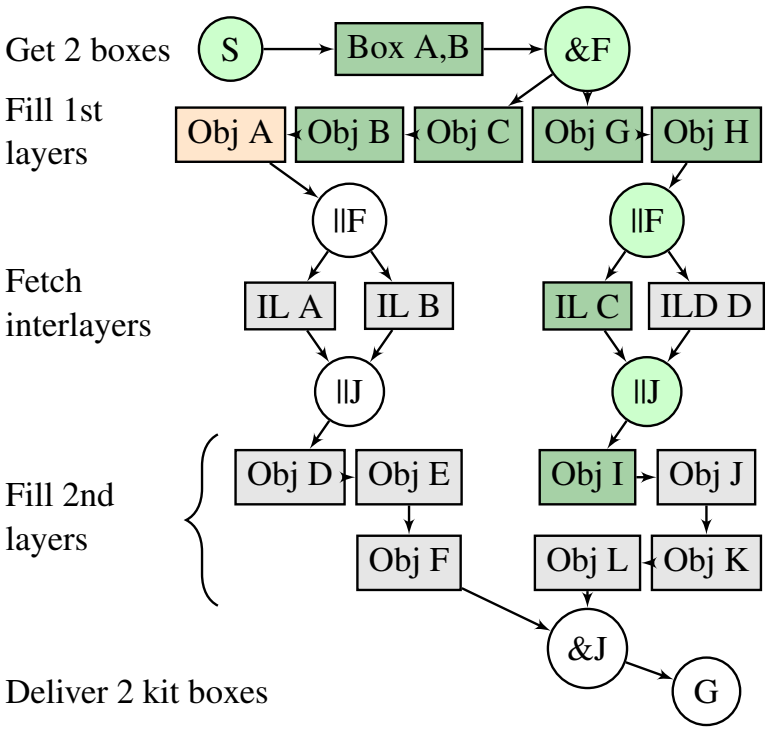

Figure 7.3: RTSG model for use case B. The colouring of the graph nodes is discussed in subsection 7.5.2

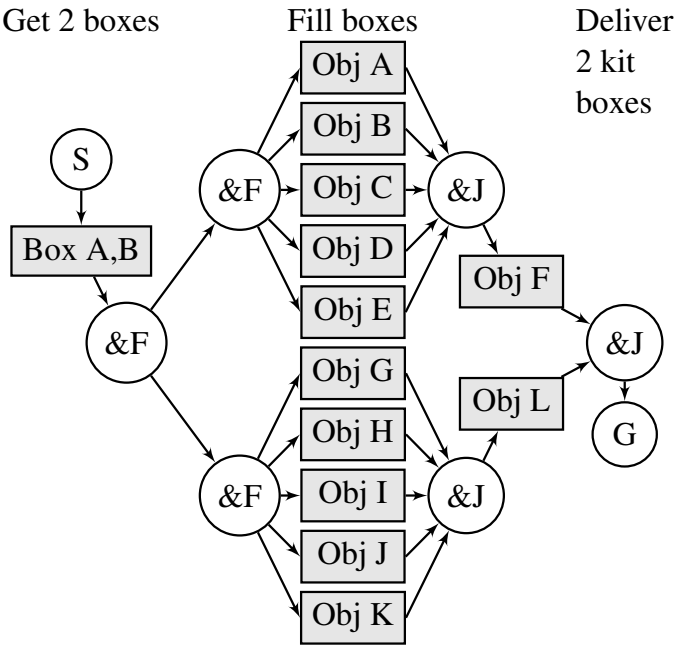

Figure 7.4: RTSG model for use case C.

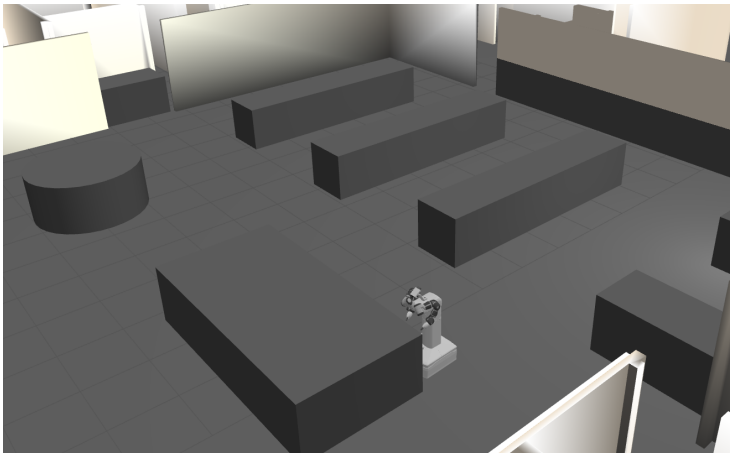

Figure 7.5: Simple warehouse world.

7.6.2 Experimental results

The use cases were tested with three different planners in a comparative study where all problem formulations were automatically generated from the RTSG models. Gurobi Optimizer [23], hereby referred to as MILP, was used with the MILP problem formulation. POPF2 and Temporal Fast Downward (TFD) [24] were used with the PDDL problem formulation. Each use case was run 100 times. For each run, the location of the objects to be handled by the robot tasks was randomized among 52 fixed locations at the different shelves. The common start- and goal location, i.e., the delivery station, was fixed. Comparisons of the planners' achievements of objective values and planning times are displayed in Figure 7.6. MILP and TFD indicated optimal solutions and reached the same objective value for all runs and the very same sequence for 89% of the runs. This suggests an equivalence of the PDDL conversion and the corresponding MILP problem formulation. MILP solved C within a few seconds while needing several minutes for A and B. The reason for this performance difference is presumably a fast-growing amount of precedence and non-cyclic constraints needed for A and B compared to the less constrained C. On the other hand, TFD needed a minute to solve use case C while solving A and B within a few seconds. POPF2 found valid but non-optimal solutions within 200 ms for A and B and within 2 seconds for C. The average cost increase for POPF2 was 15%, 14% and 11% for A, B and C respectively. The fast planning time for POPF2 can be attractive if a less optimized plan is acceptable.

In these experiments, the transition costs are based on path lengths but the optimization goal is often to minimize the makespan. To verify the correlation of path lengths and makespan an additional experiment was performed: To this end, a selection of ten MILP generated task sequences for use case B was made. The selection included the lowest and the highest costs and intermediate cost values with a fairly uniform distribution within this interval. These cycles were simulated 20 times and the cost vs makespan is given in Figure 7.7. ROS Navigation Stack [25] was used to navigate the robot with Adaptive Monte Carlo Localization for localization and Timed Elastic Bands for trajectory generation. The experiment indicates a linear correlation between cost and makespan. However, the non-monotonic part of the curve occurring between the two highest costs also confirms that there is an uncertainty imposed by a simplified cost model when applied to a more realistic setting.

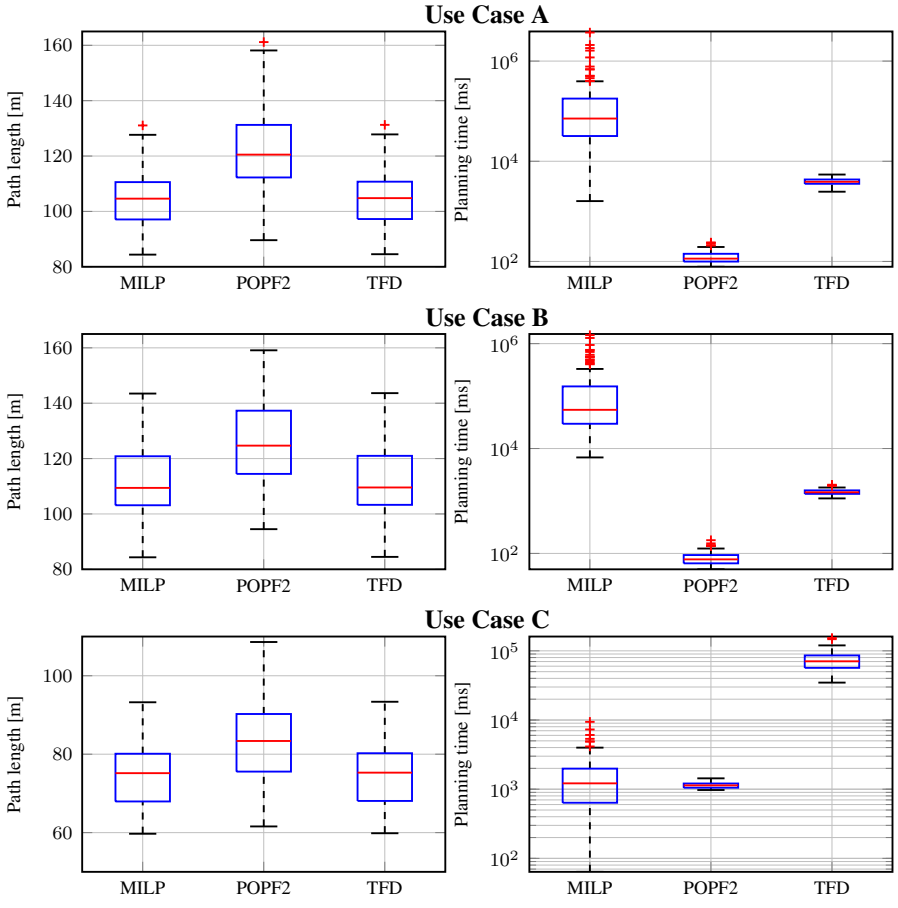

Figure 7.6: Use cases results.

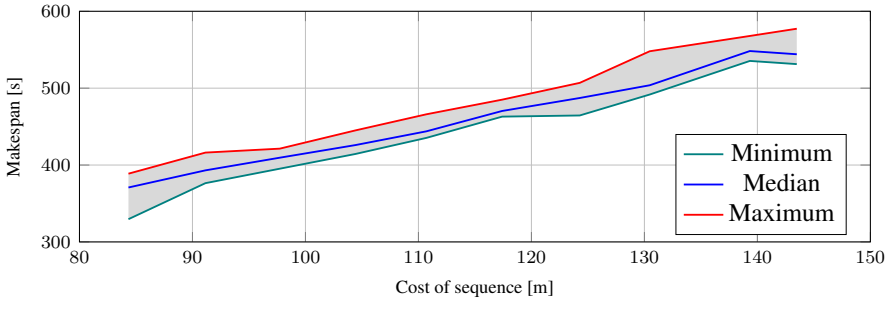

Figure 7.7: Sequence cost vs simulated makespan for use case B. The grey area highlights the distribution of the obtained makespans in simulation.

7.7 Conclusion and future work

We have presented Robot Task Scheduling Graph, a novel task modelling formalism. Descriptions are provided on how to automatically convert an RTSG model to a scheduling problem for MILP solvers as well as for PDDL planners. An experimental comparison of two PDDL planners and one MILP solver suggests the converted problem formulations for MILP and PDDL are equivalent. These experiments also indicate that MILP solvers are more efficient than PDDL planners for an RTSG model with fewer precedence constraints while the opposite hold for an RTSG model with more precedence constraints. Mobile robot simulations of scheduled task sequences confirm the validity of estimating transition costs from 2-dimensional path lengths.

Future work may cover an extension to modelling and scheduling in the context of multi-robot applications and continuous applications. Another line of work may cover efficient modelling and handling of disturbance behaviours.

Bibliography

[1] Erez Karpas and Daniele Magazzeni. Automated planning for robotics. *Annual Review of Control, Robotics, and Autonomous Systems*, 3(1):417–439, 2020.

[2] Branko Miloradović, Baran Çürüklü, Mikael Ekström, and Alessandro Vittorio Papadopoulos. GMP: A genetic mission planner for heterogeneous multi-robot system applications. *IEEE Trans. on Cybernetics*, 2021.

[3] D. McDermott, M. Ghallab, A. Howe, Craig A. Knoblock, A. Ram, M. Veloso, Daniel S. Weld, and D. Wilkins. PDDL – the planning domain definition language. 1998.

[4] H. Nakawala, P. J. S. Goncalves, P. Fiorini, G. Ferringo, and E. D. Momi. Approaches for action sequence representation in robotics: A review. In *IROS*, pages 5666–5671, 2018.

[5] M. Fox and D. Long. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *ArXiv*, abs/1106.4561, 2003.

[6] Dana Nau, Tsz-Chiu Au, Okhtay Ilghami, Ugur Kuter, J William Murdock, Dan Wu, and Fusun Yaman. Shop2: An htn planning system. *J. Artif. Intell. Res. (JAIR)*, 20:379–404, 2003.

[7] Tobias Rittweiler. Cram design & implementation of a reactive plan language. 2010.

[8] Drew Mcdermott. A reactive plan language. 1993.

[9] Matthew Crosby, Francesco Rovida, Mikkel Rath Pedersen, Ronald P. A. Petrick, and Volker Krüger. Planning for robots with skills. In *Workshop on Planning and Robotics (PlanRob)*, pages 49–57, 2016.

[10] David Weintrop, Afsoon Afzal, Jean Salac, Patrick Francis, Boyang Li, David Shepherd, and Diana Franklin. Evaluating CoBlox: A comparative study of robotics programming environments for adult novices. In *Conf. on Human Factors in Computing Systems (CHI)*, pages 1–1, 2018.

[11] Jonathan Bohren and Steve Cousins. The smach high-level executive. *Robotics & Automation Magazine, IEEE*, 17:18–20, 2011.

[12] Vittorio Ziparo, Luca Iocchi, Pedro Lima, Daniele Nardi, and Pier Francesco Palamara. Petri net plans: A framework for collaboration and coordination in multi-robot systems. *Autonomous Agents and Multi-Agent Systems*, 23:344–383, 2011.

[13] Hugo Costelha and Pedro Lima. Robot task plan representation by Petri nets: Modelling, identification, analysis and execution. *Autonomous Robots*, 33, 2012.

[14] Alejandro Marzinotto, Michele Colledanchise, Christian Smith, and Petter Ögren. Towards a unified behavior trees framework for robot control. In *IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 5420–5427, 2014.

[15] Izabela Nielsen, Quang Vinh Dang, Grzegorz Bocewicz, and Zbigniew Banaszak. A methodology for implementation of mobile robot in adaptive manufacturing environments. *Journal of Intelligent Manufacturing*, 28:1171–1188, 2015.

[16] Luiz Mello and Arthur Sanderson. Representations of mechanical assembly sequences. *Robotics and Automation, IEEE Transactions on*, 7(2):211–227, 1991.

[17] Xinwen Niu, Han Ding, and Youlun Xiong. A hierarchical approach to generating precedence graphs for assembly planning. *Int. Journal of Machine Tools and Manufacture*, 43(14):1473–1486, 2003.

[18] A. Salmi, Pierre David, Joshua Summers, and Blanco Eric. A modelling language for assembly sequences representation, scheduling and analyses. *Int. Journal of Production Research*, 52:3986–4006, 2014.

[19] Richard E. Fikes and Nils J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3):189 – 208, 1971.

[20] Andrew Coles, Amanda Coles, Allan Clark, and Stephen Gilmore. Cost-sensitive concurrent planning under duration uncertainty for service-level agreements. 2011.

[21] Gerard Canal and Michael Cashmore. ROSPlan: AI planning and robotics. 2019.

[22] N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, volume 3, pages 2149–2154 vol.3, 2004.

[23] LLC Gurobi Optimization. Gurobi optimizer reference manual, 2021.

[24] Patrick Eyerich, Robert Mattmüller, and Gabriele Röger. Using the context-enhanced additive heuristic for temporal and numeric planning. In *ICAPS*, 2009.

[25] Rodrigo Longhi Guimarães, André Schneider de Oliveira, João Alberto Fabro, Thiago Becker, and Vinícius Amilgar Brenner. *ROS Navigation: Concepts and Tutorial*, pages 121–160. 2016.

Chapter 8

Paper B
Task Roadmaps - Speeding up Task Replanning

Anders Lager, Giacomo Spampinato, Alessandro V. Papadopoulos and Thomas Nolte. In Frontiers in Robotics and AI, section Robotic Control Systems, 2022.

Abstract

Modern industrial robots are increasingly deployed in dynamic environments, where unpredictable events are expected to impact the robot's operation. Under these conditions, runtime task replanning is required to avoid failures and unnecessary stops, while keeping up productivity. Task replanning is a long-sighted complement to path replanning, which is mostly concerned with avoiding unexpected obstacles that can lead to potentially unsafe situations. This paper focuses on task replanning as a way to dynamically adjust the robot behaviour to the continuously evolving environment in which it is deployed. Analogously to probabilistic roadmaps used in path planning, we propose the concept of *Task roadmaps* as a method to replan tasks by leveraging an offline generated search space. A graph-based model of the robot application is converted to a task scheduling problem to be solved by a proposed Branch and Bound (B&B) approach and two benchmark approaches: Mixed Integer Linear Programming (MILP) and Planning Domain Definition Language (PDDL). The B&B approach is proposed to compute the task roadmap, which is then reused to replan for unforeseeable events. The optimality and efficiency of this replanning approach are demonstrated in a simulation-based experiment with a mobile manipulator in a kitting application. In this study, the proposed B&B Task Roadmap replanning approach is significantly faster than a MILP solver and a PDDL based planner.

8.1 Introduction

With the introduction of stationary and mobile robots in collaborative settings [1], robots need a more sophisticated autonomous behaviour to handle an increasingly dynamic environment both safely and efficiently. Robots must be capable of dealing with such uncertainty at runtime, without impacting too much on their expected productivity. The path planning problem has been extensively discussed in the literature [2, 3, 4, 5, 6] as one important aspect to be able to guarantee a safe operation of the robot, and avoid collision with humans, robots, or other unexpected objects present in the environment. However, an efficient feasible path may not be easy to find at runtime, e.g., due to physical constraints of the environment, and the robot may need to stop waiting for the path to be cleared or make an extended detour. Whenever an unforeseeable event is perceived, e.g., the robot path is not cleared, or a task exception occurs, a task replanner can re-assign the sequence of tasks to the robot to keep its productivity high [7].

In this paper, we propose a *task planning* approach for industrial robots and service robots, called *Task Roadmaps* (TRM), that can be used for replanning the robot's task allocation at runtime. The approach is inspired by Probabilistic Roadmaps [6], as it uses a similar idea to speed up the replanning of tasks at runtime. An initial plan may be generated offline while replanning is an online activity that has a direct impact on productivity, as well as the perceived reactive responsiveness of a robot.

In this work, the TRM approach is applied to a robot application modelled in the form of a Robot Task Scheduling Graph (RTSG). RTSG is an intuitive graph-based task modelling formalism for robot applications in dynamic environments that was proposed in our previous work [8]. An RTSG model can be converted to a mathematical representation of the related task scheduling problem as a Mixed Integer Linear Programming (MILP) problem. The solution of the MILP problem provides the execution sequence of tasks to complete the mission with a minimized makespan. Additionally, an RTSG model can be converted to a domain and problem description in the Planning Domain Definition Language (PDDL), allowing for the scheduling problem to be solved by planners compatible with this format.

Unfortunately, the MILP formulation is an NP-hard problem [9, 10], and computing a solution can be time-consuming. Compared to MILP solvers, PDDL based planners tend to be more efficient for RTSG models with more constraints but less efficient for models with fewer constraints [8].

In this paper, we propose the concept of TRM and present a Branch and Bound (B&B) algorithm to solve the very same scheduling problem described

above while generating a reusable planning space (a task roadmap). Whenever replanning is needed, the B&B algorithm can leverage the planning space, which will speed up the replanning time considerably. This usage scenario of the algorithm is referred to as B&B-TRM.

In a simulation-based experimental study, we compare the replanning performance for a MILP solver, a PDDL planner, B&B, and B&B-TRM in a kitting application with a mobile manipulator. The experiments show a significant reduction of task replanning time with B&B-TRM compared to the other approaches, while providing equivalent solutions in terms of cost.

The remainder of this paper is organized as follows. Section 8.2 presents related works, Section 8.3 gives an introduction to the task modelling formalism, RTSG, and the general scheduling problem. Section 8.4 details the scheduling problem formulation as a MILP, Section 8.5 shows how RTSG can be converted to PDDL. Section 8.6 introduces Task Roadmaps, exemplified with a B&B scheduling algorithm for RTSG models. Section 8.7 presents the experimental results, while Section 8.8 concludes the paper.

8.2 Related work

Some replanning approaches make a new plan from scratch when an unexpected condition occurs, e.g., see the work by [11]. This is a solid approach for high-quality plans but often at a high price of computational time for large problem instances. Moreover, our approach essentially makes a replanning from scratch but in addition, it leverages the search space generated to find the initial plan, thereby reducing the planning time.

Other approaches try to reuse the initially generated *plan*, modifying parts of it to adapt to unexpectedly changed or more refined conditions. The purpose can be to locally optimize the initially planned sequence, e.g., with rule-based transformational planning [12] or by rearranging subgoals at runtime using Hierarchical Planning [13]. The purpose can also be to repair a plan, e.g., by making a rule-based rearrangement of operations [14]. This way of replanning can be more simple and efficient than replanning from scratch, but the quality of a modified plan may become less optimal or invalid [15]. A sophisticated variant of this approach creates an adaptable and partially ordered initial plan, having an online algorithm generating a set of completely ordered plans and dispatching the one with the best chance for success given the current state [16].

The Traveling Salesperson Problem (TSP) with Precedence Constraints (PCs) with a fixed starting- and endpoint is a special case of the scheduling problem for RTSG models targeted in this work. RTSG models additionally

include alternative sequences and interrupt locks. One example of a TSP-PC problem instance is TSP with pickup and delivery [17]. Recently, a dynamic programming approach to solve TSP-PC dating back to 1979 was revisited [18]. In this approach, which is akin to our proposed B&B (that uses a breadth-first and forward search approach), the algorithm starts from an empty set of nodes and uses an expansion operator to select the order-theoretic minimal of the remaining nodes in every iteration.

8.3 Task modelling formalism and scheduling problem formulation

In this section, RTSG, the task modelling formalism used in this paper, is presented. This is followed by a description of the task scheduling problem and related assumptions.

8.3.1 Robot Task Scheduling Graph

An RTSG is a directed acyclic graph, as exemplified in Figure 8.1. The graph is composed, e.g. by a domain expert, to specify the variability of a task sequence from a start node (S) to a goal node (G) that will achieve a higher-level goal, e.g., to fetch and deliver a selection of different objects from a warehouse. S has one outgoing edge and G has one incoming edge. Intermediate nodes in rectangular form represent tasks that may be executed in a scheduled sequence to reach the goal. Tasks have one incoming and one outgoing edge and represent robot actions at different locations in the environment, e.g. the fetching of an object. Edges and paths (of edges) represent precedence constraints. For example, if there is a directed path from task A to task B, then task A must precede task B in any schedule where both A and B are present. The remaining nodes, with a circular shape, are logical nodes that guide the variability of the task sequence. These are intuitively described in the next subsections 8.3.2, 8.3.3 and 8.3.4.

8.3.2 AND-pairs

An *AND-pair* is an AND-Join node (&J) and a corresponding preceding AND-Fork node (&F). The AND-Fork node has a single incoming edge and multiple outgoing edges, while the AND-Join node has multiple incoming edges and a single outgoing edge. AND-pairs split a single branch into parallel branches at the AND-Fork node (&F) and rejoin them at the AND-Join node (&J).

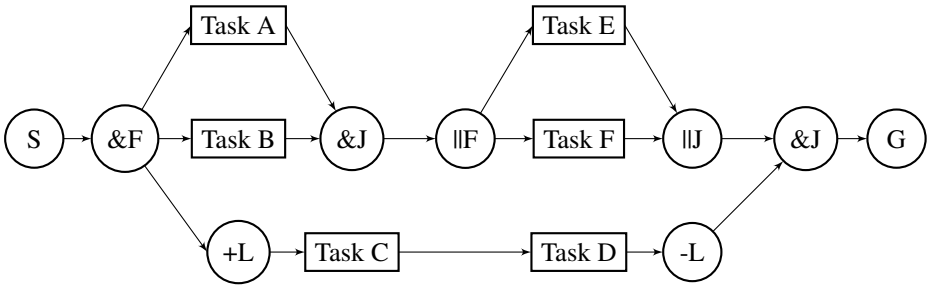

Figure 8.1: Robot Task Scheduling Graph.

The function of AND-pairs is to indicate more complex precedence constraints by being able to fork and rejoin branches. The mutual scheduling order of tasks in different parallel branches is variable since there is no directed path between them. Additionally, tasks in these branches must be scheduled before any task succeeding the AND-Join node.

8.3.3 OR-pairs

An *OR-pair* is an OR-Fork node and a corresponding succeeding OR-Join node. The OR-Fork node has a single incoming edge and multiple outgoing edges while the OR-Join node has multiple incoming edges and a single outgoing edge. An OR-pair contains *alternative* branches of tasks, where at most one of them will be scheduled. If an OR-pair is contained by another OR-pair, it is said to be *internal*, otherwise, it is *external*. For an external OR-pair, one of its contained branches will be scheduled. For an internal OR-pair, one of its branches will be scheduled if the OR-pair is a part of a scheduled branch.

8.3.4 Lock-pairs

A *Lock-pair* is an +L node and a corresponding succeeding -L node. These nodes have a single incoming edge and a single outgoing edge. The sub-graph between a Lock-pair must be scheduled uninterrupted by externally located tasks.

8.3.5 The task scheduling problem

The problem to solve is to generate a sequence of tasks that minimises the cost to achieve the goal in a way that satisfies the constraints of the given RTSG model. Apart from the sequencing of tasks, a selection of alternative tasks

is generally included in the scheduling problem. The cost to be minimized includes routing costs implicated by the task sequence selection.

It is a deterministic, single robot scheduling problem with non-concurrent tasks, where the task allocation type is a time extended assignment [19]. The state is fully observable at planning/replanning.

Targeted replanning scenarios handle unexpected states that are blocking or delaying the progress of task execution. They include the considering of obstacles that are obstructing the execution of the initially planned path/route or blocking the access of planned task locations. Additionally, they may include unexpected circumstances affecting the time to execute a task, e.g., when the robot needs to pick an object from a shelf location, and there are no objects in the box; the box will be eventually refilled, e.g. by a human but the completion of the task is affected by an unexpected duration. As a consequence of the replanning scenario, the cost for the initially planned sequence may increase and in the extreme case make it impracticable. The transition cost may become changed between many tasks and not only affect the currently running task or its successor. After rescheduling, the order of tasks to be executed may be changed or remain. Additionally, alternative tasks may become replaced.

Replanning for an unexpected adding of sub-goals, requiring a structural modification/extension of the RTSG model, is not investigated in this work. However, the removal of modelled sub-goals may be handled, e.g. by penalizing the cost for related tasks or with a selective pruning by the proposed B&B algorithm.

The computational complexity of the RTSG scheduling problem depends on the structure of the graph. As a simplistic example, the RTSG can be used to model two alternative branches of totally-ordered sequences of actions encapsulated by an OR-pair. The solution to this problem can be solved in polynomial time. On the other hand, RTSG can also be used to model a Traveling Sales Person problem (TSP). This is a problem that is known to be NP-hard [20] indicating the general RTSG problem is at least NP-hard.

A planner supporting temporal PDDL may be used to address problems of harder complexity classes than NP, e.g. a temporal plan *existence* problem may be EXPSPACE-complete [21]. Rintanen shows that a significant fragment of temporal PDDL planning problems can be reduced in polynomial time to classical planning with a complexity class of PSPACE. The requirements for this reduction include no overlapping of the same action and state-independent action duration. However, RTSG planning problems have state-dependent action durations. Classical domain-independent planning languages do not support state-dependent cost. However, it might be possible to reduce the problem into a classical problem by generating a manifold of fixed-cost actions [22]. The

modelling approach in this work is not based on a standardized format, e.g. classical PDDL, which is too limited for the approach. Instead, it uses the native SAS+ format [23]. Despite the improvements suggested by Geißer et al., these conversions will in the worst case grow exponentially. The combination of these drawbacks for the usage of classical planners motivated the selection of a temporal PDDL planner as one of the benchmarks in this study.

8.4 MILP representation

The task scheduling problem can be formulated as a Mixed Integer Linear Programming (MILP) problem where the decision variables and the constraints are derived from the RTSG model. The optimization objective is to minimize a cost function, e.g., in the form of a total completion time. The MILP problem formulation is detailed in this section.

8.4.1 Notation

A is the set of all task nodes in the RTSG, S is the start node and G is the goal node. We indicate with $A^S = A \cup S$, with $A^G = A \cup G$, and with $\tilde{A} = A \cup S \cup G$.
The set $O \subseteq A$, is the set of all task nodes contained by OR-pairs, thus indicating alternative tasks that may, or may not be a part of a valid task sequence. The notation $j \prec k$ where $j, k \in \tilde{A}$ indicates that task j must be scheduled before task k. The relation $j \prec k$ holds if there is a directed path from j to k in the RTSG.

8.4.2 Problem formulation

The problem that needs to be solved is to select a set of tasks within $\tilde{A}$ and their sequence, starting from S and ending in G, subject to the constraints indicated by the RTSG so that the cost is minimized. Such a problem, can be formulated as an optimization problem, where the decision variables are $X_{j,k} \in \{0, 1\}$, $\forall j, k \in \tilde{A}$, where

$$X_{j,k} = \begin{cases} 1, & \text{if there is a scheduled transition from } j \text{ to } k \\ 0, & \text{otherwise.} \end{cases}$$

Note that $X_{j,j} = 0, \forall j \in \tilde{A}$ since we require that there is always a transition to a different task.

The cost for selecting a transition between task j and task k is indicated with $K_{j,k} \in \mathbb{R}_{\geq 0}$, and it includes the transition cost $\tau_{j,k}$, i.e., the time to move

from the location of task j to the location of task k, and the time α_k that is required to complete the action of task k:

$$K_{j,k} = \tau_{j,k} + \alpha_k. \tag{8.1}$$

The optimization problem aims to minimize the following cost function:

$$J = \sum_{j \in A^S} \sum_{k \in A^G} X_{j,k} K_{j,k} \tag{8.2}$$

8.4.3 General constraints

The minimization problem is subject to the following constraints.

- There is exactly one transition from, and one transition to the non-alternative nodes. However, there is no transition *from* the goal node and no transition *to* the start node:

$$\sum_{k \in A^G} X_{j,k} = 1 \qquad \forall j \in A^S \setminus O \tag{8.3}$$

$$\sum_{k \in A^S} X_{G,k} = 0 \tag{8.4}$$

$$\sum_{j \in A^S} X_{j,k} = 1 \qquad \forall k \in A^G \setminus O \tag{8.5}$$

$$\sum_{j \in A^G} X_{j,S} = 0 \tag{8.6}$$

- No cyclic sub-routes: Let $V \subseteq A$ be any non-empty subset of tasks.

$$\sum_{j \in V} \sum_{k \in V} X_{j,k} \leq |V| - 1 \qquad \forall V \subseteq A, V \neq \emptyset \tag{8.7}$$

- Precedence constraints: Let $D \subseteq \tilde{A}$ be any ordered subset with multiple elements where the last element precedes the first element. D_i is the i-th element of D.

$$\sum_{j=1}^{|D|-1} X_{D_j, D_{j+1}} \leq |D| - 2 \quad \forall D \subseteq \tilde{A},\, |D| \geq 2,\, D_{|D|} \prec D_1. \tag{8.8}$$

8.4.4 Lock-pair definitions and constraints

A Lock-pair contains a set of tasks, $L \subseteq A$. Tasks in L must be scheduled as a group in a sub-sequence that is uninterrupted by other tasks. Two subsets of L are defined: $L^F = \{a \in L \mid b \not\prec a \quad \forall b \in L\}$ specifies the *first tasks* of L while $L^L = \{a \in L \mid a \not\prec b \quad \forall b \in L\}$ specifies the *last tasks* of L.

The constraints associated with Lock-pairs restrict transitions from/to tasks contained by the pair. There can at most be one transition from external tasks to the first tasks, and at most one transition from the last tasks to external tasks:

$$X_{j,k} = 0 \qquad \forall L, \forall j \in L \setminus L^L, \forall k \in A^G \setminus L \tag{8.9}$$

$$X_{j,k} = 0 \qquad \forall L, \forall j \in A^S \setminus L, \forall k \in L \setminus L^F \tag{8.10}$$

$$\sum_{j \in L^L} \sum_{k \in A^G \setminus L} X_{j,k} \leq 1 \quad \forall L \tag{8.11}$$

$$\sum_{j \in A^S \setminus L} \sum_{k \in L^F} X_{j,k} \leq 1 \quad \forall L \tag{8.12}$$

8.4.5 OR-pair definitions and constraints

The OR-pair constraints presented in this section can handle OR-Pairs inside OR-Pairs etc., in a recursive manner. To express the OR-pair constraints we first need to prepare some supporting definitions and operators:

An OR-pair contains a set of *OP nodes*, where an *OP node* is either a task node or an internal OR-pair. In the same way, internal OR-pairs contain OP nodes etc.

Formally, $O_1, \ldots, O_v$ are OP node sets contained by OR-pair $1, \ldots, v$; $O_{p1}, \ldots, O_{pm}$, are OP node sets in branches $1, \ldots, m$ of OR-pair p; $O^T_{pq} = \{a \in O_{pq} \mid a \text{ is a task}\}$ contains task nodes and $O^{OP}_{pq} = \{a \in O_{pq} \mid a \text{ is an internal OR-Pair}\}$ contains internal OR-pairs so that $O_{pk} = O^T_{pq} \cup O^{OP}_{pq}$.

One *primary* OP-node, $P_{pq} \in O_{pq}$ is arbitrarily selected for each OR-pair branch. We define the following operators:

- F is a recursive operator that returns a set of tasks for a given OR-pair branch. The set represents alternative tasks in the branch that shall not be combined:

$$F(O_{pq}) = \begin{cases} \{a\} & \text{if } P_{pq} \text{ is task } a. \\ F(O_{r1}) \cup \ldots \cup F(O_{rm}) & \text{if } P_{pq} \text{ is OR-pair } O_r \end{cases} \tag{8.13}$$

- H returns a set of tasks for a given OR-pair. It represents alternative tasks in the OR-pair that shall not be combined:

$$H(O_p) = F(O_{p1}) \cup F(O_{p2}) \cup \ldots \cup F(O_{pm}) \tag{8.14}$$

- R returns a *set of sets* of tasks for a given OR-pair branch. It represents other sets (than F) with alternative tasks in the branch that shall not be combined:

$$R(O_{pq}) = O_{pq}^T \setminus P_{pq} \cup \bigcup_{i \in O_{pq}^{OP} \setminus P_{pq}} \{H(i)\} \tag{8.15}$$

With these definitions, the OR-pair constraints can be summarized:

- Transitions from and to tasks contained by OR-pairs: There is at most one transition from/to such a task and the number of incoming transitions is the same as the number of outgoing transitions:

$$\sum_{k \in A^G} X_{j,k} \leq 1 \qquad \forall j \in O \tag{8.16}$$

$$\sum_{j \in A^S} X_{j,k} \leq 1 \qquad \forall k \in O \tag{8.17}$$

$$\sum_{k \in A^G} X_{j,k} = \sum_{k \in A^S} X_{k,j} \qquad \forall j \in O \tag{8.18}$$

- Transition from and to tasks in alternative OR-pair branches: For an external OR-pair, exactly one branch will be scheduled. For any OR-pair: If the primary OP node in one branch is scheduled, the remaining OP-nodes in the same branch will also be scheduled.

$$\sum_{j \in A^G} \sum_{q=1}^{m} \sum_{s \in F(O_{pq})} X_{s,j} = 1 \qquad \forall \text{ external } O_p \tag{8.19}$$

$$\sum_{j \in A^S} \sum_{q=1}^{m} \sum_{s \in F(O_{pq})} X_{j,s} = 1 \qquad \forall \text{ external } O_p \tag{8.20}$$

$$\sum_{j \in A^G} \sum_{k \in R'} X_{k,j} = \sum_{j \in A^G} \sum_{s \in F(O_{pq})} X_{s,j} \quad \forall O_{pq}, \forall R' \in R(O_{pq}) \tag{8.21}$$

$$\sum_{j \in A^S} \sum_{k \in R'} X_{j,k} = \sum_{j \in A^S} \sum_{s \in F(O_{pq})} X_{j,s} \quad \forall O_{pq}, \forall R' \in R(O_{pq}) \tag{8.22}$$

8.4.6 Replanning constraints

In dynamic environments, some unforeseeable events may make the initially computed plan not possible to execute. To complete the robot mission, a replanning can be initiated. Such a replanning can, besides changing the order of tasks, exploit other OR-pair branches of the RTSG to successfully complete the mission in an alternative way. On the other hand, if one or more tasks in an OR-pair branch are already completed, the remaining tasks in this branch will also become scheduled in the new plan. For replanning, one needs to introduce additional constraints to account for completed tasks, and for capturing the changed situation that hinders the completion of the initial plan. The set $C = \{C_1, \ldots, C_l\}$ contains the sequence of already completed tasks. The initial transitions for these tasks become additional replanning constraints:

$$X_{S,C_1} = 1 \tag{8.23}$$

$$X_{C_i,C_{i+1}} = 1 \quad \forall i = 1, \ldots, l-1 \tag{8.24}$$

Further, the costs for possible transitions between tasks, $K_{j,k} \quad j \in A^S \backslash C, k \in A^G \setminus C$ are updated to describe the current situation. For example, the costs are affected by unexpected obstacles and the new location of S, i.e., the current location of the robot. Thereafter, the cost of transitions involving completed tasks are initialized:

$$\begin{aligned} K_{S,C_1} &= 0 \\ K_{C_i,C_{i+1}} &= 0 \quad \forall i = 1, \ldots, l-1 \\ K_{C_l,j} &= K_{S,j} \quad \forall j \in A^G \setminus C \end{aligned}$$

8.5 PDDL representation

PDDL is a general domain-independent modelling formalism for setting up planning problems, originating from classical planning [24]. It is used to define planning problems in many areas, also outside the field of robotics. Planning problems that are represented in a PDDL format can be solved by many different planner algorithms, e.g., Temporal Fast Downward (TFD) [25] and POPF2 [26]. Since an RTSG model can be converted to a PDDL planning problem [8], many different planner algorithms are available. One of these, TFD, is used in this work as a benchmark. The reason for selecting TFD was it's ability to find high quality solutions in comparison with other planners/-solvers in our previous work [8]. For compatibility, planners need to support some parts of PDDL2.1 [27] that extends PDDL with syntax for temporal planning.

Listing 8.1: PDDL domain

```
(define (domain RTSG)
    (:types
      node - object
      task logical andjoin2 - node
      startcond goalcond robtask - task
      andfork orfork orjoin - logical
      nofork - orfork)
    (:predicates
      (edge ?n1 ?n2 - node)
      (orfork-branch ?orf - orfork ?to - node)
      (not-locked ?from ?to - task)
      (andjoin2-inputs ?n1 ?n2 - node)
      (fired ?n - node)
      (latest-completed ?t - task)
      (branch-not-selected ?orf - orfork))
    (:functions
      (cost ?from ?to - task))
    (:durative-action RUN-TASK
      :parameters (?this ?prev - task ?input - node ?orf - orfork)
      :duration (= ?duration (cost ?prev ?this))
      :condition (and
          (at start (latest-completed ?prev))
          (at start (edge ?input ?this))
          (at start (fired ?input))
          (at start (orfork-branch ?orf ?this))
          (at start (branch-not-selected ?orf))
          (at start (not-locked ?prev ?this)))
      :effect (and
          (at start (not(latest-completed ?prev)))
          (at start (not(branch-not-selected ?orf)))
          (at end (latest-completed ?this))
          (at end (fired ?this))))
    (:durative-action FIRE-LOGICAL
      :parameters (?this - logical ?input - node ?orf - orfork)
      :duration (= ?duration 0)
      :condition (and
          (at start (edge ?input ?this))
          (at start (fired ?input))
          (at start (orfork-branch ?orf ?this))
          (at start (branch-not-selected ?orf)))
      :effect (and
          (at start (not(branch-not-selected ?orf)))
          (at end (fired ?this))))
    (:durative-action FIRE-ANDJOIN2
      :parameters (?this - andjoin2 ?input1 ?input2 - node ?orf -
          orfork)
      :duration (= ?duration 0)
      :condition (and
          (at start (edge ?input1 ?this))
          (at start (edge ?input2 ?this))
          (at start (fired ?input1))
          (at start (fired ?input2))
          (at start (andjoin2-inputs ?input1 ?input2))
          (at start (orfork-branch ?orf ?this))
          (at start (branch-not-selected ?orf)))
      :effect (and
          (at start (not(branch-not-selected ?orf)))
          (at end (fired ?this)))))
```

Listing 8.2: PDDL problem

```
(define (problem RTSG-config)
(:domain RTSG)
    (:objects
        s - startcond
        g - goalcond
        af1 - andfork
        aj1 aj2 - andjoin2
        of1 - orfork
        oj1 - orjoin
        ta tb tc td te tf - robtask
        nfs nfg ... nftd - nofork   ; Dummy objects
    )
    (:init
        ; Static facts
        (edge s af1)
        (edge af1 ta) ... (edge aj2 g)
        (not-locked s ta) ... (not-locked tf g)
        (andjoin2-inputs ta tb)
        (andjoin2-inputs oj1 td)
        (orfork-branch of1 te)
        (orfork-branch of1 tf)
        (orfork-branch nfs s)       ; Dummy fact
        ...                         ; ...
        (orfork-branch nfg g)       ; Dummy fact
        ; Dynamic facts
        (fired s)
        (latest-completed s)
        (branch-not-selected of1)
        (branch-not-selected nfs)   ; Dummy fact
        ...                         ; ...
        (branch-not-selected nftd)   ; Dummy fact
        (= (cost s ta) 100) ... (= (cost tf g) 100))
    (:goal (fired g))
    (:metric minimize total-time)
)
```

8.5.1 PDDL introduction

A PDDL problem specification is divided into a domain description and a problem description. In the domain description, definitions are made that can be reused for similar planning problems. The most basic definitions include:

- *Types* are used to instantiate different types of objects, e.g., robots, locations, tools, paths, or boxes. The types can be organized with polymorphism, e.g., a tool may be a gripper or a camera.
- *Predicates* are used to instantiate different types of facts, describing relations between objects, e.g., "robot1 is at location1", "location1 and location2 is connected with path1".
- *Actions* are operators that can be applied if a set of preconditions, specified as predicates, hold for a set of object parameters. The application of an action causes a set of effects, also specified as predicates, that will add or remove facts. Action example: Move robot1 from location1 to location2. Preconditions are: 1) "robot at location1" and 2) "location1 and location2 is connected with path". Effects are: 1) "robot is at location2" and *not* "robot is at location1".

A basic PDDL problem description includes:

- Existing objects of different types.
- Facts describing initial relations between objects, i.e., the initial state.
- Facts describing desired or undesired relations between objects, i.e., the goal state.

The task of a *planner algorithm* is to process the domain and problem description and find a sequence of applicable actions operating on the existing objects, that will change the initial state to a state where the goal state is fulfilled. This plan generation process is not investigated here. Instead, details are presented on how to convert the RTSG task modelling formalism into PDDL, thereby enabling plan generation with already existing planner algorithms.

8.5.2 Conversion from RTSG to PDDL

In general, when converting an RTSG model to a PDDL specification, the RTSG nodes become objects and the edges become facts. Two types of PDDL actions are defined:

- Running a task. This is a durative action where the duration is the cost to perform the task.
- Firing a transition of a logical node. The purpose of this action type is to enable the execution of tasks under the constraints imposed by the logical nodes, e.g., precedence constraints and alternative OR-pair branches. They do not correspond to real actions and their duration is zero in order not to affect the cost.

A PDDL domain for RTSG

Listing 8.1 presents a PDDL domain for a general RTSG. The domain is almost independent of the RTSG model that shall be converted.

All object types are derived from the *node* type, and are used to represent nodes in an RTSG model. There are two types of *node*: *task* and *logical*. *task* represents actions or states that affects the cost of a plan: *startcond*, *goalcond* and *robtask*. *logical* represents different types of logical nodes: *andfork*, *orfork* and *orjoin*. Different *andjoin* types are defined for each number of incoming edges that needs to be represented in the RTSG model. For example, *andjoin2* represents an AND-Join with two incoming edges. A *nofork* object is a dummy object that helps in the modelling of alternative branches. Lock-pair nodes are not represented by object types; instead their constraints are modelled with *not-locked* facts.

Among the static predicates, that cannot be affected by any action, *edge* represents an edge between two nodes. *orfork-branch* specifies the outgoing connection from an OR-Fork to another node. *not-locked* specifies if a transition between two tasks is not locked. The other predicates are dynamic and can be created or removed by actions: *fired* specifies if a node (*task* or *logical*) is completed. *latest-completed* specifies if a task is the latest completed task. *branch-not-selected* specifies if an alternative branch for an OR-Fork not has been selected.

There are two types of actions: to run a task and to fire a transition for a logical node. There is one action that runs all tasks, i.e. RUN-TASK. There are a limited number of actions for logical nodes, i.e. FIRE-LOGICAL to run transitions for all *non* AND-Join nodes, FIRE-ANDJOIN2 for transitions of AND-Joins with two incoming edges, FIRE-ANDJOIN3 for 3 incoming edges etc. Additional AND-Join actions must only be defined if they exist in the RTSG model.

Running a task has a duration and the duration is represented by a *cost* function that specifies the cost of performing a task (*to*) after completing another task (*from*). The preconditions require that the node connected to the

incoming edge has been fired. If this connected node is an OR-Fork, it is required that a branch not yet has been selected. It is also required that a transition from the latest completed task not is locked. The effects update the dynamic predicates: The task becomes both *fired* and the *latest-completed*. The *branch-not-selected* is removed for the task's *orfork*.

Fire a transition for a logical node has a zero duration, i.e., free of cost. The preconditions require that the node connected to the incoming edge has been fired. If this connected node is an OR-Fork, it is required that a branch not yet has been selected. The effects update the dynamic predicates: The logical node becomes *fired* and the *branch-not-selected* is removed for the logical node's *orfork*. For an AND-Join action, the preconditions additionally require *every* node connected to the incoming edges to be fired.

A PDDL problem for an RTSG model

Listing 8.2 exemplifies a PDDL problem description at initial planning, that is converted from the RTSG model in Figure 8.1. Some of the data in the conversion are left out, indicated with '...', to get a more compact overview.

The objects and the static facts listed in the PDDL problem are dependent on the structure of the RTSG model.

Objects are defined for all nodes in the RTSG graph except for Lock-pair nodes. Additionally, *nofork* objects are created for all nodes that do not have an incoming edge from an OR-Fork node. *noforks* are dummy objects assisting in the selection of alternative OR-Fork branches.

Static facts are created to represent the structural elements of the RTSG model. The edges are represented with *edge* facts that specify connected nodes. However, edges to/from Lock-Pair nodes are bypassed. *not-locked* facts are created for each valid transition between two tasks, with respect to precedence constraints as well as Lock-pair constraints. For each AND-Join node, an *andjoinX-inputs* fact is created that specifies all nodes connected to its X incoming edges. For each node connected to the outgoing edge of an OR-Fork node, an *orfork-branch* fact is created indicating this *orfork*. For all other nodes, an *orfork-branch* fact is created indicating their *nofork* object.

Dynamic facts are created to represent all possible intermediate states of a task execution sequence. A *fired* fact is added for all completed nodes. For the initial state, as illustrated in Listing 8.2, there is only a single *fired* fact for the start node. At replanning, a *fired* fact is additionally added for all completed tasks and for the logical nodes that precede completed tasks. A single *latest-completed* fact specifies the latest completed task. At initial planning, the start node is always the *latest-completed*, while this may have changed at

replanning. For the initial plan, a *branch-not-selected* fact is created for each *orfork* and nofork. At replanning, these facts are removed if the *orfork* has a *fired* fact or if the node that is connected to the *nofork* has a *fired* fact.

Finally, the numerical cost value for all *not-locked* transitions between tasks is specified. In a replanning scenario, these values may become changed by the current state of the modelled application.

The required goal state is a *fired* fact for the goal object. The objective of the plan, the *metric*, is to minimize the makespan.

8.6 Task Roadmaps

We developed a Branch and Bound (B&B) algorithm, to compute solutions of scheduling problems modelled with RTSG as detailed in Section 8.3. The algorithm makes a breadth-first expansion of a search tree. It is a forward search that is guided from the start node of the RTSG model. When replanning is needed, the algorithm is designed to make a new plan while considering the current conditions, e.g., the location of the robot and the sequence of already completed tasks. Optionally, the search space that was constructed while generating the initial sequence can be reused. This option avoids the need of expanding a new search space from scratch, making the replanning significantly faster with preserved quality of generated plans.

In analogy with Probabilistic Roadmaps [6], the search space is referred to as the Task Roadmap (TRM) that is created in a *learning phase* and used for runtime replanning in a *query phase*. In Table 8.1, we compare the basic characteristics between Probabilistic Roadmaps (PRM) and TRM.

Additionally, a saved Task Roadmap may be leveraged to speed up *initial* planning for an RTSG model if it has the same graph structure as the model used to generate the Task Roadmap.

8.6.1 Learning phase

In the learning phase, a search tree is expanded by the B&B algorithm acting on a Robot Task Scheduling Graph (RTSG) to find an initial task sequence. In the search tree, see Figure 8.2, the initial start condition is represented by the root node S, and other nodes represent sub-sequences. The number of tasks in a sub-sequence corresponds to the distance between the node and the root node. The best sequence is the sequence reaching the leaf G with the lowest cost (indicated in green in the figure).

An important aspect of the scheduling algorithm is the pruning of equivalent nodes. Equivalent nodes have the same distance to the root node and the

Probabilistic Roadmaps	**Task Roadmaps**
Represents a robot configuration space with a graph.	Represents a robot task sequence space with a search tree.
Used for *path* planning with obstacle avoidance.	Used for *task* planning with scheduling constraints.
The graph is built using probabilistic sampling of the robot configuration space.	The search tree is built with a deterministic B&B algorithm where the scheduling constraints are specified with an RTSG model.
Nodes represent collision-free configurations in the configuration space.	Nodes represent valid task sequences with respect to the scheduling constraints.
Edges represent collision-free paths between robot configurations. Path costs are typically fixed.	Edges represent valid transitions that extend task sequences with applicable tasks. Transition costs are updated for a new replanning scenario.
Offline learning phase to build the graph.	Offline learning phase to build the search tree.
Online query phase where the graph is used, e.g., at replanning, to identify a collision-free and potentially efficient path from a current state to a goal state.	Online query phase where the search tree is used, e.g. at replanning to identify a valid and potentially optimal task sequence from a current state to a goal state.

Table 8.1: Similarities of the basic characteristics for Probabilistic Roadmaps and Task Roadmaps, respectively.

same combination of tasks (but in different orders) where the last task is the same, leading to a similar state. In the example in Figure 8.2, the two nodes representing the sequences S-A-B-C and S-B-A-C are equivalent. The difference between equivalent nodes is mainly the cost of the respective sequence leading to this state. The possible propagation of task sequences from equivalent nodes is identical. This conclusion is not formally proved here but verified experimentally in the scenarios presented in Section 8.7 where B&B always finds valid and potentially optimal solutions with the same objective value as a MILP solver and a PDDL planner, at initial planning as well as at replanning in the query phase. The identical task sequence propagation from equivalent nodes is leveraged in the query phase. For this purpose, the algorithm always adds a *pruning edge* from a pruned node to the equivalent node with a better

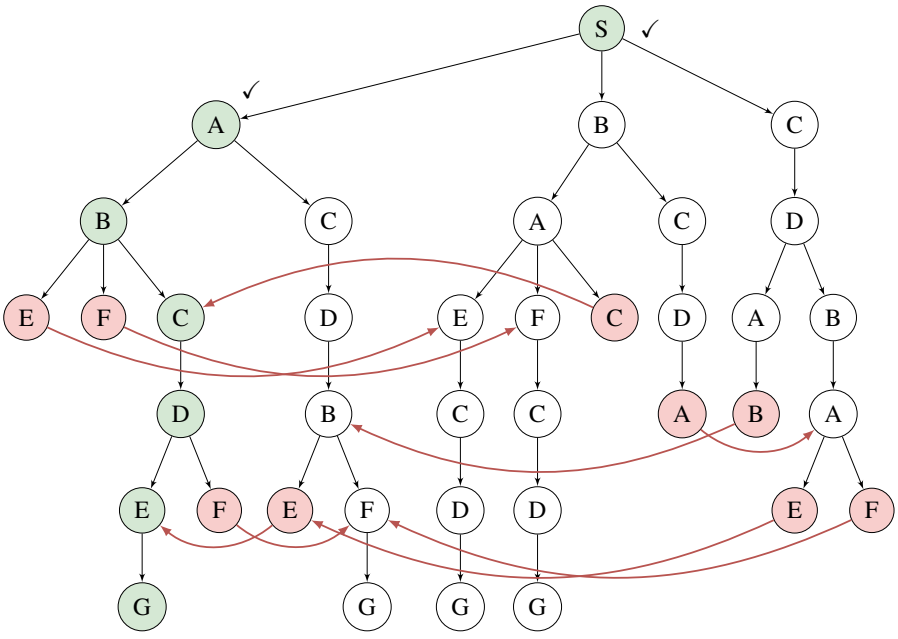

Figure 8.2: Visualization of a B&B search tree for the RTSG model in Figure 8.1. This expanded search tree makes up the Task Roadmap. The initial task sequence is marked in green and completed nodes have a checkmark. Pruned nodes are red and have red pruning edges.

cost that caused the pruning. In the following description, equivalent nodes that are interconnected with pruning edges are referred to as *peers*.

8.6.2 Query phase

At a point of replanning, the world is in a new state where the robot has completed 0 to $|A|$ tasks of the initial sequence. It is expected that the transition costs and the action costs of the remaining tasks (Equation (8.1)) may have changed, e.g., since the previously planned path to a task may have become temporarily blocked. Especially, the transition cost from the latest completed task to other tasks should be guided by the current state of the robot, e.g. its location. The query phase tries to identify an efficient sequence of tasks that will bring the robot from the current world state to the goal.

The query phase is divided into 2 steps:

1. The first step is to identify the *current node* of the search tree. This is the node that represents the sequence of already completed tasks. If no

tasks have been completed, the root node becomes the current node.

2. The second step is to find the most efficient task sequence between the current node and a goal-reaching node in the search tree. In this step, pruning edges are leveraged to explore nodes without children.

When replanning is made while reusing an existing search tree, we will refer to this method as B&B-TRM. And when replanning is made from scratch with a single and non-expanded root node, we will refer to this method as B&B. The memory required for replanning with B&B-TRM is always the same amount as required by B&B to find the initial task sequence in the learning phase and create the Task Roadmap. Replanning with B&B will in worst case require the same amount of memory as B&B-TRM, i.e. if no tasks are completed. If some tasks are already completed the memory need becomes reduced due to the smaller problem size. Both methods are realized with the same algorithm. This algorithm takes as input a list of already completed tasks. It also needs cost estimates for actions and transitions with respect to the current state. The search starts from the root node, which may be an initial single node or the top node of an existing search tree. It is a breadth-first search where any existing children are reused instead of being created while expanding the search. It is pruning all nodes in the first generation except the one that matches the first completed task. Then all nodes in the second generation are pruned except the one that matches the second completed task etc. This goes on until the current node is reached. From this node and onwards, only equivalent nodes are pruned. If an expansion is required from a node without children, the algorithm will look for a peer of the node. If no peer is found (e.g., since the algorithm is run without a TRM), all applicable children of the node are identified by a search of the RTSG and created. If instead a peer *is* found, the peer's children are adopted by this node. If the peer lacks children, the peer's peer's children are adopted etc. These adopted children are top nodes of subtrees that are disconnected from the peer and reconnected to the new parent node. The cost of all reused children adopted or previously existing, need to be updated to consider the new cost of the parent. The search continues until the goal is reached in all active (non-pruned) search nodes. Finally, the goal-reaching node with the lowest cost is identified.

Due to the exchange of subtrees between peers, each replanning will potentially modify the structure of the search tree. However, no information is lost that is required for later replanning. The significant reduction of planning time from reusing nodes comes from 1) removing the process of searching the RTSG to find possible search tree propagations and 2) removing the process of creating nodes from scratch.

Algorithm 1 B&B and B&B-TRM.

function SCHEDULETASKS($root$, $completedTasks$)
  $GS \leftarrow \varnothing$ ▷ Goal reaching sequences
  for $i \leftarrow 0$ to ∞ **do**
    $GN \leftarrow$ GETGENERATION$(root, GS, i+1)$
    if $GN = \varnothing$ **then**
      break ▷ No more generation to explore
    end if
    if $i \in [1, \ldots, |completedTasks|]$ **then**
      Prune all $c \in GN$ except the one matching the i-th
        element in the $completedTasks$ sequence
    else
      for all $c \in GN$ **do**
        if $\exists d \in GN$, c is equivalent with d, $cost(c) \geq cost(d)$ **then**
          $Prune(c)$
          $c.peer \leftarrow d$ ▷ Add a pruning edge
        end if
      end for
    end if
  end for
  return sequence in GS with lowest cost
end function

Algorithm 2

```
1: function GETGENERATION(tn, GS, depth)
2:   GN ← ∅
3:   if depth = 1 then                                   ▷ Check if this generation
4:     GN ← tn
5:   else if depth = 2 then                              ▷ Check if next generation
6:     if NotPruned(tn) then
7:       if HasNoChildren(tn) then
8:         if HasPeer(tn) then
9:           p ← GetPeerWithChildren(tn)
10:          for all c ∈ p.children do
11:            c.parent ← tn                             ▷ Assign a new parent
12:          end for
13:          tn.children ← p.children                    ▷ Adopt the peer's children
14:          tn.peer ← ∅                                 ▷ Remove the pruning edge to the peer
15:          p.children ← ∅                              ▷ Disconnect the peer from it's former children
16:          p.peer ← tn                                 ▷ Establish a pruning edge from the peer
17:        else
18:          tn.children ← SearchAndCreateChildren(tn)   ▷ A forward search of the RTSG for children
19:        end if
20:      end if
21:      if IsGoalReaching(tn) then
22:        AddGoalCost(tn)                               ▷ Add cost to reach the goal.
23:        GS ← GS ∪ tn                                  ▷ Add to the collection of goal reaching sequences
24:      else
25:        for all c ∈ tn.children do
26:          InheritParentCost(c)
27:          AddTaskCost(c)
28:        end for
29:        GN ← tn.children
30:      end if
31:    end if
32:  else                                                ▷ Later generations
33:    for all c ∈ tn.children do
34:      if NotPruned(c) then
35:        GN ← GN ∪ GETGENERATION(c, GS, depth − 1)
36:      end if
37:    end for
38:  end ifreturn GN
39: end function
```

8.6.3 B&B and B&B-TRM

A pseudo-code for B&B and B&B-TRM is given in Algorithm 1 and 2.

The algorithm starts at the *ScheduleTasks* function. It takes as input arguments a *root* node and a list of already completed tasks. The root node keeps the starting conditions for planning, e.g., the current location of the robot. At initial planning, the root node does not have any children. After completing the initial planning, it has been expanded to a search tree. At replanning, the expanded root node can be used as an input argument to *ScheduleTasks*, thereby speeding up the planning time. This usage scenario is referred to as the B&B-TRM algorithm. If instead a non-expanded root node is used, the search tree will be generated from scratch, which is referred to as the B&B algorithm.

ScheduleTasks runs a loop where a new generation of nodes is fully explored in every cycle, starting from the root node. From each generation, any goal-reaching nodes (sequences) are collected. The loop continues until no more generations can be explored. Finally, *ScheduleTasks* returns the goal-reaching sequence with the lowest cost. For each explored generation, pruning is made among equivalent nodes (in the algorithm referred to as $peers$) and pruning edges (object references to peers) are recorded for all pruned nodes to keep track of reusable sub-trees. If there are completed tasks, all children of the first generations, except the ones matching the completed sequence, are pruned.

To explore a new generation, the recursive *GetGeneration* function is used. The first argument, tn, is a reference to a tree node. The function returns a list of nodes representing a generation with respect to the tree node. The generation is specified with a relative $depth$ argument, where the value of 1 specifies the current generation, i.e. the tree node itself. A value of 2 indicates the children of the tree node while 3 indicates the grandchildren etc. If the grandchildren or later generations are specified, intermediate generations have already been explored due to the breadth-first search approach, and the existing children are used to explore the specified generation recursively. If instead the children generation is specified, the way of exploration will depend on the usage scenario. For the B&B scenario, where a reusable roadmap is missing, the children of the tree node are identified with the method *SearchAndCreateChildren*. This is a method that searches the RTSG model recursively, where any possible child nodes are created, configured, cost estimated and attached to the search tree. For B&B-TRM, existing children can be reused and if a tree node does not have children, these can be localized from its peer. If the peer has no children, the peer's peer is checked etc until children are found. Thereafter, the children are reconnected from the peer to the tree node. For

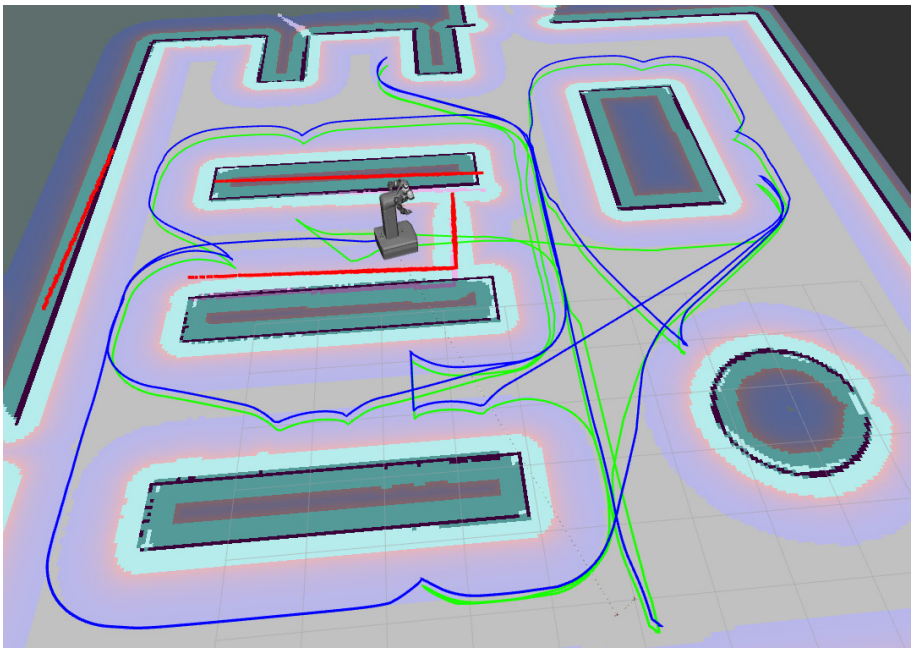

Figure 8.3: Warehouse layout. The light green path, starting and ending at the bottom right, is the initial route for Scenario B. The robot has stopped in front of an obstacle that is blocking the initially planned movement. The red lines along the shelves, the wall and the obstacle indicate the objects perceived by the robot's laser scanner. A replanned route in blue, starting at the robot's location, has changed the order of the remaining tasks.

both usage scenarios, the costs of the children are estimated as the cost of the parent plus the cost to perform the last task of the child with respect to the last task of the parent. Apart from returning a list of a generation, *GetGeneration* also updates the list of goal-reaching sequences, i.e. the GS argument, with any goal-reaching sequence.

8.7 Results

8.7.1 Use case

The use case is a kitting application in a warehouse. A mobile manipulator shall deliver kit boxes filled with several specified objects to a delivery station. Objects and empty kit boxes are located on 5 different shelves in a simple warehouse as illustrated in Figure 8.3.

The robot can carry 2 kit boxes and fill them in parallel. The process to

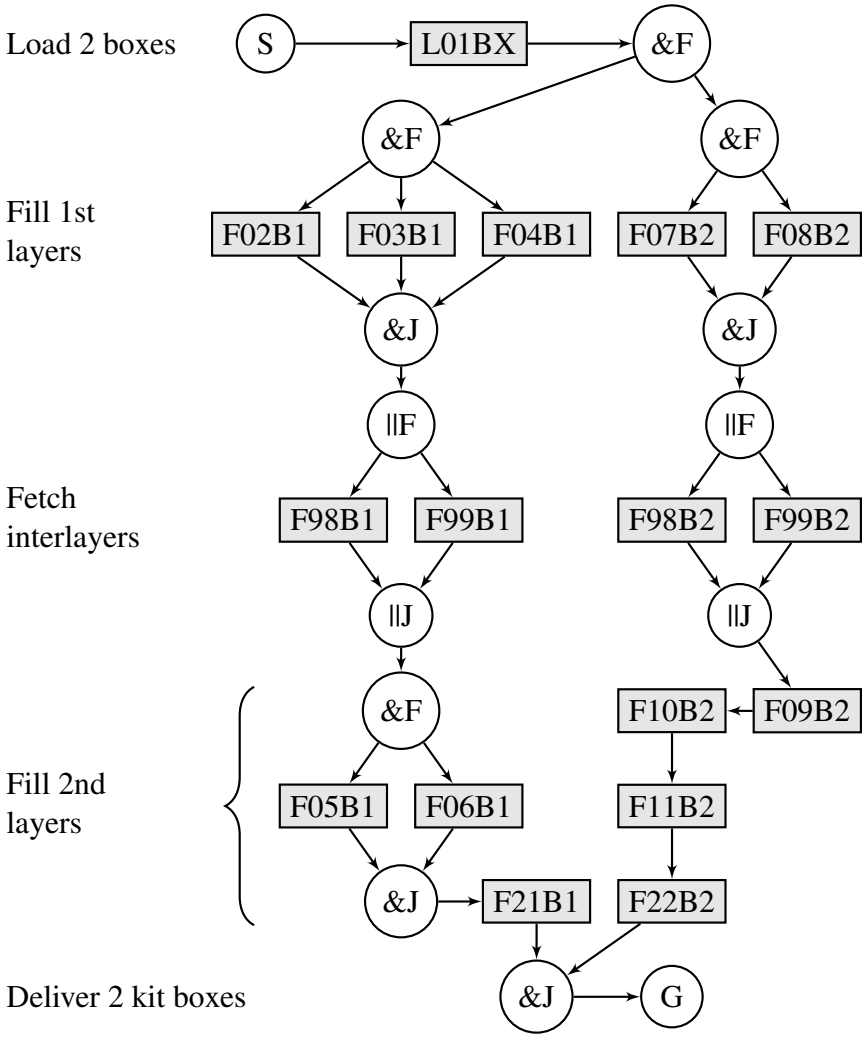

Figure 8.4: RTSG representation of Scenario A.

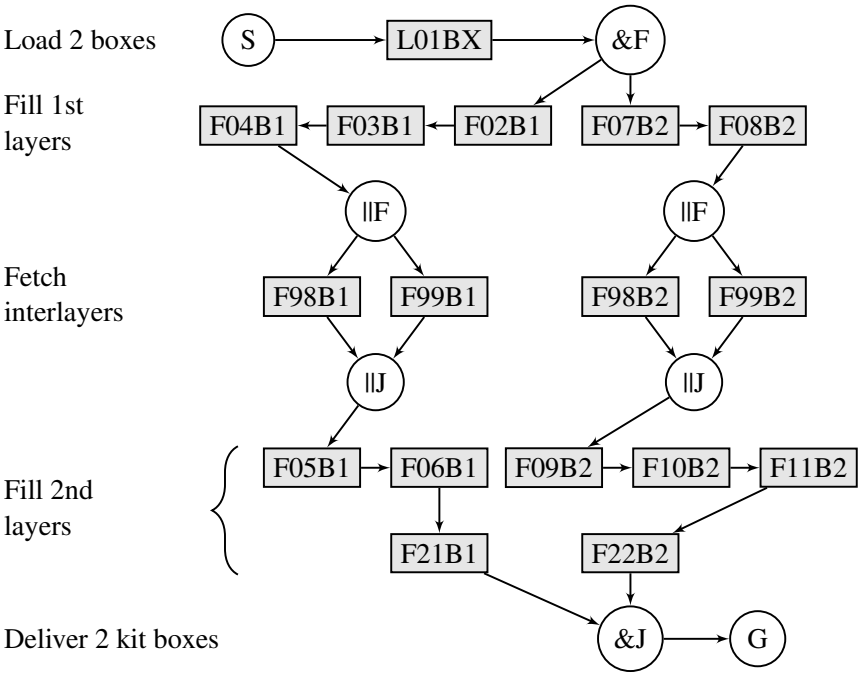

Figure 8.5: RTSG representation of Scenario B.

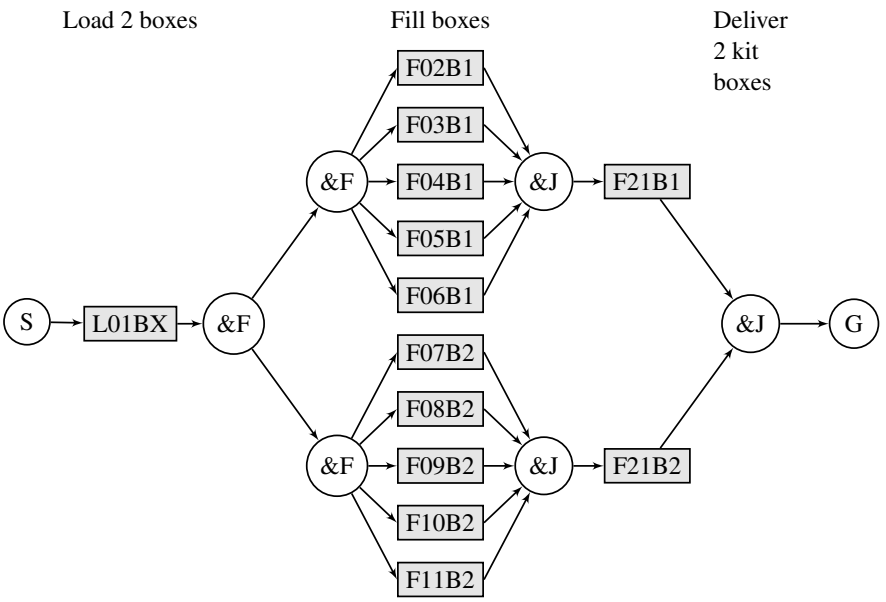

Figure 8.6: RTSG representation of Scenario C.

Scenario	#Variables	#Constraints
A	342	46
B	342	46
C	210	30

Table 8.2: Variables and constraints for the MILP problem formulations.

deliver the kits is divided into 5 phases: *Fetch empty kit boxes*, *Fetch layer 1*, *Fetch interlayer*, *Fetch layer 2*, and *Deliver kits*. There are two types of tasks: The first is to load empty kit boxes, and the second is to fetch an object to a carried kit box. Three different scenarios (A, B and C) are modelled with RTSG, see Figures 8.4–8.6. Task names, e.g., $F98B1$, indicate type of task (Fetch), location (98) and kit box (1). In all scenarios, the first task, $L01BX$, is to load two kit boxes from a shelf location (01). The remaining tasks require fetching one object at a shelf location into one of the two carried kit boxes. The first AND-pair splits the graph into two branches, each one modelling the filling of one kit box. The goal node represents the movement of the robot to the goal location where the filled kit boxes shall be delivered. Scenario A is a realistic specification for how kit boxes can be filled in an industrial context. For layer one, the kit objects can be fetched in any order. For layer 2, the right-hand side kit box is modelled to be filled in strict order ($F09B2$-$F10B2$-$F11B2$) while the left-hand side kit box provides some variability. A strict order can be desired, e.g., to achieve a predefined overlap of objects in the kit box. There are two alternative locations, 98 and 99, where an interlayer can be fetched. In Scenario B, tasks from Scenario A are rearranged in the graph, applying a strict order for each layer, thereby introducing more precedence constraints that reduce the variability of the task sequence. In Scenario C, the tasks are rearranged to minimize the number of precedence constraints, thereby maximizing the variability. Scenario C is closer to a TSP problem with no precedence constraints among the objects that shall be placed in the boxes.

When converting the three scenarios to MILP problem formulations, the number of variables and constraints are indicated in table 8.2. Additionally, some lazy constraints are added dynamically when required for equations 8.7 and 8.8 while running the MILP solver. The theoretical number of constraints for these equations can be very high and defining them all may not be viable.

8.7.2 Experimental setup

The simple warehouse world is modelled with the Gazebo simulator [28]. This includes the shelves, the mobile robot and an obstacle that may interfere with the robot path. ROS Navigation Stack [29] is used to navigate the robot between different locations. The navigation is guided by a 2D map of the warehouse that initially does not include the obstacle. While navigating, the robot simultaneously maps changes in the simulated world with respect to the map.

The benchmarked planners include the proposed B&B and B&B-TRM algorithms, a MILP solver [30] and Temporal Fast Downward (TFD) [25] which is a PDDL based temporal planner. An initial task sequence is generated from the RTSG model with B&B, MILP and TFD. Additionally, B&B generates a task roadmap that is reused by B&B-TRM in all replanning experiments. The transition costs between tasks are represented by the collision-free path lengths generated with the Dijkstra algorithm from the initial map. The initial task sequence corresponds to a route in the warehouse that starts and ends at the same location. In the simulation-based replanning experiments, the robot navigates the initially computed route, which turned out to be the very same route for all planners, while simultaneously mapping the environment with a simulated laser scanner. When the first task is reached, the next task is dispatched etc. During the progress of the plan execution, the planned motion to the next task becomes blocked by the obstacle at randomized locations along the path. The part of the obstacle that is visible from the robot while approaching it and stopping a short distance nearby (1-1.5 m), becomes included in the map. A replanning is initiated from a randomized location of the robot in front of the obstacle on the planned path. For this purpose, the initial planning problem is updated to become a replanning problem, considering already completed tasks, the robot's (randomized) current location and updated transition costs caused by the updated map. Depending on the location of the obstacle and its effect on collision-free path lengths between tasks, the replanned routes may change or keep the sequence of tasks. Replanning may fail if the location of the obstacle blocks a planned task when there are no alternative tasks. If a collision-free path can not be found between two tasks, the transition cost is penalized to a very high value that will help to detect a failed plan. All replanning scenarios were run with the benchmarked planners, including B&B-TRM and B&B (without the TRM). The number of already completed tasks at replanning, l, was in the range $l \in \{0, 1, \ldots, |A|\}$. 50 replannings were made for each l, with randomized robot and obstacle locations along the path between the location of the latest completed task and the next task. All experiments were run on the same computer with an Intel i5-4570, quadcore, 64-bit processor

having 7.6 GB RAM running on Ubuntu 18.04.5. For the MILP solver, the no-cyclic-subroutes constraints and the precedence constraints were implemented as lazy constraints in a customized Python callback routine. The B&B and the B&B-TRM algorithms were implemented in Python.

8.7.3 Experimental results

Figure 8.7 shows the experimental results for the four planner algorithms, i.e., B&B, B&B-TRM, TFD and the MILP solver. The graphs show the minimum, the median, and the maximum replanning time for the four algorithms, as a function of the number of completed tasks l; if l is 0, no task has been completed, and the replanning is performed with all the tasks in the task set, and therefore the replanning time is expected to be higher.

For each experiment in all scenarios, B&B-TRM and B&B reach the very same objective value as the solution computed by the MILP solver, suggesting that the proposed algorithms may be optimal. However, further investigation on the optimality of the proposed algorithms is needed. TFD also reaches the very same objective values in all experiments. The replanning time for B&B-TRM is less than 1 second (and often a fraction of that) for all scenarios. As expected from B&B-TRM essentially being a reduced version of B&B, it outperforms B&B in every scenario by an average factor of 26. The factor tends to increase slightly for scaled-up problem instances with fewer completed tasks.

B&B-TRM outperforms the MILP solver in Scenario A by a factor greater than 152 for the largest problem instances $l \in \{0, \ldots, 8\}$. The difference is slightly less for the smaller problem instances but still significant (> 18). For Scenario B, B&B-TRM is more than 137 times faster for $l \in \{0, \ldots, 8\}$. For Scenario C and $l \in \{0, 1, 2\}$, B&B-TRM is only 1.8, 2.3 and 2.6 times faster than the MILP solver, but for $l > 2$, the proposed method exhibits significantly better performance. The decreasing difference to the MILP solver for the larger problem instances in Scenario C is correlated with a fast-growing search tree and increased memory consumption. For Scenario A and B, the growth of the search tree at each breadth-first expansion step is limited by the more numerous precedence constraints, thereby reducing the memory consumption considerably. Scenario C highlights the main limitation of the proposed approach, i.e., for loosely constrained problems (Figure 8.6), the number of alternatives grow significantly in breadth-first tree-based search approaches. On the other hand, the specification of robot applications is typically closer to Scenario A (Figure 8.4), with a more structured sequence of tasks, hence with additional constraints. In such cases, the proposed approach can provide a significant speedup.

B&B-TRM outperforms TFD by a factor greater than 149 for all $l \in \{0, \ldots, |A|\}$ in Scenario A. This factor is 151 for Scenario B and 68 for Scenario C, respectively. A part of this performance difference may depend on the fact that the proposed algorithm is designed for the RTSG modelling formalism used to model these scenarios, while TFD is designed for the PDDL modelling formalism.

Finally, it is worth noticing that both the B&B-TRM approach and TFD provide a more predictable performance than MILP, highlighted by the limited variability (standard deviation) of the replanning time.

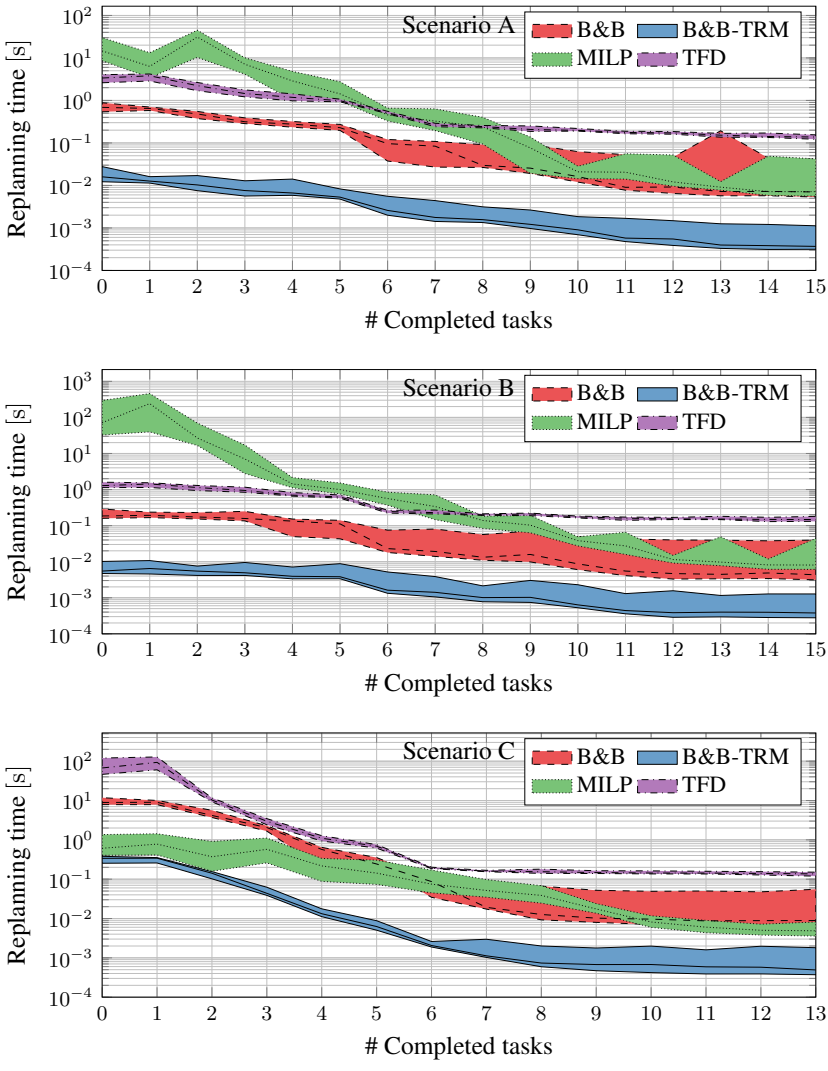

Figure 8.7: Replanning time for B&B (dashed area), B&B-TRM (solid area) and MILP (dotted area) at different task completion levels. The central line in each area indicates the median. Note that the vertical axis is logarithmic.

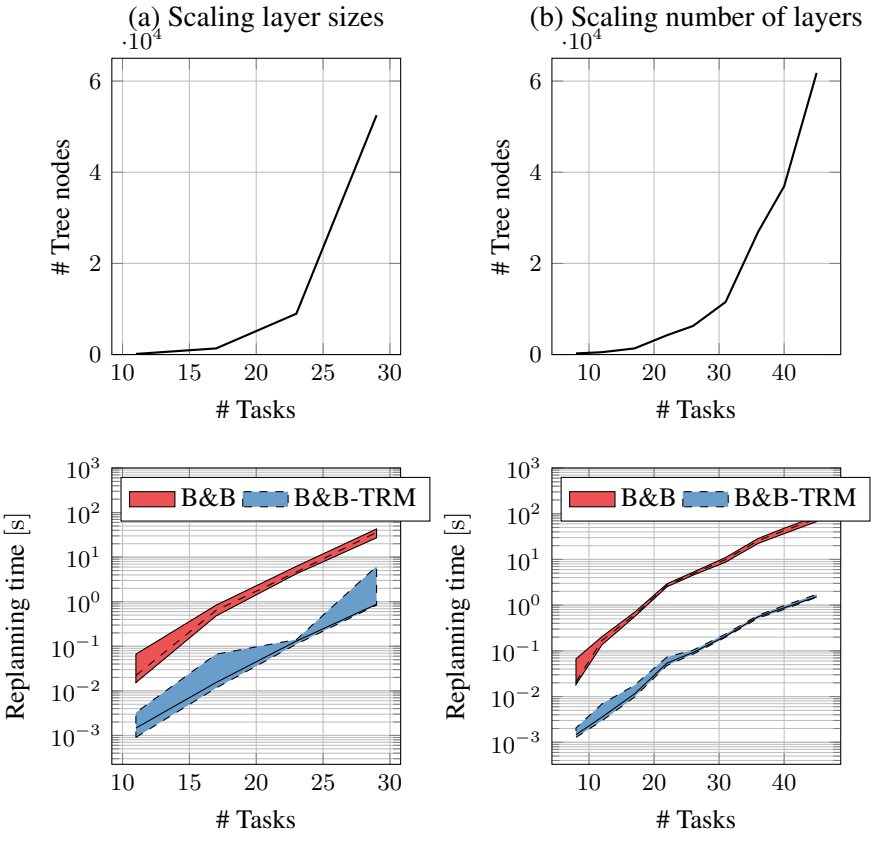

Figure 8.8: Scalability experiments for Scenario A. '#Tasks' indicates the number of tasks in the RTSG. '#Tree nodes' indicates the number of nodes in the Task Roadmap search tree.

8.7.4 Scalability investigation

As discussed in section 8.3.5, the complexity of the scheduling problem depends on the structure of the graph. It is assumed that the scalability of the presented algorithm also depends on this structure, and how the graph is scaled. In order to investigate how the algorithm may scale for growing problem sizes, an experiment was setup targeting the graph in scenario A. The size of this graph was modified in different steps. For each problem size, memory consumption and replanning time were measured. The number of tree nodes in the Task Roadmap are used to represent memory consumption. All replanning times were measured at random locations before completing the first very task, which is the worst case. Scenario A was scaled in two ways, (a) by changing the number of tasks in each layer of the kit boxes (Figure 8.8(a)), and (b) by changing the number of layers in the kit boxes (Figure 8.8(b)). Both scaling scenarios indicate an exponential growth of the memory consumption, but at different rates where scaling the number of layers is more advantageous. Scaling the number of layers introduces a lot of precedence constraints, while the scaling of layer sizes introduces very few. The B&B-TRM replanning times are within a few seconds for the included problem sizes. The scalability experiments confirms that memory consumption may be a limitation for the algorithm in some scenarios. Especially for less constrained, scaled up problem scenarios.

8.8 Conclusion and future work

We have proposed the concept of Task Roadmaps (TRM) and shown that it is a promising strategy to speed up online replanning of robot tasks, thereby contributing to improved productivity in a dynamic environment. We have presented a strategy to implement Task Roadmaps, using a Robot Task Scheduling Graph to model a robot application, Branch and Bound (B&B) for initial planning and B&B-TRM for replanning. The benefits, as well as the limitations for this strategy, have been investigated in an experimental study where a MILP solver and a PDDL based planner have been used as benchmarks.

Future work will address the combining of different replanning strategies with the modelling and runtime observation of disturbance behaviours. Another interesting extension is to widen the scope to multi-robot task allocation and scheduling.

Bibliography

[1] A. Ajoudani, A.M. Zanchettin, S. Ivaldi, A. Albu-Schaeffer, K. Kosuge, and O. Khatib. Progress and prospects of the human–robot collaboration. *Auton Robot*, 42:957–975, 2018.

[2] M. M. Costa and M. F. Silva. A survey on path planning algorithms for mobile robots. In *2019 IEEE Int. Conf. on Autonomous Robot Systems and Competitions (ICARSC)*, pages 1–7, 2019.

[3] Thi Thoa Mac, Cosmin Copot, Duc Trung Tran, and Robin De Keyser. Heuristic approaches in robot path planning: A survey. *Robotics and Autonomous Systems*, 86:13–28, 2016.

[4] P. Tajvar, F. S. Barbosa, and J. Tumova. Safe motion planning for an uncertain non-holonomic system with temporal logic specification. In *IEEE Int. Conf. on Aut. Science and Eng. (CASE)*, pages 349–354, 2020.

[5] Anna Mannucci, Lucia Pallottino, and Federico Pecora. Provably safe multi-robot coordination with unreliable communication. *IEEE Robotics and Automation Letters*, 4(4):3232–3239, 2019.

[6] L. E. Kavraki, P. Svestka, J. . Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Rob. and Aut.*, 12(4):566–580, 1996.

[7] A. Lager, G. Spampinato, A. V. Papadopoulos, and T. Nolte. Towards reactive robot applications in dynamic environments. In *IEEE Int. Conf. on Emerging Tech. and Factory Automation (ETFA)*, pages 1603–1606, 2019.

[8] Anders Lager, Alessandro Papadopoulos, Giacomo Spampinato, and Thomas Nolte. A task modelling formalism for industrial mobile robot applications. In *20th International Conference on Advanced Robotics*, 2021.

[9] Branko Miloradović, Baran Çürüklü, Mikael Ekström, and Alessandro Vittorio Papadopoulos. A genetic algorithm approach to multi-agent mission planning problems. In *Operations Research and Enterprise Systems*, pages 109–134, Cham, 2020. Springer Int. Publishing.

[10] Branko Miloradović, Baran Çürüklü, Mikael Ekström, and Alessandro Vittorio Papadopoulos. Gmp: A genetic mission planner for het-

erogeneous multi-robot system applications. *IEEE Transactions on Cybernetics*, Mar. 2021.

[11] Martin Weser, Dominik Off, and Jianwei Zhang. Htn robot planning in partially observable dynamic environments. pages 1505–1510, 05 2010.

[12] G. Kazhoyan, A. Niedzwiecki, and M. Beetz. Towards plan transformations for real-world mobile fetch and place. In *IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 11011–11017, 2020.

[13] D. Hadfield-Menell, L. P. Kaelbling, and T. Lozano-Pérez. Optimization in the now: Dynamic peephole optimization for hierarchical planning. In *2013 IEEE Int. Conf. on Robotics and Automation*, pages 4560–4567, 2013.

[14] Ping Lou, Quan Liu, Zude Zhou, Huaiqing Wang, and Sherry Sun. Multi-agent-based proactive–reactive scheduling for a job shop. *Int. Journal of Advanced Manufacturing Technology - INT J ADV MANUF TECHNOL*, 59, 03 2012.

[15] Drew McDermott. Robot planning. *AI Magazine*, 13(2):55, Jun. 1992.

[16] Oscar Lima, Michael Cashmore, Daniele Magazzeni, Andrea Micheli, and Rodrigo Ventura. Robust plan execution with unexpected observations, 2020.

[17] Irina Dumitrescu, Stefan Ropke, Jean-François Cordeau, and Gilbert Laporte. The traveling salesman problem with pickup and delivery: Polyhedral results and a branch-and-cut algorithm. *Mathematical Programming*, 121:269–305, 07 2009.

[18] Yaroslav Salii. Revisiting dynamic programming for precedence-constrained traveling salesman problem and its time-dependent generalization. *European Journal of Operational Research*, 272(1):32 – 42, 2019.

[19] Brian P. Gerkey and Maja J. Matarić. A formal analysis and taxonomy of task allocation in multi-robot systems. *The International Journal of Robotics Research*, 23(9):939–954, 2004.

[20] J. Grefenstette, R. Gopal, B. Rosmaita, and D. Van Gucht. Genetic algorithms for the traveling salesman problem. pages 160–168, 1985.

[21] Jussi Rintanen. Complexity of concurrent temporal planning. In *Proceedings of the Seventeenth International Conference on International Conference on Automated Planning and Scheduling*, ICAPS'07, page 280–287, 2007.

[22] Florian Geißer, Thomas Keller, and Robert Mattmüller. Delete relaxations for planning with state-dependent action costs. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15, page 1573–1579, 2015.

[23] Christer Bäckström and Bernhard Nebel. Complexity results for sas+ planning. *Computational Intelligence*, 11(4):625–655, 1995.

[24] Richard E. Fikes and Nils J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3):189 – 208, 1971.

[25] Patrick Eyerich, Robert Mattmüller, and Gabriele Röger. Using the context-enhanced additive heuristic for temporal and numeric planning. ICAPS'09, 2009.

[26] Andrew Coles, Amanda Coles, Allan Clark, and Stephen Gilmore. Cost-sensitive concurrent planning under duration uncertainty for service-level agreements. 2011.

[27] M. Fox and D. Long. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *ArXiv*, abs/1106.4561, 2003.

[28] N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, volume 3, pages 2149–2154 vol.3, 2004.

[29] Rodrigo Longhi Guimarães, André Schneider de Oliveira, João Alberto Fabro, Thiago Becker, and Vinícius Amilgar Brenner. *ROS Navigation: Concepts and Tutorial*, pages 121–160. 2016.

[30] LLC Gurobi Optimization. Gurobi optimizer reference manual, 2021.

Chapter 9

Paper C
A Scalable Heuristic for Mission Planning of Mobile Robot Teams

Anders Lager, Branko Miloradović, Alessandro V. Papadopoulos, Giacomo Spampinato and Thomas Nolte. In 22nd World Congress of the International Federation of Automatic Control (IFAC), 2023.

Abstract

In this work, we investigate a task planning problem for assigning and planning a mobile robot team to jointly perform a kitting application with alternative task locations. To this end, the application is modeled as a Robot Task Scheduling Graph and the planning problem is modeled as a Mixed Integer Linear Program (MILP). We propose a heuristic approach to solve the problem with a practically useful performance in terms of scalability and computation time. The experimental evaluation shows that our heuristic approach is able to find efficient plans, in comparison with both optimal and non-optimal MILP solutions, in a fraction of the planning time.

9.1 Introduction

Coordinating a fleet of robots to perform various tasks is a problem with high complexity, requiring the solving of many sub-problems. In this paper, we investigate how to efficiently plan a kitting application mission, including the selection of a robot team for this purpose. In a kitting application, a specified group of objects of different types shall be fetched from different locations and thereafter delivered to a specified location. To increase operational efficiency, some object types may be available at multiple alternative locations, e.g., if the demand for these objects is high. Robots available for the mission may be located at different locations, e.g., for charging. The targeted objective is to minimize the makespan, which promotes a balanced usage of the robots and reduces the aggregation time at the delivery location.

This problem belongs to the category of Multi-Robot Task Allocation (MRTA) [1]. It can also be categorized as a variant of the Capacitated Vehicle Routing Problem (VRP), first studied by [2], and extensively studied during the latest decades to solve many related problem variants in different domains, e.g., logistic operations and production planning [3, 4]. For a basic VRP, the problem is to deliver a set of customer orders with a set of vehicles located at a single depot. The customers to be served are located at different locations and the vehicles need to return to the depot after all customers have been served. Vehicles have a limited order capacity. A solution to the problem will decide the number of vehicles and their routes, while the objective is to minimize the total cost of all routes. In our problem variant, vehicles represent mobile robots, and customers to be visited represent robot tasks to be done. Since the robots are assumed to have an unlimited capacity, the problem may also be categorized as a Multiple Travelling Salesperson Problem (MTSP) which covers a subset of VRPs without capacity constraints [5]. The selection of a team of robots, each one having a different start location, can be categorized as a VRP with Multiple Depots [6], here assuming there is only one robot at each depot. The problem includes AND-type precedence constraints [7], limiting the execution order of intra-schedule tasks [8]. It also includes separation constraints [9] to provide mutual exclusion of delivery tasks within the same route. These tasks are used to deliver aggregated objects. Additionally, the robots are not required to return to their starting depots to complete the mission, making it an Open VRP. In practice, the robots may immediately become available for a new mission and they may start moving towards a currently free charging location. The problem can also be categorized as a Balanced VRP since a minimum makespan objective promotes a balanced usage of the robot team, as long as the mission does not

include dominating tasks [10]. The existence of sets of alternative tasks, in which only one task needs to be visited, motivates the categorization of the problem as a Multiple Generalized TSP (MGTSP). MGTSP and the single robot version GTSP [11], are generalizations of MTSP and TSP respectively, where the tasks are divided into different subsets and at least one task in each subset needs to be visited. In our problem, these subsets are mutually exclusive and exactly one task needs to be visited. To summarize, we have a Balanced, Open, Multi Depot, VRP/MGTSP with Precedence Constraints (PC) and separation constraints.

In this paper, we propose an interpretation and extension of the Robot Task Scheduling Graph (RTSG) [12] to model the addressed multi-robot problem. Moreover, we provide a problem formulation as a Mixed Integer Linear Program (MILP), that can be used by a MILP Solver to find optimal solutions for smaller problem instances, given enough time. We also propose a heuristic multi-step approach, targeting a reduced planning time and efficient solving of larger problem instances. In the first step, tasks are partitioned into clusters with a semi-supervised clustering approach based on K-medoids. Thereafter, the clusters are modeled as separate single-route scheduling problems using MILP, or as asymmetric Travelling Salesperson Problems. Finally, any remaining alternative tasks are removed and the computed schedules are balanced with a local search approach to further minimize the makespan.

The solution quality and the planning time of the heuristic approach are compared with a MILP solver, indicating an ability to generate high-quality solutions within a practically acceptable time frame. Other benchmark approaches were not found that can be applied to the full problem description in an obvious way. To the best of our knowledge, OR-type PC for VRP, [7], with *alternative* predecessors, have not been addressed in the VRP literature.

The remainder of this paper is organized as follows. Sect. 9.2 presents related works. Sect. 9.3 describes the problem and the assumptions made, while Sect. 9.4 gives a formal problem formulation as a MILP. Sect. 9.5 details the heuristic solution approach, and Sect. 9.6 provides the experimental evaluation. The study is concluded in Sect. 9.7.

9.2 Related work

Robot Task Scheduling Graph is an intuitive task modeling formalism proposed by [12]. It can be used by a domain expert to model an industrial mobile robot application while leveraging automated planning. Until this work, this modeling formalism has only been applied to single-robot planning.

For a VRP, a customer normally *must* be served and there is only one address to go to. However, a task to be performed by a robot can often be alternative or be performed at alternative locations. Examples of alternative customers (or task locations) in the VRP literature are sparse but can be found, e.g., in the work by [13], where the profit is maximized by deciding if a transportation request shall be assigned to a vehicle or bought. [14] used an MGTSP problem formulation to plan a team of charging robots to support a team of Unmanned Aerial Vehicles on the ground at alternative locations along planned flight trajectories. [15] targeted a combined sequencing and path generation problem for industrial robots, using a genetic algorithm to solve an MGTSP for the sequencing including a selection of alternative robot configurations.

Precedence constraints of AND-type have been used extensively in previous works, and OR-type PC was suggested recently for VRP where only one of a task's predecessors must precede the task in a plan [7, 16]. In our problem, OR-type PC are modeled. However, different from the mentioned works, OR-type predecessors of a task are *alternative* in the sense that only one will exist in a plan.

The objective to minimize the makespan can often be improved by increasing the robot team size. However, robots are a limited resource that may be used in parallel for alternative missions or multi-mission problems [17]. Therefore, being able to specify the number of robots, i.e., the number of clusters for the proposed heuristic approach, can be considered an advantage for this problem. K-means is a popular method to partition nodes into K clusters where the total distance between the nodes and their Euclidian cluster center points is minimized [18]. However, the routes for our problem, e.g., in a warehouse, are seldom linear and a Euclidian center point may not be close or even reachable from other cluster nodes. K-medoids [19] is a more suitable approach where a node is identified as a center point. To consider separation constraints in the computation of clusters, a supervised clustering approach can be applied. Our supervised approach is based on the Variable Neighborhood Search (VNS) algorithm presented by [20]. Cluster algorithms that also identify the number of clusters sometimes referred to as *automatic clustering*, can be appropriate for some problem types. Several automatic clustering approaches for MRTA are listed by [21].

[21], addressed an MRTA problem where tasks shall be assigned to robots with the objective to minimize the total completion time given by a fitness function. They used a two-step approach with dynamic clustering based on Particle Swarm Optimization followed by a robot assignment to clusters with an approach including the solving of TSP problems for each combination. [22] demonstrated a two-phase heuristic approach to compute solutions for a Bal-

anced MTSP, where a balanced K-means was used to get clusters with evenly distributed nodes. This was followed by a genetic algorithm to compute routes. [23] compared the performances of different clustering algorithms for solving a Balanced MTSP, where a convex hull TSP algorithm was used to generate the routes for each cluster. The modeling of alternative tasks for the single route problem within an asymmetric TSP has similarities with the approach for converting an asymmetric GTSP into an asymmetric TSP described by [24].

9.3 Problem description and assumptions

The problem is to fetch and deliver a set of objects while minimizing the mission makespan. To this end, a team of robots needs to complete a set of *tasks*. These are Single-robot Tasks and the robots are Single-task Robots in a Time-extended Assignment [25]. The team size is fixed beforehand. The problem has In-Schedule Dependencies but no Cross-Schedule Dependencies [8]. It is assumed there is no restriction in the number of fetch tasks that can be allocated to a single robot. Furthermore, the robots are homogeneous and initially located at different start locations. Each task is associated with a location and an action duration. A *schedule* is referred to as an ordered sequence of tasks to be executed by a single robot. The solution shall identify 1) the robots to be used for the mission, and 2) the schedules for the used robots. A mission work description can be modeled with an RTSG model [12]. In this representation, exemplified in Fig. 9.1, rectangular nodes represent tasks. Directed edges and paths represent precedence constraints. A start state is represented by an S-node and the goal state is represented by a G-node. AND-Fork (&F) and AND-Join (&J) node pairs are used to split and rejoin edges. OR-Fork (||F) and OR-Join (||J) node pairs split a branch into alternative branches. In previous works, RTSG has exclusively been used with a single robot.

In our model (Fig. 9.1), a set of fetch tasks ($F_i, G_{j,k} : i, j, k \in \mathbb{N}$) need to be completed before the execution of delivery tasks ($D_l : l \in \mathbb{N}$). The model allows for defining a number of OR-pair groups, each one containing a set of alternative fetch tasks ($G_{j,k}$) where only one of them will be planned (j indicates the OR-pair group). There is one delivery task for each schedule. The delivery tasks can be co-located, but this is not a requirement for the solution approaches. We propose a few multi-robot interpretations and extensions of RTSG for the multi-robot problem at hand:

- Precedence constraints do only apply if the involved tasks become allocated to the same schedule.
- If a predecessor of a task is mandatory, it matches an AND-type PC for

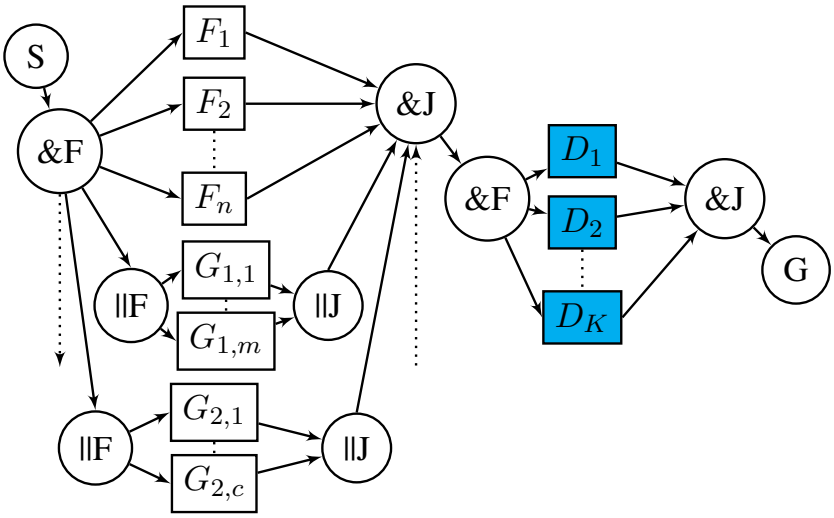

Figure 9.1: A work description for the multi-robot mission modeled as a Robot Task Scheduling Graph.

VRP in accordance with [7], where all predecessors of a task must be planned before any successor.

- If a predecessor is alternative, it matches an OR-type PC for VRP, as proposed by [7], where only one of the predecessors to a common task must precede it. Since the predecessors are alternative, only one of them will be planned for execution.

- A Separation Constraint (SC) indicates a group of tasks that needs to be separated into different schedules. In Fig. 9.1, the blue color is used to indicate a SC for the group of delivery tasks.

In the proposed form, the RTSG model is a flexible but constrained multi-robot work description.

9.4 Problem formulation

In this section, a MILP problem formulation is proposed that can be used to compute optimal solutions of the MRTA problem presented in Section 9.3.

9.4.1 Decision variables and objective

Let r be a robot schedule in the set of schedules R, where the number of total schedules is known *a priori*. The set of robot start locations is defined by S, representing available robots. For each start location $s \in S$, there is a corresponding goal state, $E(s) \in G$. $|S| = |G| \geq |R|$. The set of all tasks is denoted as A. For convenience, we indicate with $A^S = A \cup S$, with $A^G = A \cup G$, and with $\tilde{A} = A \cup S \cup G$. The set B is the set of all non-mandatory tasks, indicating they may, or may not be a part of a schedule. The set B is a subset of set A, i.e., $B \subseteq A$. Other tasks that belong to the set $A \setminus B$ are mandatory. The notation $j \prec k$ where $j, k \in \tilde{A}$, indicates task j must be scheduled before task k if they are assigned to the same schedule.

Decision variable $X_{j,k,r}$ is a binary variable, i.e., $X_{j,k,r} \in \{0, 1\}, \forall j, k \in \tilde{A}$, and $\forall r \in R$, where

$$X_{j,k,r} = \begin{cases} 1, & \text{if there is a scheduled direct transition} \\ & \text{from task } j \text{ to task } k \text{ within schedule } r, \\ 0, & \text{otherwise,} \end{cases}$$

where $X_{j,j,r} = 0$, $\forall j \in \tilde{A}$, $\forall r \in R$. The cost to perform task k after task j within the same schedule includes routing cost ($\tau_{j,k}$) and action cost (α_k):

$$C_{j,k} = \tau_{j,k} + \alpha_k. \tag{9.1}$$

There is no cost to reach the goal state, $C_{j,G} = 0, \forall j \in \tilde{A}$. The objective function used for optimization is a MiniMAX [26]. The objective function J is defined as:

$$J = \max_r \sum_{j \in A^S} \sum_{k \in A^G} (X_{j,k,r} \cdot C_{j,k}). \tag{9.2}$$

Since J is not a linear function in $X_{j,k,r}$ for Equation 9.2, the objective function is modeled as a constraint by including J as one extra decision variable where the size of J is limited by additional MiniMAX constraints for the total cost of each schedule:

$$-J + \sum_{j \in A^S} \sum_{k \in A^G} (X_{j,k,r} \cdot C_{j,k}) \leq 0, \qquad \forall r \in R, \tag{9.3}$$

9.4.2 General constraints

There are transitions from exactly $|R|$ robot start locations:

$$\sum_{j \in S} \sum_{k \in A^G} \sum_{r \in R} X_{j,k,r} = |R|. \tag{9.4}$$

If a transition from a robot start location goes directly to the goal state, i.e., $X_{j,k,r} = 1 : j \in S, k \in G, r \in R$, schedule r is considered empty and the robot at start location j will not be allocated. There is at most one transition from a robot start location and there are no transitions to a robot start location as defined by:

$$\sum_{k \in A^G} \sum_{r \in R} X_{j,k,r} \leq 1, \qquad \forall j \in S, \tag{9.5}$$

$$\sum_{j \in \tilde{A}} \sum_{r \in R} X_{j,k,r} = 0, \qquad \forall k \in S. \tag{9.6}$$

Similarly, there is at most one transition to a robot goal state and there are no transitions from a robot goal state:

$$\sum_{j \in A^S} \sum_{r \in R} X_{j,k,r} \leq 1, \qquad \forall k \in G, \tag{9.7}$$

$$\sum_{k \in \tilde{A}} \sum_{r \in R} X_{j,k,r} = 0, \qquad \forall j \in G. \tag{9.8}$$

One start location is included in each schedule:

$$\sum_{j \in S} \sum_{k \in A^G} X_{j,k,r} = 1, \qquad \forall r \in R. \tag{9.9}$$

The number of transitions from a robot start location in a schedule is the same as the number of transitions to the corresponding goal state:

$$\sum_{k \in A^G} X_{j,k,r} = \sum_{j' \in A^S} X_{j',E(j),r}, \quad \forall j \in S, \forall r \in R. \tag{9.10}$$

There is one transition from and one transition to mandatory tasks:

$$\sum_{k \in A^G} \sum_{r \in R} X_{j,k,r} = 1, \qquad \forall j \in A \setminus B, \tag{9.11}$$

$$\sum_{j \in A^S} \sum_{r \in R} X_{j,k,r} = 1, \qquad \forall k \in A \setminus B. \tag{9.12}$$

There is at most one transition from and one transition to non-mandatory tasks.

$$\sum_{k \in \tilde{A}} \sum_{r \in R} X_{j,k,r} \leq 1, \qquad \forall j \in B, \tag{9.13}$$

$$\sum_{j \in \tilde{A}} \sum_{r \in R} X_{j,k,r} \leq 1, \qquad \forall k \in B. \tag{9.14}$$

The number of incoming and outgoing transitions to non-mandatory tasks must be the same:

$$\sum_{k\in\tilde{A}}\sum_{r\in R} X_{j,k,r} = \sum_{j'\in\tilde{A}}\sum_{r\in R} X_{j',j,r}, \qquad \forall j \in B. \tag{9.15}$$

A task is entered and departed within the same schedule:

$$\sum_{j\in A^S} X_{j,k,r} = \sum_{k'\in A^G} X_{k,k',r}, \qquad \forall k \in A, \forall r \in R. \tag{9.16}$$

No cyclic sub-route are allowed:

$$\sum_{j\in V}\sum_{k\in V} X_{j,k,r} \leq |V| - 1, \quad \forall V \subseteq A : V \neq \emptyset, \forall r \in R. \tag{9.17}$$

Thus, to eliminate the sub-tours, it is required that for each nonempty subset $V \subseteq A$, the number of edges between the elements of V must be at most $|V| - 1$. Let O be a set of alternative tasks in the set of all sets of alternative tasks $\tilde{O}$, i.e., $O \in \tilde{O}$ where $O \cap U = \emptyset, \forall U \in \tilde{O} \setminus O$. Additionally, O is a subset of B, i.e., $O \subseteq B$. The number of transitions to and from a set of alternative tasks is limited to 1.

$$\sum_{s\in O}\sum_{j\in A^G}\sum_{r\in R} X_{s,j,r} = 1, \qquad \forall O \in \tilde{O} \tag{9.18}$$

$$\sum_{j\in A^S}\sum_{s\in O}\sum_{r\in R} X_{j,s,r} = 1, \qquad \forall O \in \tilde{O} \tag{9.19}$$

In order to constrain fetch tasks to be done before delivery tasks within the same schedule, as indicated by the edges of the RTSG in Fig. 9.1, general precedence constraints are introduced:

$$\begin{aligned} &\sum_{j=1}^{|D|-1} X_{D_j,D_{j+1},r} \leq |D| - 2, \\ &\forall D \subseteq \tilde{A} : |D| \geq 2,\ D_{|D|} \prec D_1,\ \forall r \in R, \end{aligned} \tag{9.20}$$

where D is an ordered subset $D = \{D_1, \ldots, D_{|D|}\} \subseteq \tilde{A}$.

9.4.3 Delivery task constraints

Delivery tasks, $P \subseteq B$, cannot be allocated to the same schedule. The number of delivery tasks equals the number of computed schedules, $|P| = |R|$. Since

schedules may become empty, delivery tasks are a subset of the non-mandatory tasks, B. At most one delivery task in P is allocated to schedule r, i.e., a separation constraint. No delivery task is allocated to r if the schedule is empty, meaning there is a direct transition between the start location and the goal state:

$$\sum_{j \in A^S} \sum_{k \in P} X_{j,k,r} \leq 1, \qquad \forall P \subseteq B, \forall r \in R, \tag{9.21}$$

$$X_{s,E(s),r} + \sum_{j \in A^S} \sum_{k \in P} X_{j,k,r} = 1, \quad \forall P \subseteq B, \forall s \in S, \forall r \in R. \tag{9.22}$$

9.5 Heuristic approach

The general idea of the proposed heuristic approach is to partition all tasks into K mutually exclusive clusters, where K is the number of robots to use. Thereafter, a single route scheduling problem is solved for each cluster. Eventually, alternative tasks remaining in different routes are reduced, and the costs of the routes are balanced by transferring tasks between the routes.

9.5.1 Task Clustering

To reduce routing costs, it is assumed that tasks within the same cluster should be in proximity to each other. K-medoids partitions the nodes into K clusters while minimizing the total dissimilarities of the nodes and their *medoid*, i.e., a node assigned as the center for a cluster:

$$\sum_{l \in L} \sum_{i \in l} d_{m_l,i}, \tag{9.23}$$

where L is the set of clusters ($|L| = K$), $d_{x,y}$ is a dissimilarity measure of nodes x and y. The assigned medoid of cluster l is m_l.

For our problem, nodes represent robot tasks, and the elements of the dissimilarity matrix, $d_{x,y}$, represent routing costs between task x and task y. The separation constraints for the delivery tasks require them to end up in different clusters. These constraints are converted to *cannot-link constraints*, indicating tasks that must be separated in different clusters [27].

The clustering approach is detailed in Algorithm 3. It implements the Semi-Supervised K-medoids algorithm presented by [20]. It starts with an initial random selection of K medoids, and the tasks in the set A are grouped with their closest medoid. In each iteration, a randomly selected subset of the medoids are exchanged and the tasks are regrouped. Thereafter, the $LocalSearch$ algorithm, proposed by [28], reduces the total cluster cost

Algorithm 3 Semi Supervised K-medoids

```
function ComputeClusters(A,k)
  CL ← K-medoids clustering of A with k random medoids
  CL ← Repair(CL)
  ret ← CL
  while Stop criterion not reached do
    v ← 1
    while v ≤ k do
      CL ← Switch v medoids randomly in CL
      CL ← Repair(CL)
      CL ← LocalSearch(CL)
      CL ← Repair(CL)
      if CL.cost() < ret.cost() then
        ret ← CL
      end if
      v ← v + 1
    end while
  end while
  return ret
end function
```

(9.23) by greedy modifications of the medoid selections and regrouping of tasks. After each modification of the medoid selection, a repair step adjusts the clusters to satisfy cannot-link constraints by moving conflicting, non-medoid tasks between clusters. Different from the original approach by [20], the cost of the clusters to be minimized is set in this paper as the max cluster cost:

$$\max_{l \in L} \sum_{i \in l} d_{m_l,i} + \alpha_i \tag{9.24}$$

where $d_{m_l,i}$ and α_i represent routing cost and action cost.

9.5.2 Routing and robot selection

A MILP model is used to model the routing and robot selection problem for an individual cluster, having a significantly smaller complexity than the MILP model in Section 9.4. The decision variables are significantly less as we do not have dimension r. Since the clustering already provided the allocation of tasks to robots, we can now solve the sub-problems for each robot. Specifically, at least K such problems need to be solved in this approach. These smaller problems can be solved by a MILP solver to compute (sub-)optimal solutions for the sub-problems. Moreover, for the kitting application problem

investigated in this work, the sub-problems are modeled as asymmetric TSPs making it possible to use off-the-shelf efficient TSP solvers, instead of using more general-purpose solvers to compute solutions of the downsized MILP problem presented in Section 9.4. The optimality of the overall solution will be affected by the clustering resulting from the semi-supervised K-medoids method, i.e., the solution of the asymmetric TSP will only optimize the route of the individual robots, but the task allocation is decided a priori through the clustering.

9.5.3 TSP modeling

As it is already mentioned in the sub-section above, the reduced routing and robot selection problem can be expressed as an asymmetric TSP problem. The reason for this conversion is the possibility to use efficient solvers dedicated to the TSP problem, which will have some performance advantage over a more general MILP solver. In order to use symmetric TSP solvers, e.g., Concorde [29], the asymmetric TSP is transformed into a symmetric TSP with the approach by [30]. In a TSP problem, a salesperson needs to visit all assigned cities and return to the starting point. An optimal solution will find a visiting order that minimizes the total travel distance. A presumption for the presented conversion to TSP is a limited problem instance in terms of modeled precedence constraints, where one task, i.e., the delivery task d, is preceded by all other tasks and needs to be visited last. This is in accordance with the RTSG model in Fig. 9.1, where the objects can be fetched in any order before being delivered. The TSP problem can be specified with a cost matrix, $C_{j,k} : j, k \in A^S$:

$$C_{j,k} = \begin{cases} 0 & \forall j, k \in S \\ 0 & j = d, \forall k \in S \\ M & k = d, \forall j \in S : |A| > 1 \\ M & j = d, \forall k \in A \\ M & \forall j \in A, \forall k \in S \\ \tau_{j,k} + \alpha_k & \text{otherwise, where } j, k \in A^S \end{cases} \tag{9.25}$$

where M is a value big enough to block related transitions. A solution will put all start locations in a sequence, where the last indicates the selected robot. It is followed by all the tasks where the delivery task becomes last. Thereafter, the cost matrix is modified and extended to handle subsets of alternative tasks:

For a subset representing a set of alternative tasks, $O = \{O_1, \ldots O_n\} \subseteq A$, where n is the number of tasks in a corresponding OR-pair group of the

RTSG model, one extra task $O_i^{'}$ is added for each $O_i, \forall i \in \{1, \ldots, n\}$, where $O^{'} = \{O_1^{'}, \ldots O_n^{'}\} \subseteq A \subseteq A^S$. The transition costs are arranged to make O_i represent transitions *to* this alternative task while the added $O_i^{'}$ represents transitions *from* the very same task. We indicate with $O^E = O \cup O^{'}$ and order the tasks of O and $O^{'}$ in closed loop sequences, so that $O_{i+n} = O_i, \forall i \in \{1, \ldots, n\}$ and $O_{i-n}^{'} = O_i^{'}, \forall i \in \{1, \ldots, n\}$. For each subset O, the related elements of the cost matrix are defined as:

$$C_{j,k} = \begin{cases} \tau_{j,k} + \alpha_k & \forall j \in A^S \setminus O^E, \forall k \in O \\ M & \forall j \in O, \forall k \in A^S \setminus O^E \\ \tau_{j,k} + \alpha_k & \forall j \in O^{'}, \forall k \in A^S \setminus O^E \\ M & \forall j \in A^S \setminus O^E, \forall k \in O^{'} \\ 0 & j = O_i, k = O_{i+1}, \forall i \in \{1, \ldots, n\} \\ 0 & j = O_i, k = O_i^{'}, \forall i \in \{1, \ldots, n\} \\ 0 & j = O_i^{'}, k = O_{i-1}^{'}, \forall i \in \{1, \ldots, n\} \\ M & \text{otherwise: } j, k \in O^E \end{cases} \tag{9.26}$$

where M is a value big enough to block related transitions. Within a solution of the TSP, all the tasks of a set O^E become grouped in a sub-sequence: $(O_i, O_{i+1}, \ldots, O_{i+n}, O_{i+n}^{'}, \ldots, O_{i+1}^{'}, O_i^{'})$ where $i \in \{1, \ldots, n\}$. Only the outer task pair, $O_i, O_i^{'}$, contributes a cost to the solution, while the inner transitions have zero cost. In turn, the outer task pair indicates the selected alternative task, and the intermediate tasks are removed from the solution. All sets of alternative tasks that were distributed over multiple clusters will remain with one task in each computed schedule. In this step, all of them are removed and bypassed, except the one whose removal causes the smallest cost saving.

9.5.4 Balancing

The balancing step is a local search algorithm, detailed in Algorithm 5. At each iteration, a task is selected and moved from the schedule with the highest cost into another sequence that does not increase the maximum cost. The selected task may not have a separation constraint from tasks in the receiving sequence. The selection criteria of the task, the receiving schedule, and the predecessor in the receiving schedule is critical to avoid sub-optimization. E.g., a prioritization of the largest overall cost reduction will often move tasks to the least expensive sequence, even if other scheduled sequences are much closer. Another important aspect is to minimize the intersections between different routes. Our selection criterion is based on maximizing a gain vs loss

ratio, where the gain is the cost reduction of the sending sequence and the loss is the cost increase of the receiving sequence.

9.5.5 Algorithmic overview

A pseudo-code for the heuristic approach is given in Algorithm 4. The main function COMPUTESCHEDULES, takes as input arguments the set of tasks (A), the set of robot start locations (S), and the number of schedules (k) to be computed. First, the clusters are computed using the function *ComputeClusters*. Thereafter, a schedule and a robot selection are computed for each cluster, using the TSP model in Section 9.5.3. Since there is a chance that a group of multiple schedules may select the same robot, the schedule in the group with the highest cost will be used, while the other schedules are recalculated without this robot available. The worst case for this conflict resolution approach is $K(1+K)/2$ schedule computations, which may cause a planning efficiency problem if the given robot team size, K, is large. In the next step, redundant alternative tasks are removed. Finally, the schedules are balanced with Algorithm 5, $BalanceSchedules$.

9.6 Experiments

The goal of the experiments is to compare the proposed heuristic planning approach with a MILP solver with respect to plan quality and planning performance. In our experiments, the operational area where the missions are planned is 40×50 m, containing 468 task locations and 72 robot locations. For each experiment, a mission work description is generated in the form of the RTSG model in Figure 9.1. The model is populated with a randomly selected subset of the tasks in the operational area, where the number of tasks is a controlled parameter. 20% of the tasks are modeled to be alternative, partitioned in randomly composed alternative sets with 2-4 tasks in each set. Delivery tasks are co-located. Eight randomized robot locations indicate possible robot team members to be selected, where the team size is a controlled parameter. Routing distances are Euclidian and routing costs, i.e., routing times, are estimated with a robot speed of 0.5m/s. Task action durations are randomized in the interval 5-15s. The *makespan* is minimized by the two compared planning approaches, while *planning time* is a corresponding performance measure. 10 experiments were run for each combination of team size and the number of tasks. The planning time was limited to a maximum of 30 minutes for the MILP solver. It can be noted that the problem is NP-hard, since it generalizes the NP-hard TSP problem [11].

Algorithm 4 Clustering-based heuristic approach.

```
function ComputeSchedules(A,S,k)
  CL ← ComputeClusters(A, k)                          ▷ Clusters
  SC ← ∅                                              ▷ Schedules
  AR ← S                                              ▷ Available robots
  for all c ∈ CL do
    s ← Schedule()                                    ▷ New empty schedule
    s.cl ← c                                          ▷ Schedule has cluster c
    s.seq ← ∅                                         ▷ Schedule has no sequence yet
    s.robot ← ∅                                       ▷ Schedule has no robot yet
    SC ← SC ∪ s
  end for
  while |S| − |AR| < k do
    for all s ∈ SC do
      if s.robot = ∅ ∨ s.robot ∉ AR then
        s.robot, s.seq ← ComputeSchedule(s.cl, AR)
      end if
    end for
    for all s ∈ SC do
      if s.robot ∈ AR then
        y ← x ∈ SC : x.robot = s.robot, max_x x.cost
        AR ← AR \ s.robot
      end if
    end for
  end while
  SC ← ReduceAlternativeTasks(SC)
  SC ← BalanceSchedules(SC)
  return SC
end function
```

The generated plans for one such experiment is illustrated in Figure 9.2, where the diagrams indicate $2D$ locations in the operational area. Units of axes are meter. The heuristic solution is visualized in the upper diagram and the MILP solver solution in the lower diagram. Available robots are marked with pink X-markers and tasks are marked with O-markers. Each route starts from a selected robot (an X-marker) and ends with one of the black-colored and co-located delivery tasks ($d01$,$d02$,$d03$). Action durations are indicated by the size of the O-markers. The color of non-selected alternative tasks is grey, i.e., $a07$ and $a17$ for the heuristic solution, and $a04$ and $a15$ for the MILP solution. For the heuristic approach, the medoids of the clusters are highlighted in orange and the colors of planned tasks indicate their clusters. The balancing step has moved two red tasks ($a1$,$a13$) from the red route to the

green route.

The experiments were run on an Intel i5-4570 with 8 GB of RAM and Ubuntu 20.04.5 operating system. Gurobi [31] was used to compute MILP solutions. For the heuristic approach, Concorde [29] was used to compute TSP solutions while the remaining algorithmic steps, e.g., clustering, were implemented in Python.

Algorithm 5 Balancing of schedules

```
function BALANCESCHEDULES(SC)
  SC.SORT()                                   ▷ Sort in order of decreasing routing cost
  while true do
    costImproving ← false
    maxRatio ← 0
    for all t ∈ SC[0].seq do                  ▷ tasks in the longest route
      for all s ∈ SC \ SC[0] do
        for all p ∈ s.seq do
          if NOSEPARATIONCONSTR(p, SC[0].seq) then
            loss ← CALCLOSS(s.seq, t, p)
            if s.seq.cost + loss < SC[0].seq.cost then
              gain ← CALCGAIN(SC[0].seq, t)
              ratio ← gain/loss
              if ratio > maxRatio then
                costImproving ← true
                maxRatio ← ratio
                task ← t
                predecessor ← p
                receiver ← s
              end if
            end if
          end if
        end for
      end for
    end for
    if costImproving then                     ▷ Move task into shorter route
      receiver.INSERT(task, predecessor)
      SC[0].REMOVE(task)
      SC.SORT()                               ▷ Update the sorting for next iteration
    else
      return SC
    end if
  end while
end function
```

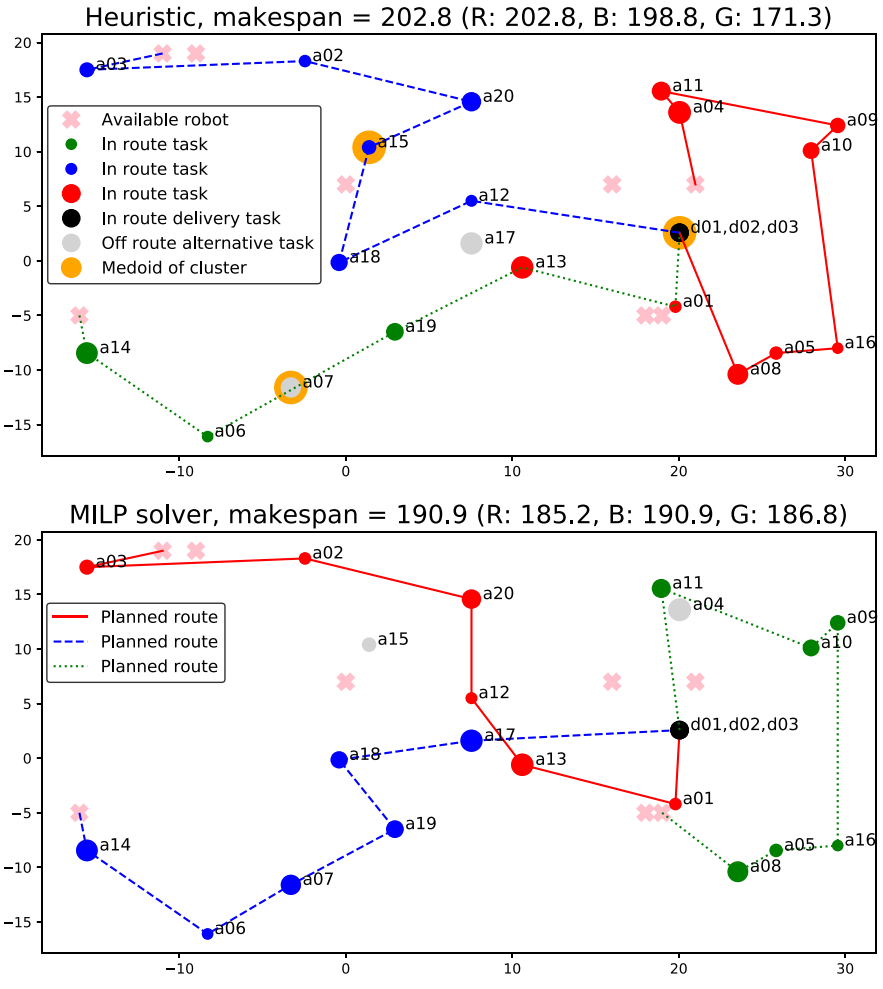

Figure 9.2: Illustrated example of computed problem solutions.

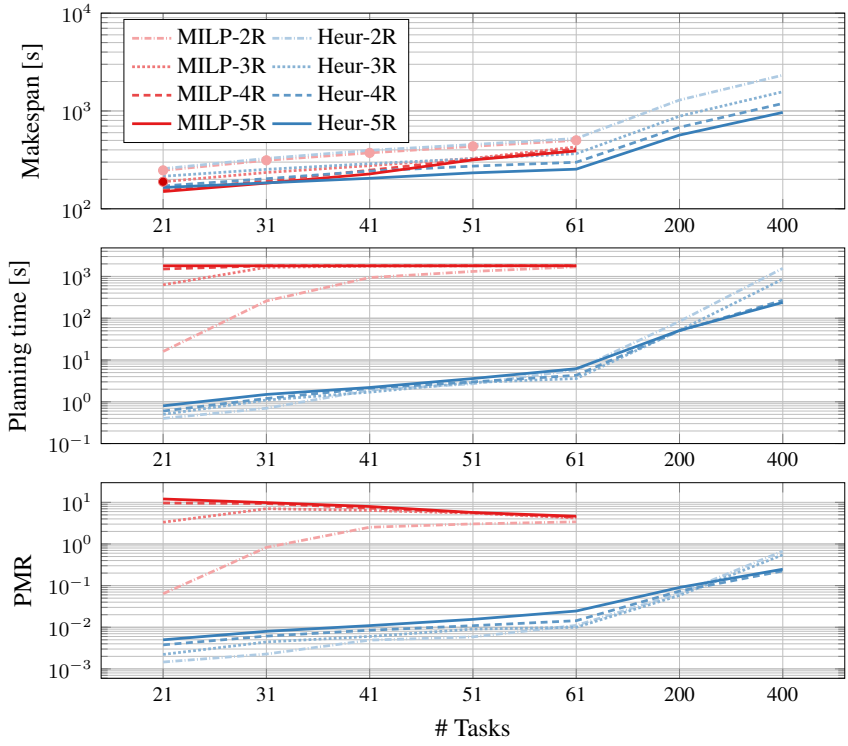

Figure 9.3: Makespan and planning time for different sizes of robot teams, displayed with logarithmic scales to provide a compact view.

A comparison of the heuristic approach and the MILP solver is found in Fig. 9.3. The horizontal axes indicate the number of tasks. In the top graph, the average computed makespan is given for the MILP solver and for the heuristic approach for different team sizes. For the MILP solver, a few makespan values are marked with a red dot. They indicate problem instances where the MILP solver was able to find optimal solutions for at least 5 out of 10 runs. To give a quantitative indication of the optimality of the heuristic approach, only the runs with optimal MILP solver solutions are included in the data set for these problem instances. The middle graph indicates the average planning time, and the bottom graph gives the Planning-time to Makespan Ratio (PMR).

The heuristic approach generated solutions with a slightly higher makespan than the MILP solver for the smallest problem instances, while performing better than the MILP solver for the larger problem instances. The heuristic approach found solutions within a few seconds, while the MILP solver delivered an optimal solution in less than 30 minutes, or a sub-optimal solution at the 30 minute timeout. The PMR comparison in the bottom graph indicates that the MILP solver in general uses significantly more time for producing a plan compared to the makespan of that plan. This implies the MILP solver is a less suitable approach, especially in online planning scenarios where the planning time may have a direct impact on the mission time. On the other hand, the heuristic approach has a planning time that is a fraction of the makespan. With 200 tasks, the planning time is still reasonable with PMR $< 10\%$. Some quantitative data of the solution optimality for the heuristic approach can be indicated with a Makespan Optimality Ratio (MOR):

$$MOR = \frac{Makespan_{heur}}{Makespan_{opt}} \tag{9.27}$$

$MOR \geq 1$, where a value of 1 indicates an optimal solution. MOR of the heuristic approach can be evaluated for the problem instances with optimal MILP solver solutions, i.e., $Makespan_{opt} = Makespan_{MILP}$. For problem sizes with two robots, MOR was $104\%, 105\%, 106\%, 104\%$ and 104% for $21, 31, 41, 51$ and 61 tasks, respectively. For 3 robots, MOR was 113% for 21 tasks. These are all sub-optimal solutions, but indicate an acceptable gap to optimality, especially when considering the superior planning time compared to the MILP solver. However, the evaluation is quantitative and we do not provide a guarantee on the optimality. For the largest problem instances with 200 and 400 tasks, the MILP solver was unable to find any feasible solution in the given time. With 400 tasks, the planning time of the heuristic approach increases with smaller robot teams. This is caused by TSP computations becoming the dominating sub-problem, where a smaller team size scales up the

size of the TSP problems.

9.7 Conclusion

We have investigated a novel heuristic approach to select and plan a multi robot team for an industrial kitting application modeled with a Robot Task Scheduling Graph. It is benchmarked against a MILP model implemented in Gurobi, that is able to generate optimal solutions for smaller problem instances. The experiments confirm an ability to generate high quality solutions within a few seconds, i.e., a fraction of the time required by the MILP solver. Additionally, solutions can be generated within reasonable time for significantly scaled up problem instances.

Future extensions of this work may investigate, e.g., Cross-Schedule Dependencies and mission planning in a dynamic environment.

Bibliography

[1] Alaa Khamis, Ahmed Hussein, and Ahmed Elmogy. Multi-robot task allocation: A review of the state-of-the-art. *Cooperative robots and sensor networks 2015*, pages 31–51, 2015.

[2] George B Dantzig and John H Ramser. The truck dispatching problem. *Management science*, 6(1):80–91, oct 1959.

[3] Grigorios D Konstantakopoulos, Sotiris P Gayialis, and Evripidis P Kechagias. Vehicle routing problem and related algorithms for logistics distribution: A literature review and classification. *Op. Res.*, 22(3):2033–2062, sep 2020.

[4] Rahma Lahyani, Mahdi Khemakhem, and Frédéric Semet. Rich vehicle routing problems: From a taxonomy to a definition. *Europ. J. Op. Res.*, 241(1):1–14, feb 2015.

[5] Omar Cheikhrouhou and Ines Khoufi. A comprehensive survey on the multiple traveling salesman problem: Applications, approaches and taxonomy. *Computer Science Review*, 40:100369, may 2021.

[6] Jairo R Montoya-Torres, Julián López Franco, Santiago Nieto Isaza, Heriberto Felizzola Jiménez, and Nilson Herazo-Padilla. A literature review on the vehicle routing problem with multiple depots. *Computers & Ind. Eng.*, 79:115–129, jan 2015.

[7] Unes Bahalke, Nima Hamta, Amir Reza Shojaeifard, Maryam Alimoradi, and Samira Rabiee. A new heuristic algorithm for multi vehicle routing problem with and/or-type precedence constraints and hard time windows. *Op. Res. in Eng. Sciences: Theory & Applications*, 5(2):28–60, aug 2022.

[8] G Ayorkor Korsah, Anthony Stentz, and M Bernardine Dias. A comprehensive taxonomy for multi-robot task allocation. *The Int. J. Rob. Res.*, 32(12):1495–1512, oct 2013.

[9] Andreas Bortfeldt and Gerhard Wäscher. Constraints in container loading: A state-of-the-art review. *Europ. J. Op. Res.*, 229(1):1–20, aug 2013.

[10] Branko Miloradović, Baran Cürüklü, Mikael Ekström, and Alessandro V Papadopoulos. A genetic algorithm approach to multi-agent mission planning problems. In *Int. Conf. Op. Res. and Enterp. Syst.*, pages 109–134, 2019.

[11] K Ilavarasi and K Suresh Joseph. Variants of travelling salesman problem: A survey. In *Int. Conf. on Inf. Comm. and Emb. Syst. (ICICES)*, pages 1–7, feb 2014.

[12] Anders Lager, Alessandro Papadopoulos, Giacomo Spampinato, and Thomas Nolte. A task modelling formalism for industrial mobile robot applications. In *Int. Conf. on Adv. Rob. (ICAR)*, dec 2021.

[13] Asvin Goel and Volker Gruhn. A general vehicle routing problem. *Europ. J. Op. Res.*, 191(3):650–660, dec 2008.

[14] Neil Mathew, Stephen L Smith, and Steven L Waslander. Multirobot rendezvous planning for recharging in persistent tasks. *IEEE Trans. Robotics*, 31(1):128–142, feb 2015.

[15] Hicham Touzani, Hicham Hadj-Abdelkader, Nicolas Séguy, and Samia Bouchafa. Multi-robot task sequencing & automatic path planning for cycle time optimization: Application for car production line. *IEEE Rob. & Autom. Lett.*, 6(2):1335–1342, apr 2021.

[16] Mina Roohnavazfar, Seyed Hamid Reza Pasandideh, and Roberto Tadei. A hybrid algorithm for the vehicle routing problem with and/or precedence constraints and time windows. *Computers & Op. Res.*, 143:105766, jul 2022.

[17] Branko Miloradović, Mirgita Frasheri, Baran Cürüklü, Mikael Ekström, and Alessandro Vittorio Papadopoulos. Tamer: Task allocation in multi-robot systems through an entity-relationship model. In *Int. Conf. Principles and Practice of Multi-Agent Systems*, pages 478–486, 2019.

[18] Anil K Jain. Data clustering: 50 years beyond k-means. *Pattern Recognition Letters*, 31(8):651–666, jun 2010.

[19] Hae-Sang Park and Chi-Hyuck Jun. A simple and fast algorithm for k-medoids clustering. *Expert systems with applications*, 36(2):3336–3341, 2009.

[20] Rodrigo Randel, Daniel Aloise, Nenad Mladenović, and Pierre Hansen. On the k-medoids model for semi-supervised clustering. In *Variable Neighborhood Search*, pages 13–27, 2019.

[21] Asma Ayari and Sadok Bouamama. ACD3GPSO: automatic clustering-based algorithm for multi-robot task allocation using dynamic distributed double-guided particle swarm optimization. *Assem. Autom.*, 40(2), sep 2019.

[22] Xiaolong Xu, Hao Yuan, Mark Liptrott, and Marcello Trovati. Two phase heuristic algorithm for the multiple-travelling salesman problem. *Soft Computing*, 22(19):6567–6581, jul 2017.

[23] Elango Murugappan, Nachiappan Subramanian, Shams Rahman, Mark Goh, and Hing Kai Chan. Performance analysis of clustering methods for balanced multi-robot task allocations. *Int. J. Prod. Res.*, 60(14):4576–4591, aug 2021.

[24] Gilbert Laporte and Frédéric Semet. Computational evaluation of a transformation procedure for the symmetric generalized traveling salesman problem. *INFOR: Information Systems and Op. Res.*, 37(2):114–120, may 1999.

[25] Brian P Gerkey and Maja J Matarić. A formal analysis and taxonomy of task allocation in multi-robot systems. *The Int. Journal of Robotics Research*, 23(9):939–954, sep 2004.

[26] Ernesto Nunes, Marie Manner, Hakim Mitiche, and Maria Gini. A taxonomy for task allocation problems with temporal and ordering constraints. *Rob. & Aut. Syst.*, 90:55–70, apr 2017.

[27] Sugato Basu, Ian Davidson, and Kiri Wagstaff. *Constrained clustering: Advances in algorithms, theory, and applications*. CRC Press, 2008.

[28] Mauricio G. C. Resende and Renato F. Werneck. A fast swap-based local search procedure for location problems. *Annals of Op. Res.*, 150(1):205–230, jan 2007.

[29] David L Applegate, Robert E Bixby, Vašek Chvátal, and William J Cook. The traveling salesman problem. dec 2011.

[30] Roy Jonker and Ton Volgenant. Transforming asymmetric into symmetric traveling salesman problems. *Op. Res. Letters*, 2(4):161–163, nov 1983.

[31] LLC Gurobi Optimization. Gurobi optimizer reference manual, 2021.

Chapter 10

Paper D
Risk Aware Planning of Collaborative Mobile Robot Applications with Uncertain Task Durations

Anders Lager, Branko Miloradović, Giacomo Spampinato, Thomas Nolte and Alessandro V. Papadopoulos. In 33rd IEEE International Conference on Robot and Human Interactive Communication (RO-MAN), 2024.

Abstract

The efficiency of collaborative mobile robot applications is influenced by the inherent uncertainty introduced by humans' presence and active participation. This uncertainty stems from the dynamic nature of the working environment, various external factors, and human performance variability. The observed makespan of an executed plan will deviate from any deterministic estimate. This raises questions about whether a calculated plan is optimal given uncertainties, potentially risking failure to complete the plan within the estimated timeframe. This research addresses a collaborative task planning problem for a mobile robot serving multiple humans through tasks such as providing parts and fetching assemblies. To account for uncertainties in the durations needed for a single robot and multiple humans to perform different tasks, a probabilistic modeling approach is employed, treating task durations as random variables. The developed task planning algorithm considers the modeled uncertainties while searching for the most efficient plans. The outcome is a set of the best plans, where no plan is better than the other in terms of stochastic dominance. Our proposed methodology offers a systematic framework for making informed decisions regarding selecting a plan from this set, considering the desired risk level specific to the given operational context.

10.1 Introduction

Integrating collaborative robot applications in the industrial landscape started over a decade ago [1]. While robotic utilization enhances productivity and ergonomic conditions by managing assistive, repetitive, and strenuous tasks, the contemporary industrial trend towards mass customization values the unique skills, adaptability, dexterity, and problem-solving capabilities of human workers [2]. Our paper specifically focuses on mobile robots in collaborative applications, appreciating their flexibility to execute diverse tasks across various locations, catering to the needs of human workers [3]. In collaborative industrial environments characterized by semi-structured and dynamic settings, the duration of robot routing can be influenced by temporary obstacles and concurrent human and robot activities. Additionally, uncertainties in robot task durations arise from unpredictable placements of required parts or tools and the possibility of task failures, leading to retries. Similarly, the duration of human tasks introduces uncertainty, influenced by variables such as workload, fatigue, availability, and location.

This paper aims to study the uncertainty associated with the duration of tasks performed by robots and humans, affecting collaborative plans' accuracy. Traditional deterministic planning methods fail to account for potential deviations from the estimated duration of tasks. Additionally, identifying an optimal plan is difficult due to the inherent uncertainty in task durations and the subjectivity involved in choosing the "best" plan, which can impact the risk tolerance of a human planner. For example, a plan with a given probability of completing the tasks within a certain time limit may be preferred over a plan with a guaranteed upper bound for the makespan. While the former may result in a lower makespan, it can sometimes lead to a higher makespan than the upper bound of the latter. Taking a medium risk may reduce the expected makespan over multiple runs. Taking a larger risk can be motivated if a low makespan gives a reward, whereas the distinction between a shorter or longer additional delay might not be crucial. This paper investigates industrial task planning problems for a mobile robot collaborating with humans in dynamic environments. To model the planning problem, we use Robot Task Scheduling Graph (RTSG) [4]. This is motivated by the intuitiveness of the representation rather than its expressiveness, benefiting domain experts who have broad knowledge of the application but no in-depth knowledge of all system parts, e.g., robot programming. This paper extends the RTSG model to enhance the flexibility of work descriptions by considering concurrent human tasks and robot tasks with inter-dependencies. The stochastic modeling approach presented in this paper introduces uncertainty into task and routing durations

using random variables with unrestricted probability distributions. When concrete data is lacking, initial simplifications of input distributions, such as uniform distributions, are employed. These distributions can be iteratively refined through data collection during system execution, progressively improving accuracy. The main contribution is a novel task planning methodology involving computing optimized plans where uncertainties are considered during plan generation. The output is a set containing the best plans, where no plan is better than the other in terms of stochastic dominance. Notably, the methodology allows a human planner to make an informed decision on the most suitable plan from this candidate set by providing a risk level value and/or by visually inspecting probabilistic makespan distributions of the candidates. This approach is inspired by Mixed-Initiative Planning [5]. The novel task planning methodology leverages a Branch-and-Bound (B&B) algorithm [6] able to solve planning problems modelled with RTSG, extended to handle stochastic durations. It explores alternative sub-sequences, with their durations derived as probability density functions. One contribution is the derivation of unrestricted probability distributions to represent makespans of collaborative plans. Such a distribution is a realistic estimate of the makespan range that may occur if the plan is executed. Another contribution is a novel pruning strategy, which uses the first-order stochastic dominance property to ensure safe pruning. It guarantees that pruning never eliminates a superior plan in a stochastic sense. Additionally, we contribute by proposing *stochastic set dominance* as a criterion to filter full-length plans into a candidate set containing only the very best plans, where no plan's makespan dominates others in a stochastic sense. To validate the approach, it is benchmarked against a deterministic counterpart. Additionally, Monte Carlo simulations [7] are conducted to verify the correctness of stochastic makespan computations generated by the planning algorithm. These evaluations demonstrate the ability of our methodology to provide more efficient plans and to support risk-aware task planning under uncertain conditions.

10.2 Related work

In a recent approach for task planning in collaborative assembly applications, a policy for task allocation based on the current state is trained to optimize future rewards [8]. This work and all other related works differs from ours, by not providing a set of plans optimized for different risk levels. Our approach accounts for uncertainties in the planning problem, while some other approaches solve a deterministic planning problem and handle uncertainties during execution [9] [10], potentially providing a suboptimal plan. Probabilis-

tic Simple Temporal Networks (PSTN) are used to model scheduling problems with temporal constraints using random variables to represent uncertain task durations [11]. In general, PSTN addresses *scheduling risks* by searching for robust plans to minimize or control the risk for plan execution failures caused by violation of temporal constraints, e.g., missed deadlines. The problem addressed in this paper has no temporal constraints, and the type of risk addressed is different, i.e., the risk of getting a longer duration than expected at plan execution. In some works, the primary focus of a robot is assisting by anticipating the next human task and providing needed tools or parts just in time [12, 13]. In our work, the robot serves multiple humans, and a long-sighted sequence of robot actions is planned. One scheduling approach used Monte Carlo simulations in a receding horizons approach to estimate the cost distributions of alternative execution sequences of robot tasks and human tasks with uncertain durations [14]. A receding horizons approach naturally limits the growth of a search tree to a manageable level. For our problem type, where the objective is to minimize the makespan, a plan may become greedy and less optimal with a short-sighted look ahead, causing later costs to dominate. Additionally, we compute cost distributions in a closed form, giving a qualitative advantage when comparing alternative plans and sub-sequences, including the possibility of safe pruning decisions to reduce the growth of a search tree. In a receding horizon scenario similar to [14], the uncertainties in durations of human and robot tasks were modeled and propagated for alternative task sequences as triangular fuzzy sets, which in an actual application provided a better plan optimization than an approach where task durations were modeled more simplistic, as uniform distributions [15]. This result motivates the usage of richer representations, as proposed in this paper, to represent uncertain durations when modeling collaborative applications. Similar to our approach, [16] models routing and action durations as random variables and uses a framework of stochastic operations to compute stochastic start and completion times of tasks for different scenarios where multiple robots share a mutually exclusive resource. However, probability distributions are limited to be Gaussian, while our approach does not impose such restrictions. In [17], this Gaussian framework was applied to a task planning problem of a replenishment agent serving other agents, where a finite-horizon schedule of tasks is computed with a B&B approach, including pruning of branches where a conservative estimate of the minimum cost is higher than currently the best solution. We propose a novel pruning strategy based on stochastic dominance, proven to be safe for this application when comparing *unrestricted* distributions of search nodes representing the same state. A proactive scheduling approach for a Job shop problem was proposed by [18], using durations modeled as random variables. As a part

of the solution, the sequence of operations on a machine was computed with a B&B algorithm, minimizing a weighted combination of expectation and variance of the completion time. A safe pruning criterion was proposed, using Stably Stochastic Dominance, providing an ordering of alternative sequences based on expectation and variance of operations. While this approach is motivated by the need to find a robust plan with limited time variations, our approach computes a candidate set of plans, whose makespans are stochastically dominated by alternative plans, thereby providing the most *efficient* choices for any risk level.

10.3 Modeling the planning problem

In this section, we define and model the planning problem. We present the modeling of task durations as random variables and provide related definitions for later reference.

10.3.1 Problem description and assumptions

A mobile industrial robot is used to deliver parts to different assembly workstations. At these stations, sub-assemblies are made either by human workers or by the robot itself. The robot fetches the finished sub-assemblies, and finally, all sub-assemblies are delivered to another station for further processing. It is assumed there is no load restriction related to the mobile robot's ability to carry parts and sub-assemblies. Task allocation is static, i.e., there are robot tasks and human tasks. While considering the dependencies between robot tasks and human tasks, we assume human tasks are mutually independent. There is no physical interaction between the robot and humans and their interaction level can be categorized as synchronized, i.e., they share the same space at the delivery and fetching locations, but not at the same time [19]. The effects of tasks are assumed to be deterministic, i.e., all tasks will eventually succeed. However, task and routing durations are uncertain. The goal is to compute an offline plan [20] that minimizes the uncertain makespan while considering the aversion or willingness to risk. In this context, the risk is related to efficiency. Increasing the risk means we increase the probability of reaching shorter makespans while also accepting the risk of sometimes reaching longer makespans than before. The risk willingness may influence what plan is considered to be the best.

10.3.2 Modeling a collaborative planning problem

The planning problem is modeled with an RTSG model, exemplified in Fig. 10.1. It is a directed acyclic graph giving an intuitive workflow description of how tasks can be sequenced. The *Start* node and the *Goal* node represent the initial state and the desired goal state, respectively. Task nodes have rectangular shapes. Edges (or paths) represent precedence constraints, e.g., $DL1$ must precede $MV1$ while $DL1$ and $DL2$ may be performed in any order. Alternative branches are modeled between OR-Fork (||F) and OR-Join (||J) node pairs, and in this model, they describe the alternative selections of human assembly ($HA1$) or robot assembly ($RA1$). Lock node pairs (+L,-L) indicate a branch section where robot tasks must be scheduled in an uninterrupted sequence. Here, the robot movement to a human assembly station ($MV1$) must be followed by the robot picking ($PI1$) the sub-assembly provided by the human ($HA1$). AND-Fork nodes (&F) create parallel branches while AND-Join nodes (&J) join branches. In this work, the RTSG modeling formalism has been extended to represent *human tasks*, i.e., tasks allocated to humans (e.g., $HA1$). Human tasks may be performed concurrently with robot tasks, and they need to follow the scheduling constraints set by the RTSG model. Additionally, the need to synchronize human tasks with robot tasks is addressed. A new node type, AND-JoinSync (&JS), has multiple incoming branches and a single outgoing branch. The &JS node blocks the robot from proceeding with succeeding robot tasks until all preceding human tasks are completed, potentially causing some robot wait time. For example, after moving to an assembly station ($MV1$), the robot may not pick the sub-assembly ($PI1$) until the human assembly task ($HA1$) has provided the assembly.

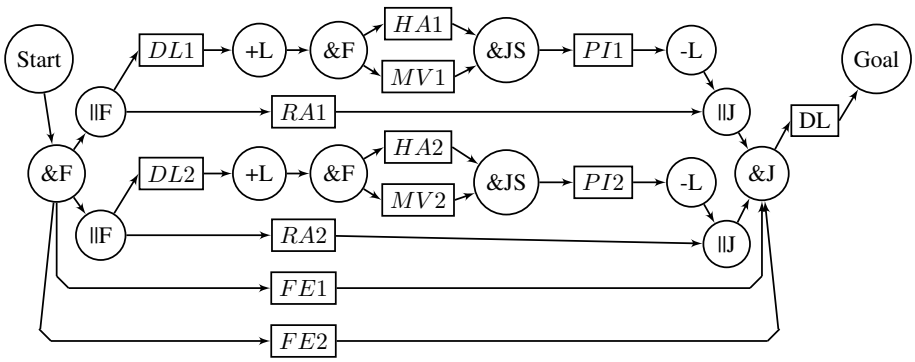

Figure 10.1: RTSG model with 2 human assembly tasks.

10.3.3 Preliminaries and definitions

Durations for performing robot tasks and human tasks are indexed variables. The routing duration $R_{\tau,\tau'}$ between robot tasks, $\tau, \tau' \in T$ where T is the set of all robot tasks. The duration A_τ to perform a robot task $\tau \in T$ and the duration B_h to perform a human task $h \in H$, where H is the set of all human tasks. This work models these durations (B_h, A_τ, $R_{\tau,\tau'}$) as independent random variables without assuming any specific probabilistic distributions.

Definition 1 (Random Variable)**.** *A random variable X on the probability space $(\Omega, \mathcal{F}, \mathbb{P})$ is a measurable function $X : \Omega \to \mathbb{R}$ such that $\{\omega \in \Omega : X(\omega) = x\} \in \mathcal{F}$ for all $x \in \mathbb{R}$.*

Definition 2 (Expected value)**.** *Given a random variable X, its* expected value *$\mathbb{E}[X] \in \overline{\mathbb{R}}$ is a measure of the central tendency or average value of the possible outcomes of X:*

$$\mathbb{E}[X] \triangleq \int_{\omega \in \Omega} X(\omega)\, d\mathbb{P}.$$

Definition 3 (Variance)**.** *Given a random variable X, its* variance*, denoted by $\mathbb{V}[X]$, is a measure of the dispersion or spread of the possible outcomes of X.*

$$\mathbb{V}[X] \triangleq \mathbb{E}\Big[(X - \mathbb{E}[X])^2\Big]$$

Definition 4 (Standard deviation)**.** *The standard deviation $\sigma[X]$ of a random variable X is the square root of its variance:*

$$\sigma[X] \triangleq \sqrt{\mathbb{V}[X]}.$$

Definition 5 (Probability density function (PDF))**.** *The probability density function $f_X(x)$ of a random variable X is defined as:*

$$f_X(x) = \mathbb{P}[\omega \in \Omega \mid X(\omega) = x]$$

Definition 6 (Cumulative distribution function (CDF))**.** *The cumulative probability distribution function $F_X(x)$ of a random variable X is defined as:*

$$F_X(x) = \mathbb{P}[\omega \in \Omega \mid X(\omega) \leq x]$$

Definition 7 (Percentile)**.** *The k-th percentile of a probabilistic distribution $f_X(x)$ is defined as:*

$$p_k = \inf\{x : F_X(x) \geq k\}, \quad 0 < k < 1.$$

Definition 8 (First-Order Stochastic Dominance [21])**.** *Consider two random variables, X and Y, with CDFs F_X and F_Y. X has a first-order stochastic dominance over Y, if and only if $\forall x,\ F_X(x) \leq F_Y(x)$, and $\exists x,\ F_X(x) < F_Y(x)$. The stochastic dominance is denoted in the following as $X \succeq Y$. If the given condition is not fulfilled, this is denoted $X \not\succeq Y$.*

Definition 9 (Stochastic Set Dominance)**.** *Consider one random variable X and a set of random variables $S = \{Y_1, \ldots, Y_a\}$, with CDFs F_X and $F_{Y_1}, \ldots, F_{y_a}$. X has a set dominance over S, if and only if $\forall x,\ F_X(x) \leq \max\{F_{Y_1}(x), \ldots, F_{Y_a}(x)\}$, and $\exists x,\ F_X(x) < \max\{F_{Y_1}(x), \ldots, F_{Y_a}(x)\}$. Stochastic set dominance is denoted in the following as $X \succeq_{SSD} S$. If the given condition is not fulfilled, this is denoted $X \not\succeq_{SSD} S$. Stochastic set dominance does not require but follows from first-order stochastic dominance, i.e., if $\exists Y \in S \mid X \succeq Y \implies X \succeq_{SSD} S$.*

Definition 10 (Independence)**.** *Two random variables X and Y are independent if the pair of events $\{X = x\}$ and $\{Y = y\}$ are independent for all $x, y \in \mathbb{R}$. Formally,*

$$\mathbb{P}[X = x, Y = y] = \mathbb{P}[X = x]\mathbb{P}[Y = y], \quad \forall\, x, y \in \mathbb{R}.$$

Definition 11 (Convolution or sum of random variables)**.** *If X and Y are independent random variables on $(\Omega, \mathcal{F}, \mathbb{P})$, then $Z = X + Y$ has probability density function, when X and Y are discrete random variables*

$$\mathbb{P}[Z = z] = \sum_{x=-\infty}^{\infty} f_X(x) f_Y(z - x), \quad \forall\, z \in \mathbb{Z}, \tag{10.1}$$

and for continuous random variables:

$$\mathbb{P}[Z = z] = \int_{-\infty}^{\infty} f_X(x) f_Y(z - x)\, \mathrm{d}x. \tag{10.2}$$

Lemma 10.3.1 (Maximum between random variables)**.** *If X and Y are independent random variables on $(\Omega, \mathcal{F}, \mathbb{P})$, then $Z = \max(X, Y)$ has cumulative probability function*

$$F_Z(z) = F_X(z) F_Y(z)$$

Proof. By definition of CDF, we have that:

$$\begin{aligned} F_Z(z) &= \mathbb{P}[\max(X, Y) \leq z] = \mathbb{P}[X \leq z \wedge Y \leq z] \\ &= \mathbb{P}[X \leq z]\mathbb{P}[Y \leq z] \ X \text{ and } Y \text{ are independent} \\ &= F_X(z) F_Y(z). \end{aligned}$$

□

10.4 Planning methodology

This section gives a step-by-step description of the planning methodology. Sec. 10.4.1 defines a feasible plan and identifies dependencies between robot tasks and human tasks. Sec. 10.4.2 derives the stochastic duration of a plan or sub-sequence. Sec. 10.4.3 introduces a B&B algorithm to search for candidate plans having the shortest durations in a stochastic sense, while Sec. 10.4.4 describes risk-aware plan selection from this set. Sec. 10.4.5 presents a pruning strategy based on stochastic dominance and proves this strategy is safe.

10.4.1 Plan feasibility and dependencies with human tasks

A plan is the set of all *robot* tasks in the RTSG model except those in non-selected alternative branches, ordered in a feasible sequence from the start node to the goal node. A feasible plan must fulfill the constraints imposed by the RTSG model (see Sec. 11.2). For the model in Fig. 10.1, a feasible plan is exemplified by ($Start$, $FE2$, $DL1$, $RA2$, $MV1$, $PI1$, $FE1$, DL, $Goal$). It does not violate precedence or lock constraints. The completion of the plan depends on the human task $HA1$, which belongs to the selected alternative branch in the RTSG model starting with $DL1$. Due to precedence constraints, $HA1$ can not start, i.e., become *enabled*, until $DL1$ is completed. This makes $DL1$ the *enabling task* of $HA1$ in this plan. Due to the AND-JoinSync (&JS) node, $MV1$ is not considered completed until $HA1$ is completed. Therefore, depending on the outcome of task durations, the robot may need to wait for completion of $HA1$ after reaching $MV1$. This makes $MV1$ the *dependent task* of $HA1$. If a human task is enabled in a plan, there is always a dependent task, e.g., if the upper &JS node in Fig. 10.1 is replaced with an &J node, the goal node becomes the dependent task of $HA1$.

10.4.2 The duration of a plan

The duration of a plan is referred to as makespan. It depends on stochastic routing and task durations, $R_{\tau,\tau'}$, A_τ, B_h, of planned robot tasks and enabled human tasks. From these inputs, we derive the duration of a plan or sub-sequence as a random variable without restricting its probability distribution. To support this derivation, a few definitions are introduced: A plan is a sequence of robot tasks without element repetition, defined as $P_{0,n} = (\tau_0, \ldots, \tau_n) \subseteq T$ where τ_0 is the start state, τ_n the goal state and $\tau_1, \ldots, \tau_{n-1}$ are robot tasks. $P_{0,n}$ represents a feasible plan of the RTSG model, e.g., $\tau_0 = Start$, $\tau_1 = DL1$, $\tau_2 = RA2$, $\tau_3 = MV1$, etc. $P_{j,k}$ represents a sub-sequence from τ_j to τ_k, where the first task, τ_j, represents the start location and the following,

$\tau_{j+1}, \ldots, \tau_k$, are robot tasks to be performed. For example, $P_{1,3}$ = (τ_1, τ_2, τ_3) starts at the location of τ_1 and thereafter performs τ_2 followed by τ_3. $\mathrm{HD}(P_{0,k}, \tau_i) \subseteq H$ represents the set of human tasks whose dependent task in $P_{0,k}$ is $\tau_i \in T$, $i \leq k$. If a human task, $h \in H$, has an enabling task $\tau_p \in P_{0,k}, 0 \leq p \leq k$, then $\mathrm{EI}(P_{0,k}, h) = p$ represents the sequence index of the enabling task. Similarly, $\mathrm{DI}(P_{0,k}, h)$ represents the sequence index of the dependent task of h within $P_{0,k}$. The duration of a sub-sequence $P_{j,k}$ is indicated as $K_{j,k}$ and computed as:

$$K_{j,k} = \max(\mathcal{D}(j,k)) \tag{10.3}$$

where $\mathcal{D}(j,k)$ is a set of alternative durations between τ_j and τ_k and the $\max$ operation is defined in Lemma 10.3.1. $\mathcal{D}(j,k)$ combines the robot's sequential routing and action durations with all possible combinations of waiting for human tasks. Recursion for $\mathcal{D}(j,k)$ is given by:

$$\begin{aligned}\mathcal{D}(j,k) =& (\mathcal{D}(j,k-1) + R_{\tau_{k-1},k} + A_{\tau_k}) \cup \\ & \bigcup_{h \in \mathrm{HD}(P_{0,k},\tau_k)} (\mathcal{D}(j, \mathrm{EI}(P_{0,k}, h)) + B_h)\end{aligned}$$

where $\mathcal{D}(j,v) = \emptyset,\ \forall v < j$. The base case $\mathcal{D}(j,j) = \{C^0\}$ represents a set with one random variable having a constant value of zero. The sum operator is defined in Def. 7. The sum of a set of durations $\mathcal{D}(j,k)$ and a duration X is element wise, i.e., $\mathcal{D}(j,k) + X = \{d + X : d \in \mathcal{D}(j,k)\}$. A human task may affect $K_{j,k}$ if the enabling and dependent tasks are included in the sub-sequence. Using Eq. 10.3 is not always the most efficient way to calculate the duration. For example, if a new task is appended to a sub-sequence with a previously known duration, a total re-computation is not always necessary. By exploiting the structure of a sequence's dependencies with human tasks, the duration can often be computed by adding the durations of consecutive sub-sequences, i.e., $K_{j,k} = K_{j,l} + K_{l,k}$, $j < l < k$. A sufficient condition for this sum rule is given in Eq. 10.4. The condition requires all human tasks enabled in $P_{j,l}$, either to have no dependent tasks in $P_{l+1,k}$, or to be completed latest before the start of τ_{l+1}:

$$\bigwedge_{h \in \mathrm{HE}(P_{j,l})} \Big(\mathrm{DT}(P_{l+1,k}, h) = \emptyset \ \vee p_0(K_{\mathrm{EI}(P_{0,l},h),l}) \geq p_{100}(B_h) \Big) \tag{10.4}$$

where $\mathrm{HE}(P_{j,k}) \subseteq H$ is the set of human tasks enabled in $P_{j,k}$. $\mathrm{DT}(P_{j,k}, h) \in T$ is the dependent task for $h \in H$ in $P_{j,k}$. $\mathrm{DT}(P_{j,k}, h) = \emptyset$, if the dependent task is elsewhere or h not is enabled. For later reference, $\mathrm{HD}(P_{j,k}) \subseteq H$ is the set of human tasks whose dependent tasks are in $P_{j,k}$.

10.4.3 Extended B&B algorithm

The goal of the planning algorithm is to identify the set of plans that reaches the goal state with a minimized makespan in a stochastic sense. Our algorithm extends a previously developed deterministic B&B algorithm [6] where a breadth-first forward search from the start state towards the goal state of an RTSG model is used to compute an optimized plan. Each search node represents a unique sub-sequence, $P_{0,i}$, with a duration, $K_{0,i}$, where i is the search depth. Children are identified by searching the RTSG graph for feasible selections of the next robot task. To limit the search tree growth, a pruning selection is made among two search nodes having the same depth, $P^A_{0,i}$ and $P^B_{0,i}$, if they are considered *equivalent*, i.e., they contain the same set of tasks, Eq. 10.5, and the last task is the same, Eq. 10.6. :

$$\{\tau : \tau \in P^A_{0,i}\} = \{\tau : \tau \in P^B_{0,i}\} \tag{10.5}$$

$$\tau^A_i = \tau^B_i \tag{10.6}$$

In essence, equivalent search nodes represent the same state, but reached with different sequences. Pruning should stop exploring the search node having the *longest duration*, i.e., the pruning selection criterion. However, this criterion is unclear when applied on random variables. A naive approach would be to replace *longest duration* with *longest k-th percentile of the duration*, e.g., $p_k(K_{0,i})$, where the risk-aware planner selects the targeted k. Unfortunately, our experiments confirm this criterion is unsafe and may stop exploring potentially better full-length plans for a given k. Instead, our pruning selection criterion is based on first-order stochastic dominance, proven safe in the targeted application types in Sec. 10.4.5. When comparing alternative equivalent sub-sequences during the tree exploration, one sub-sequence can *stochastically dominate* another (see Def. 5). This means it always has a higher chance of providing a longer duration, for any k, than the other and may therefore be pruned. The outcome of the tree exploration is a set of goal-reaching plans, E. From this set, a candidate set, $Q \subseteq E$, is identified, containing the plans that do not dominate the set of other plans:

$$Q = \{P \mid P \in E \wedge K \not\succeq_{SSD} \{K' \mid P' \in E \setminus P\}\} \tag{10.7}$$

where K and K' are the makespans of P and P', respectively. Using the proposed stochastic set dominance criterion ($\not\succeq_{SSD}$) in Eq. 10.7 instead of first-order stochastic dominance ($\not\succeq$) is more stringent, hence every plan in Q will have the lowest k-th percentile, p_k, of all plans in E for at least some k, which otherwise would not be guaranteed.

10.4.4 Risk aware plan selection

It is possible to specify a desired risk level $k \in [0, 100]$, where a lower value increases the risk. From this input, a candidate plan is automatically selected from Q, having the minimum makespan reachable with a probability of k, i.e., a plan with minimum p_k. Specifying a lower k reduces p_k and the probability of making it. Changing k may also lead to selecting a different candidate plan, which is optimal for the new k. To complement the risk level, a user decision may be supported by a visual comparison of PDF curves, indicating makespan variances of the candidate plans. $|Q| = 1$ implies there is a single plan having the highest chance of providing the shortest makespan for any risk level, i.e., a *risk independent optimal plan*

10.4.5 Safe pruning method

As previously mentioned, the B&B algorithm can prune nodes at the same search depth if considered equivalent. However, the equivalence conditions, Eqs. 10.5-10.6, are not sufficient when introducing human tasks, potentially running concurrently with the last robot task. If the search tree is further explored, children nodes will sometimes include a dependent task. For these nodes, the concurrent human task may affect the plan duration differently, depending on when the human task is enabled and when the dependent task occurs. This makes a pruning decision solely based on Eqs. 10.5-10.6 unsafe. To remedy this, one additional condition for equivalence, Eq. 10.8, is introduced. This condition is fulfilled if the influence of all enabled human tasks is fully accounted for in the search node durations, $K^A_{0,i}$ and $K^B_{0,i}$. If not, equivalence is still possible for some instances, starting with identical sub-sequences up to a point where enabling any human task also includes the corresponding dependent task in the remaining sequence. The condition above is expressed as:

$$\Bigg[\bigwedge_{h \in \mathrm{HE}(P^A_{0,i})} \Big(\mathrm{DT}(P^A_{0,i}, h) \neq \emptyset \ \vee p_0(K^A_{\mathrm{EI}(P^A_{0,i},h),i}) \geq p_{100}(B_h) \Big) \wedge$$

$$\bigwedge_{h \in \mathrm{HE}(P^B_{0,i})} \Big(\mathrm{DT}(P^B_{0,i}, h) \neq \emptyset \ \vee p_0(K^B_{\mathrm{EI}(P^B_{0,i},h),i}) \geq p_{100}(B_h) \Big) \Bigg]$$

$$\vee \left[\mathrm{MI}(P^A_{0,i}) = \mathrm{MI}(P^B_{0,i}) \wedge \bigwedge_{j=0}^{\mathrm{MI}(P^A_{0,i})} \tau^A_j = \tau^B_j \right] \tag{10.8}$$

$$\text{where } \mathrm{MI}(P_{0,i}) = \min\{l \mid \mathrm{HE}(P_{l+1,i}) = \mathrm{HD}(P_{l+1,i})\} \tag{10.9}$$

By definition, condition Eq. 10.8 is true with no human tasks.

Theorem 10.4.1. *Let $P^A_{0,i}$ and $P^B_{0,i}$ be equivalent sequences and $K^A_{0,i}$ dominate $K^B_{0,i}$ in terms of a first-order stochastic dominance (FSD), in short, $K^A_{0,i} \succeq K^B_{0,i}$. Then, the duration of any plan starting with sub-sequence $P^A_{0,i}$ dominates the duration of at least one plan starting with $P^B_{0,i}$.*

Proof. Assume $P^A_{0,n} = (\tau^A_0, \ldots, \tau^A_i, \tau^A_{i+1}, \ldots, \tau^A_n)$ is a feasible plan. We define $P^B_{0,n} = (\tau^B_0, \ldots, \tau^B_i, \tau^A_{i+1}, \ldots, \tau^A_n)$. Equivalence of $P^A_{0,i}$ and $P^B_{0,i}$ implies conditions Eqs. 10.5, 10.6 hold, i.e., the set of completed tasks and the last task τ_i are the same, representing the same planning state in the RTSG model. Therefore, by having an identical continuation as $P^A_{0,n}$ from τ_{i+1}, $P^B_{0,n}$ is also a feasible plan. The proof is segmented into two distinct cases based on fulfilling the sub-conditions specified in Eq. 10.8. In case the first sub-condition in Eq. 10.8 is true, it implies condition Eq. 10.4 is also true for $P^A_{0,i}$, $P^B_{0,i}$ and we can express the duration of corresponding plans as $K^A_{0,n} = K^A_{0,i} + K^A_{i,n}$ and $K^B_{0,n} = K^B_{0,i} + K^A_{i,n}$, respectively. Thereby, both $K^A_{0,n}$ and $K^B_{0,n}$ can be defined as a monotone, increasing and continuous function of $K_{0,i}$:

$$K_{0,n} = K_{0,i} + K^A_{i,n}$$

$K^A_{0,i} \succeq K^B_{0,i} \implies K^A_{0,n} \succeq K^B_{0,n}$ in accordance with the FSD-theorem in [22].

If the second sub-condition in Eq. 10.8 is true, it implies $P^B_{0,n} = (\tau^A_0, \ldots, \tau^A_{\mathrm{MI}}, \tau^B_{\mathrm{MI}+1} \ldots, \tau^B_i, \tau^A_{i+1}, \ldots, \tau^A_n)$ where $\mathrm{MI} = \mathrm{MI}(P^A_{0,i}) = \mathrm{MI}(P^B_{0,i})$. By the definition of MI in Eq. 10.9, all human tasks enabled within $P_{\mathrm{MI},i}$ also have their dependent tasks within $P_{\mathrm{MI},i}$. Some human tasks may be enabled within $P^A_{0,\mathrm{MI}}$ and have their dependent tasks within $P^A_{i+1,n}$, thereby running concurrently with $P_{\mathrm{MI},i}$. All remaining human tasks are enabled and have their dependent tasks locally within $P^A_{0,\mathrm{MI}}$ or $P^A_{i,n}$. Considering this structure of human dependencies, the duration of plans A and B can be derived from Eq. 10.3 as:

$$K_{0,n} = \max\Big((K^A_{0,\mathrm{MI}} + K_{\mathrm{MI},i} + K^A_{i,n}) \cup \bigcup_{\substack{h \in \mathrm{HE}(P^A_{0,\mathrm{MI}}) \cap \\ \mathrm{HD}(P^A_{i+1,n})}} (K^A_{0,\mathrm{EI}(P_{0,\mathrm{MI}},h)} + B_h + K^A_{\mathrm{DI}(P_{i+1,n},h),n})\Big)$$

where $K_{\mathrm{MI},i}$ becomes included as a single summand in a single operand of the max operator. Thus, $K_{0,n}$ is a monotone (non-strictly) increasing and continuous function of $K_{\mathrm{MI},i}$. Additionally, the second sub-condition in Eq. 10.8

implies condition Eq. 10.4 so that:

$$K_{\mathrm{MI},i} = K_{0,i} - K^{A}_{0,\mathrm{MI}}$$

Thus, $K_{\mathrm{MI},i}$ is a monotone, increasing and continuous function of $K_{0,i}$. $K^{A}_{0,i} \succeq K^{B}_{0,i} \implies K^{A}_{\mathrm{MI},i} \succeq K^{B}_{\mathrm{MI},i} \implies K^{A}_{0,n} \succeq K^{B}_{0,n}$ in accordance with the FSD-theorem in [22].

□

The theorem suggests we can safely prune $P^{A}_{0,i}$, since the exploring of this search node will not result in a better plan than the best plans starting with $P^{B}_{0,i}$.

10.5 Evaluation

This section presents an experimental evaluation of the proposed planning approach, including a use case scenario for a planning problem, a deterministic benchmark approach, Monte Carlo simulations, followed by the evaluation results and their interpretation and a discussion of the outcomes.

10.5.1 Use case scenario

A planning problem, use case A, is modeled with the RTSG model in Fig. 10.1, having two human assembly tasks at different stations, where the robot delivers parts and fetches completed assemblies. Alternatively, the robot can perform one or both assemblies at robot assembly stations. Additionally, there are two robot fetch tasks at different locations. Routing and task durations are modeled with uniform distributions. Modeling resolution is 0.1 s. Assuming humans are somewhat faster than robots, their expected duration is modeled to be slightly lower but with a higher variance. An extended version of the planning problem, use case B, has five additional tasks ($DL3$, $HA3$, $MV3$, $PI3$, $RA3$) by inserting one extra assembly branch between the AND-Fork and the AND-Join node pairs in the RTSG model.

10.5.2 Deterministic benchmark approach

The rationale of the benchmark approach is to provide a deterministic version of the probabilistic approach, highlighting differences in the outcome if using *non-stochastic* values, identical with the expected values of the stochastic approach, to model routing and task durations. The deterministic approach uses the same B&B algorithm but searches for a *single* plan with minimized makespan. Sequence durations are calculated in the same way as detailed in

Algorithm 6 Monte Carlo makespan computation

function COMPUTEMAKESPAN($P_{0,n}$)
  $i \leftarrow 1$
  $D_0 \leftarrow 0$
  $B_h^{'} = B_h \quad \forall h \in \mathrm{HE}(P_{0,0})$
  while $i \leq n$ **do**
    $B = \max\{0, B_h^{'} \mid h \in \mathrm{HD}(P_{i,i})\}$
    $D_i = \max(D_{i-1} + R_{\tau_{i-1},\tau_i} + A_{\tau_i}, B)$
    $B_h^{'} = D_i + B_h \quad \forall h \in \mathrm{HE}(P_{i,i})$
    $i \leftarrow i + 1$
  end while
  return D_n
end function

section 10.4.2, but using *non-stochastic* '+' and $\max$ operators. For pruning, it uses the same extended criteria for equivalence as detailed in Eqs. 10.5, 10.6 and 10.8, but with *longest duration* instead of *stochastic dominance* as pruning selection criterion. Since stochastic dominance does not always occur, the deterministic approach can generally prune more search nodes.

10.5.3 Monte Carlo simulations

Monte Carlo simulations are used to verify the correctness of makespans distributions computed by the incremental search tree exploration using Eq. 10.3 combined with the sum rule Eq. 10.4. For a single observation of a given plan, task and routing durations are generated randomly from their modeled distributions. The observed makespan is computed using the efficient Alg. 8, which is applicable for a full-length plan. By generating thousands of makespan observations, a probability distribution is derived by counting the number of observations that fall within different intervals. The interval length is 0.1 s and $5 \cdot 10^5$ simulations are run for each plan.

10.5.4 Evaluation results

At the top of Fig. 10.2, CDFs for the candidate plans of the stochastic approach are exemplified for use case A. The vertical axis indicates the k value of a percentile while the horizontal axis indicates the percentile, i.e., the maximum makespan for the best k share of outcomes. A risk-tolerant planner might prefer the red plan, which is the best plan with a 0-30% chance of making the corresponding percentile or better. On the other hand, the risk-averse planner might go for the black plan, which is the best plan with an 85-100%

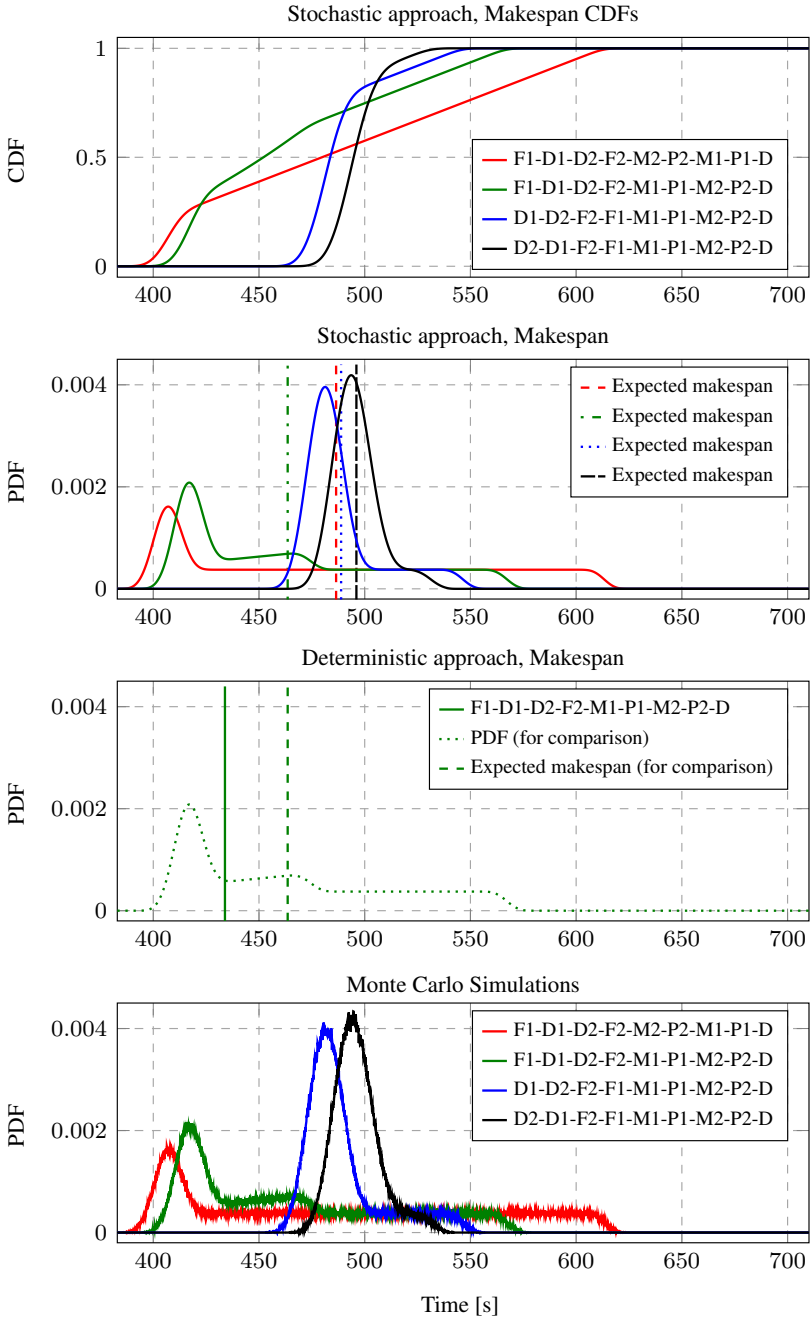

Figure 10.2: Makespans for use case (A) with two assembly tasks.

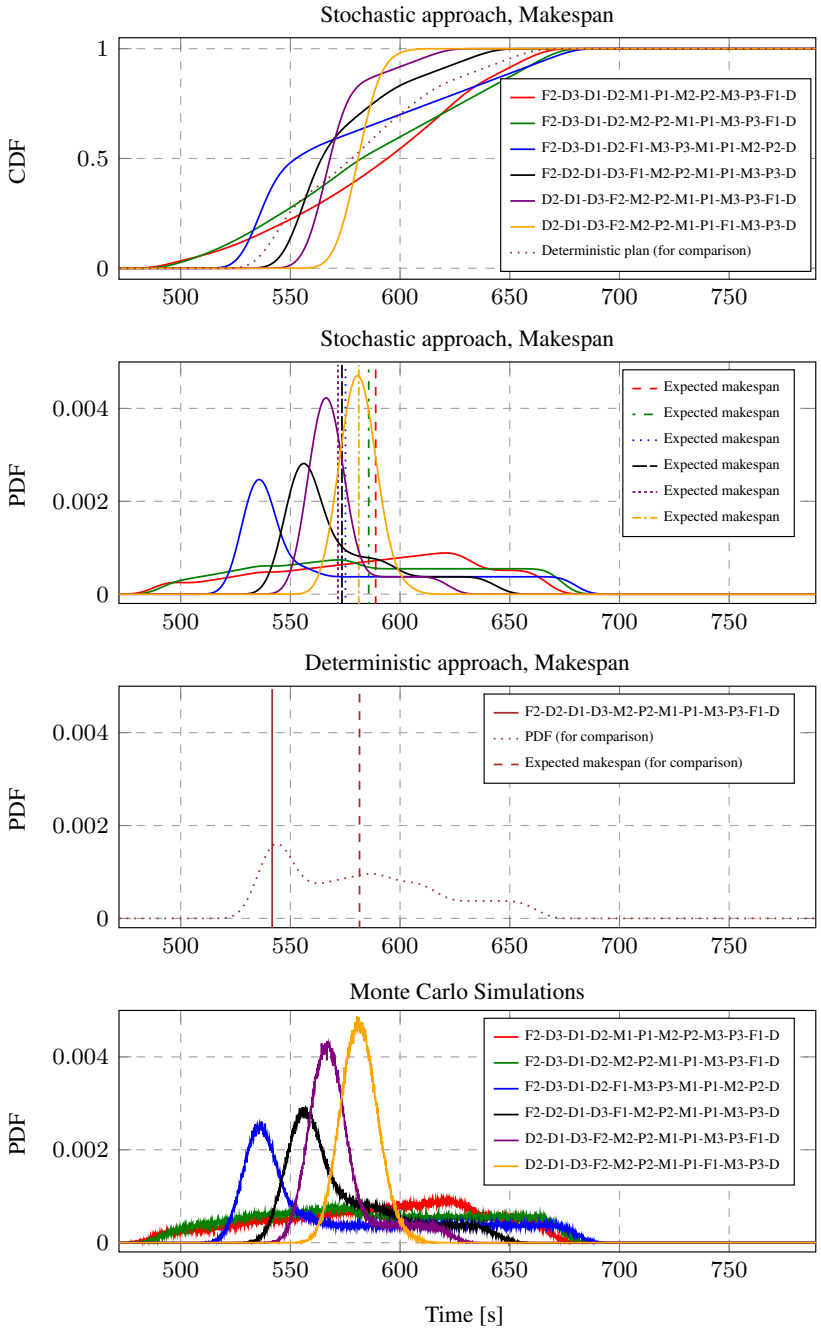

Figure 10.3: Makespans for the extended use case (B).

chance of making the corresponding percentile or better. The green plan is best with a 30-70% chance, while the blue is best between 70-85%. The second graph indicates corresponding PDFs, and their expected makespans as vertical lines, giving an intuition on possible variations. The third graph indicates the makespan generated by the deterministic approach as a solid vertical line. For comparison, the dotted lines show the PDF and the expected makespan (see Def. 2) of this plan for the stochastic approach. In this example, the deterministic plan is also among the stochastic candidate plans. The bottom graph provides Monte Carlo distributions of the plans, normalized by dividing interval counts with the total number of observations. Use case B is exemplified in the same way in Fig. 10.3. Here, the plan computed by the deterministic approach is not found among the candidate plans of the stochastic approach. This is because the deterministic plan stochastically dominates the set of all other feasible plans according to Def. 9, thereby excluding it from the candidate set of Eq. 10.7. In the top graph, the CDF of the deterministic plan is included as a dotted line. For every k-value, the percentile of the deterministic plan is higher than the percentile of at least one of the candidate plans of the stochastic approach. However, the deterministic plan does not dominate any candidate plan in terms of first-order stochastic dominance. In a statistical comparison of the planning approaches, 100 plans were computed for each use case with randomized task locations and intervals of input distributions. Table 10.1 presents the average number of explored and pruned nodes and indicates how many of the deterministic plans were found among the candidate plans of the stochastic approach. In Fig. 10.4, a histogram presents frequencies of makespans of the deterministic approach subtracted by expected makespans of the stochastic approach, expressed as several standard deviations, for use case A. A negative value means the deterministic makespan is shorter than the expected stochastic makespan.

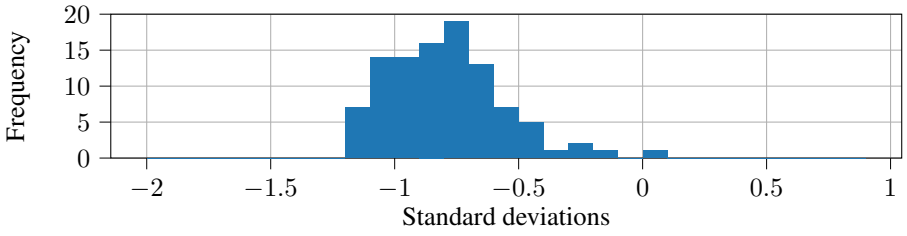

Figure 10.4: Frequencies of deterministic makespan subtracted with expected stochastic makespan, for use case A.

Planning approach	Number of explored nodes		Number of pruned nodes		Among the best stochastic plans	
	A	B	A	B	A	B
Stochastic B&B	773	16995	251	6262	(100%)	(100%)
Deterministic B&B	594	12116	259	5104	65%	47%

Table 10.1: Statistic comparison of the planning approaches.

10.5.5 Evaluation discussion

The Monte Carlo simulations in the bottom graphs of Figs. 10.2 and 10.3 are, if the noise is omitted, very similar to the PDFs in the corresponding figures, thereby supporting the correctness of the plan durations computed by the proposed planning approach. A large share of the PDFs are quite asymmetric, highlighting the advantage of not limiting the type of probabilistic distribution that can be represented. In this study, the deterministic approach tends to underestimate the makespan, as indicated in Fig. 10.4 and exemplified in the 3rd graph of Figs. 10.2, 10.3. This tendency magnifies the inherent problem of the deterministic approach, where a computed plan is associated with a more or less unknown risk. The stochastic approach considers this risk while searching for the best plans, providing information on how much a makespan can vary and suggesting the best plan with respect to the planner's willingness to risk. The deterministic plan is not always among the best plans of the stochastic approach, here in 65% and 47% of the runs (Table 10.1). This highlights the risk of computing an inferior plan by not considering uncertainties. The stochastic approach needs to explore 30% and 40% more nodes due to the safe but more restrictive pruning selection criterion. Still, the potential for pruning in the proposed approach is significant.

10.6 Conclusion

This paper proposes a novel methodology to compute an optimized collaborative plan while considering uncertain task durations and the risk willingness of a human planner. Relevant planning problems modeled with a Robot Task Scheduling Graph (RTSG) accommodate uncertainties by representing them as random variables. These are effectively tackled using a Branch-and-Bound (B&B) algorithm, incorporating a safe pruning strategy grounded in first-order stochastic dominance. The result is a set of the best plans for all risk levels, with makespans represented as probability distributions, empowering planners to make informed decisions based on their situational risk tolerance. Future

research includes the exploration of techniques for learning input distributions dynamically. Additionally, we foresee extending our methodology to consider other types of risks and handle more complex scenarios, such as centralized or decentralized multi-agent task allocation, agent load restrictions and intricate dependencies between tasks.

Bibliography

[1] Eloise Matheson, Riccardo Minto, Emanuele GG Zampieri, Maurizio Faccio, and Giulio Rosati. Human–robot collaboration in manufacturing applications: A review. *Robotics*, 8(4):100, 2019.

[2] Afshin Ameri E., Branko Miloradovic, Baran Çürüklü, Alessandro Vittorio Papadopoulos, Mikael Ekström, and Johann Dréo. Interplay of Human and AI Solvers on a Planning Problem. In *IEEE Int. Conf. Sys., Man, & Cyb. (SMC)*, 2023.

[3] Manman Yang, Erfu Yang, Remi Christophe Zante, Mark Post, and Xuefeng Liu. Collaborative mobile industrial manipulator: a review of system architecture and applications. In *Int. Conf. Autom. & Comp. (ICAC)*, pages 1–6, 2019.

[4] Anders Lager, Alessandro V. Papadopoulos, Giacomo Spampinato, and Thomas Nolte. A task modelling formalism for industrial mobile robot applications. In *Int. Conf. Advanced Robotics (ICAR)*, pages 296–303, 2021.

[5] George Ferguson, James F Allen, Bradford W Miller, et al. TRAINS-95: Towards a mixed-initiative planning assistant. In *Conf. Artificial Intelligence Planning Systems (AIPS)*, pages 70–77, 1996.

[6] Anders Lager, Giacomo Spampinato, Alessandro V Papadopoulos, and Thomas Nolte. Task roadmaps: speeding up task replanning. *Frontiers in Robotics & AI*, 9:816355, 2022.

[7] Reuven Y Rubinstein and Dirk P Kroese. *Simulation and the Monte Carlo method*. John Wiley & Sons, 2016.

[8] Zhihao Liu, Quan Liu, Lihui Wang, Wenjun Xu, and Zude Zhou. Task-level decision-making for dynamic and stochastic human-robot collaboration based on dual agents deep reinforcement learning. *Int. Journal of Advanced Manufacturing Tech.*, 115(11-12):3533–3552, 2021.

[9] Lars Johannsmeier and Sami Haddadin. A hierarchical human-robot interaction-planning framework for task allocation in collaborative industrial assembly processes. *IEEE RA-L*, 2(1):41–48, 2016.

[10] Martina Lippi and Alessandro Marino. A mixed-integer linear programming formulation for human multi-robot task allocation. In *IEEE Int. Symp. RO-MAN*, pages 1017–1023, 2021.

[11] Michael Saint-Guillain, Tiago Vaquero, Steve Chien, Jagriti Agrawal, and Jordan Abrahams. Probabilistic temporal networks with ordinary distributions: Theory, robustness and expected utility. *Journal of Artificial Intelligence Research*, 71:1091–1136, 2021.

[12] Jun Kinugawa, Akira Kanazawa, Shogo Arai, and Kazuhiro Kosuge. Adaptive task scheduling for an assembly task coworker robot based on incremental learning of human's motion patterns. *IEEE RA-L*, 2(2):856–863, 2017.

[13] Andrea Maria Zanchettin, Andrea Casalino, Luigi Piroddi, and Paolo Rocco. Prediction of human activity patterns for human–robot collaborative assembly tasks. *IEEE Trans. Industrial Informatics*, 15(7):3934–3942, 2018.

[14] Andrea Casalino and A Geraci. Allowing a real collaboration between humans and robots. *Special Topics in Information Technology*, 2021.

[15] Andrea Casalino, Eleonora Mazzocca, Maria Grazia Di Giorgio, Andrea Maria Zanchettin, and Paolo Rocco. Task scheduling for human-robot collaboration with uncertain duration of tasks: a fuzzy approach. In *Int. Conf. Control, Mechatronics and Automation (ICCMA)*, pages 90–97, 2019.

[16] Andrew W Palmer, Andrew J Hill, and Steven J Scheding. Modelling resource contention in multi-robot task allocation problems with uncertain timing. In *IEEE Int. Conf. ICRA*, pages 3693–3700, 2018.

[17] Andrew W Palmer, Andrew J Hill, and Steven J Scheding. Methods for stochastic collection and replenishment (scar) optimisation for persistent autonomy. *Robotics and Autonomous Systems*, 87:51–65, 2017.

[18] Ping Lou, Quan Liu, Zude Zhou, Huaiqing Wang, and Sherry Xiaoyun Sun. Multi-agent-based proactive–reactive scheduling for a job shop. *Int. Journal of Advanced Manufacturing Tech.*, 59:311–324, 2012.

[19] Rainer Müller, Matthias Vette, and Ortwin Mailahn. Process-oriented task assignment for assembly processes with human-robot interaction. *Procedia CIRP*, 44:210–215, 2016.

[20] Kourosh Darvish, Barbara Bruno, Enrico Simetti, Fulvio Mastrogiovanni, and Giuseppe Casalino. Interleaved online task planning, simulation, task allocation and motion control for flexible human-robot coop-

eration. In *IEEE Int. Symp. Robot and Human Interactive Comm. (RO-MAN)*, pages 58–65, 2018.

[21] Moshe Shaked and J. George Shanthikumar, editors. *Univariate Stochastic Orders*, pages 3–79. Springer New York, New York, NY, 2007.

[22] Elmar Wolfstetter, Uwe Dulleck, Roman Inderst, Peter Kuhbier, and M Lands-Berger. *Stochastic dominance: theory and applications*. Humboldt-Univ., Wirtschaftswiss. Fak., 1993.

Chapter 11

Paper E
Stochastic Scheduling for Human-Robot Collaboration in Dynamic Manufacturing Environments

Anders Lager, Branko Miloradović, Giacomo Spampinato, Thomas Nolte and Alessandro V. Papadopoulos. Accepted by 34th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN), 2025.

Abstract

Collaborative human-robot teams enhance efficiency and adaptability in manufacturing, but task scheduling in mixed-agent systems remains challenging due to the uncertainty of task execution times and the need for synchronization of agent actions. Existing task allocation models often rely on deterministic assumptions, limiting their effectiveness in dynamic environments. We propose a stochastic scheduling framework that models uncertainty through probabilistic makespan estimates, using convolutions and stochastic max operators for realistic performance evaluation. Our approach employs meta-heuristic optimization to generate executable schedules aligned with human preferences and system constraints. It features a novel deadlock detection and repair mechanism to manage cross-schedule dependencies and prevent execution failures. This framework offers a robust, scalable solution for real-world human-robot scheduling in uncertain, interdependent task environments.

11.1 Introduction

Collaborative robotics in manufacturing promises enhanced efficiency and adaptability by combining robotic precision with human dexterity [1]. While autonomous systems are effective, their high cost and inflexibility in dynamic settings have led to the rise of human-robot collaboration. This paradigm introduces significant scheduling challenges due to uncertainties in task durations and human preferences. In modern production environments, scheduling is further complicated by cross-schedule dependencies among multiple robots and human operators. For example, a human may need to complete an inspection before a robot proceeds with packaging, or several agents might coordinate transport tasks to avoid bottlenecks, or multi-agent tasks cannot start until all assigned agents have arrived. Mismanagement of these dependencies can lead to deadlocks and inefficient workflows. Additionally, uncertainties arise from robotic delays, environmental obstacles, and human factors such as fatigue, emphasizing the need for robust scheduling frameworks that integrate ergonomic constraints and individual preferences.

This work tackles the Multi-Robot Task Allocation (MRTA) [2] problem in a stochastic, human-in-the-loop context. Differently from traditional deterministic approaches, our method models task and routing durations as random variables with probability distributions. By analytically deriving the makespan distribution using convolutions and stochastic max operators, we obtain bounds that better reflect real-world variability. By incorporating human preferences, our framework not only enhances worker engagement but also achieves a more balanced workload distribution.

Prior work has explored similar challenges. Palmer et al. [3] computed stochastic objective functions under Gaussian assumptions, and our earlier work [4] extended planning to single-robot systems with non-restrictive distributions. Other studies, such as those using Probabilistic Simple Temporal Networks [5] or adaptive scheduling based on real-time human performance [6], address temporal constraints and reactive adaptation, yet they do not fully capture the stochastic and interdependent nature of collaborative manufacturing tasks. Markov Decision Processes models uncertainties in the effects of task executions [7], while we model uncertainties of task and routing durations, which may encompass some relevant aspects of such uncertainties, e.g., retries after failure. While some research has focused on individual agent performance [8, 9] or ergonomic risks [10], these efforts often overlook global task dependencies and the cross-schedule precedence constraints–namely, start-after-completion, start-after-start, complete-after-completion, and complete-after-start–as defined in [11].

Our primary contribution is a novel stochastic scheduling framework for MRTA that explicitly accounts for execution uncertainties and human-centric considerations. A novel deadlock detection and repair mechanism is introduced to manage the inherent cross-schedule dependencies. Finally, a novel heuristic is presented for guiding the search of a Greedy Randomized Adaptive Search Procedure (GRASP) [12] able to find optimized solutions to the scheduling problem in a limited time frame. We validate the correctness and accuracy of the proposed framework using Monte Carlo (MC) simulations, and a deterministic variant of the framework is compared. Further, the GRASP algorithm and a Genetic Algorithm (GA) [13] are evaluated across test instances of varying complexity. A hybrid approach, where GRASP is used to provide initial solutions for GA, is also investigated.

11.2 Motivating Example

In discrete manufacturing processes [14], activities such as assembly, inspection, and transport are orchestrated to process parts efficiently. Each part or batch is handled through a sequence of tasks–each assigned to an individual or a team of agents–that collectively form the basis for a multi-agent scheduling problem.

Manufacturing Process Model

A typical process model outlines the series of activities needed to transform an order into a finished product. For example, an order may progress through:

- *Fetch*: Retrieving parts from storage locations and deliver them to a workstation.
- *Assemble*: Combining parts to form a product.
- *Inspect*: Evaluating product quality.
- *Transport*: Moving products between workstations.
- *Palletize*: Grouping finished products for shipment.

Each activity is characterized by an input queue (holding parts to be processed) and an output queue (storing processed parts), where the output of one activity becomes the input for the next. A *task* represents the execution of one such activity, including subtasks such as loading, processing, and unloading. The

number of parts handled in a task depends on the order specifications and activity constraints. Importantly, aside from the initial order queue, all queues follow a First-In, First-Out (FIFO) rule, and certain activities (e.g., *Assemble*) may require strictly sequential processing due to limited space or resources.

Task Network: A Motivating Example

From the process model, a *task network* is constructed by simulating the flow of parts through the queues and activities. This network is modeled as a directed acyclic graph (DAG) where each node represents a task, and edges denote precedence constraints (PCs). Figure 11.1 illustrates a typical task network for sequential activities. Key features of this network include:

- The initial task τ_F, which marks the start of the process.
- A primary set of PCs enforcing FIFO access within the same activity—ensuring that, in sequential operations, a task cannot start until its predecessor completes (*start-after-completion*), while in concurrent setups, it may start once the preceding task has begun (*start-after-start*) and complete once the preceding task has finished (*complete after completion*). For the activity connected to the order queue, the start of tasks are not restricted.
- A secondary set of PCs governing transitions between consecutive ac-

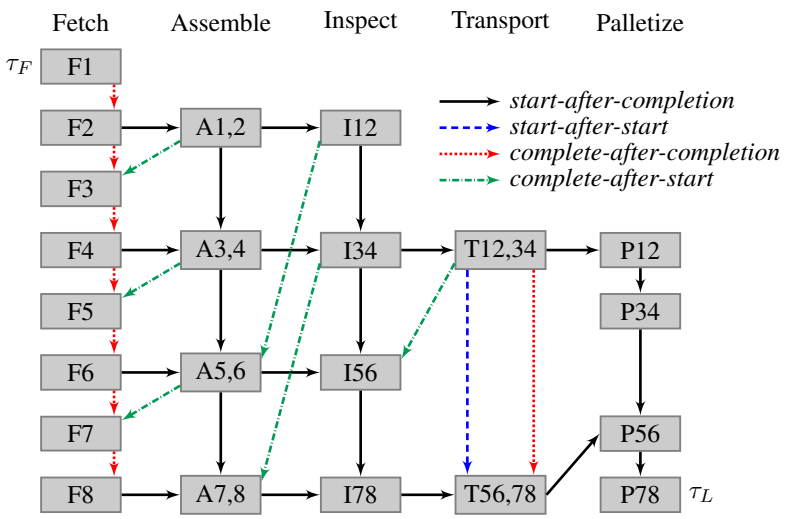

Figure 11.1: Task network for a manufacturing scenario.

tivities; for instance, a task in the next activity cannot start until the previous activity has fully populated the common queue (*start-after-completion*).

- A tertiary set of PCs arising from queue capacity limits, where a task's completion may depend on subsequent activities removing parts to create the necessary space (*complete-after-start*).

The process is considered complete when the final task τ_L of the last activity is finished. This motivating example highlights the complexity inherent in task scheduling for manufacturing systems, where managing sequential and interdependent tasks is critical to avoiding bottlenecks and ensuring smooth operations.

11.3 Defining the Scheduling Problem

This paper formulates the scheduling problem as an optimization model that assigns tasks to agents in a feasible and efficient manner. The model must respect PCs, handle uncertainty in task and routing durations, avoid deadlocks, and incorporate human preferences. The problem falls within the MRTA taxonomy [2, 15, 16] as an XD [ST-MR-TA] (Single-Task robots (ST), Multi-Robot tasks (MR), Time-extended Assignment (TA) with cross-schedule dependencies (XD) as defined in [17]). In the following, we introduce the general assumptions and notation, describe the stochastic modeling of durations and the computation of plan makespans, and finally define the objective function.

General Assumptions and Notation Let T be the set of all tasks to be completed, and A be the set of all available agents (robots and humans). Each task $\tau \in T$ can be executed by a team of agents. For every task τ, a decision variable

$$f_\tau \in \{1, \ldots, m_\tau\}$$

selects one of the m_τ predefined *team formations*. These represent alternative ways to perform τ, specifying the number and types of agents required and their given *roles* in carrying out the task. For example, $f_\tau = 1$ requires a robot and a human performer, while $f_\tau = 2$ requires a robot and a human supervisor. The set of agents assigned to τ is denoted by

$$F_\tau \subseteq A,$$

which must satisfy the requirements of the chosen formation. Agents execute tasks sequentially. For each agent $a \in A$, let

$$\tau_i^a \in T, \quad i = 0, \ldots, n_a,$$

represent the i-th task in agent a's sequence (with τ_0^a denoting the starting location or initial state). We define the set of immediate predecessor tasks for task τ as

$$\mathrm{P}_\tau^{\mathrm{apt}} = \{\tau_{i-1}^a \mid \tau_i^a = \tau,\ a \in F_\tau\}.$$

PCs are imposed by the edges of the process model, as illustrated in Figure 11.1, categorized as follows:

- $\mathrm{P}_\tau^{\mathrm{cps}}$: Tasks whose completions precede the start of τ.
- $\mathrm{P}_\tau^{\mathrm{sps}}$: Tasks whose starts precede the start of τ.
- $\mathrm{P}_\tau^{\mathrm{cpc}}$: Tasks whose completions precede the completion of τ.
- $\mathrm{P}_\tau^{\mathrm{spc}}$: Tasks whose starts precede the completion of τ.

In addition, the overall plan must be free of deadlocks, which can occur if the agents' scheduled routes, in combination with PCs, create cyclic dependencies between tasks.

11.3.1 Stochastic Modeling of Durations

Task execution and agent routing times are modeled as independent, nonnegative random variables with generic probabilistic distributions. We define the routing duration for agent $a \in A$ moving from task t to task τ as $R_{a,t,\tau}$. The duration required to execute task τ under team formation f_τ with agents F_τ is defined as A_{τ,f_τ,F_τ}. Initial distributions may be set as uniform (based on expert estimates of minimum and maximum values) and later refined with empirical data from observations in simulation or reality. In the following, we provide some stochastic definitions needed by the proposed scheduling framework.

Definition 1 (Random Variable)**.** *A random variable X on the probability space $(\Omega, \mathcal{F}, \mathbb{P})$ is a measurable function $X : \Omega \to \mathbb{R}$ such that $\{\omega \in \Omega : X(\omega) = x\} \in \mathcal{F}$ for all $x \in \mathbb{R}$.*

Definition 2 (Probability density function (PDF))**.** *The PDF $f_X(x)$ of a random variable X is defined as:*

$$f_X(x) = \mathbb{P}[\omega \in \Omega \mid X(\omega) = x]$$

Definition 3 (Cumulative distribution function (CDF))**.** *The CDF $F_X(x)$ of a random variable X is defined as:*

$$F_X(x) = \mathbb{P}[\omega \in \Omega \mid X(\omega) \leq x]$$

Definition 4 (Percentile)**.** *The k-th percentile of a probabilistic distribution $f_X(x)$ is defined as:*

$$p_k = \inf\{x : F_X(x) \geq k\}, \quad 0 < k < 1.$$

Definition 5 (First-Order Stochastic Dominance [18])**.** *Consider two random variables, X and Y, with CDFs F_X and F_Y. X has a first-order stochastic dominance over Y, if and only if $\forall x$, $F_X(x) \leq F_Y(x)$, and $\exists x$, $F_X(x) < F_Y(x)$. The stochastic dominance is denoted as $X \geq_{st} Y$.*

Definition 6 (Independence)**.** *Two random variables X and Y are independent if the pair of events $\{X = x\}$ and $\{Y = y\}$ are independent for all $x, y \in \mathbb{R}$. Formally,*

$$\mathbb{P}[X = x, Y = y] = \mathbb{P}[X = x]\mathbb{P}[Y = y], \quad \forall\, x, y \in \mathbb{R}.$$

Definition 7 (Convolution or sum of random variables)**.** *If X and Y are independent random variables on $(\Omega, \mathcal{F}, \mathbb{P})$, then $Z = X + Y$ has probability density function, when X and Y are discrete random variables*

$$\mathbb{P}[Z = z] = \sum_{x=-\infty}^{\infty} f_X(x) f_Y(z - x), \quad \forall\, z \in \mathbb{Z}, \tag{11.1}$$

and for continuous random variables:

$$\mathbb{P}[Z = z] = \int_{-\infty}^{\infty} f_X(x) f_Y(z - x)\, \mathrm{d}x. \tag{11.2}$$

Definition 8 (Maximum between random variables [4])**.** *If X and Y are independent random variables on $(\Omega, \mathcal{F}, \mathbb{P})$, then $Z = \max(X, Y)$ has cumulative probability function*

$$F_Z(z) = F_X(z) F_Y(z)$$

11.3.2 Plan Duration Computation

The completion time of a task has a probability distribution which is determined by the durations along the *critical paths*–ordered sequences of routing and task durations from the start of the plan to that task. We denote this completion time as $K(\tau)$ and define:

$$K(\tau) = \max\big(\mathcal{C}(\tau)\big), \tag{11.3}$$

where $\mathcal{C}(\tau)$ is the set of computed completion durations along all critical paths for task τ. To compute these durations, we recursively define the start and completion duration operators. The start duration operator $\mathcal{S}(\tau)$ is given by:

$$\begin{aligned}\mathcal{S}(\tau) = \mathcal{M}\Bigg(\mathcal{F}\Bigg(&\bigcup_{t\in \mathrm{P}_\tau^{\mathrm{apt}}} \Big(\mathcal{C}(t) + \max\Big(\bigcup_{a\in F_\tau \cap F_t} R_{a,t,\tau}\Big)\Big) \\ &\cup \bigcup_{t\in\mathcal{K}(\mathrm{P}_\tau^{\mathrm{cps}})} \mathcal{C}(t) \cup \bigcup_{t\in\mathcal{K}(\mathrm{P}_\tau^{\mathrm{sps}})} \mathcal{S}(t)\Bigg)\Bigg),\end{aligned} \tag{11.4}$$

and the completion duration operator $\mathcal{C}(\tau)$ is defined as:

$$\begin{aligned}\mathcal{C}(\tau) = \mathcal{M}\Bigg(\mathcal{F}\Bigg(&\mathcal{S}(\tau) + A_{\tau,f_\tau,F_\tau} \\ &\cup \bigcup_{t\in\mathcal{K}(\mathrm{P}_\tau^{\mathrm{cpc}})} \mathcal{C}(t) \cup \bigcup_{t\in\mathcal{K}(\mathrm{P}_\tau^{\mathrm{spc}})} \mathcal{S}(t)\Bigg)\Bigg),\end{aligned} \tag{11.5}$$

with initial durations, $\{\mathcal{S}(\tau), \mathcal{C}(\tau) | \tau = \tau_0^a\} = 0 \ \forall a \in A$, and where a sum of a set and a single element is performed element-wise. $\mathcal{M}(\cdot)$ is an operator that merges durations from a set into a smaller representative set. $\mathcal{F}(\cdot)$ is an operator that prunes inferior durations from a set, where X is inferior

Table 11.1: Examples of the merge operator, $\mathcal{M}(\cdot)$

#	X	Y	$\mathcal{M}(\{X, Y\})$
1	$A + B$	$A + C$	$\{A + \max(B, C)\}$
2	A	$A + B$	$\{Y\}$
3	$B + D$	$\max(B, C)$	$\{\max(B + D, C)\}$
4	B	$\max(B + C, D) + E$	$\{Y\}$
5	$B + D$	$\max(B, C) + E$	$\{X,\ Y\}$

if $\exists Y | \max(X, Y) = Y$. The operator $\mathcal{K}(\cdot)$ selects the subset of preceding tasks that are critical due to causality, e.g., a preceding task is not critical if assigned to the same agent. The purpose of these operators is to enhance the computational efficiency and accuracy by reducing $\mathcal{S}(\tau)$ and $\mathcal{C}(\tau)$. Further efficiency is gained by avoiding the recomputation of $\mathcal{S}(\tau)$ and $\mathcal{C}(\tau)$ with the same τ. Table 11.1 provides examples illustrating the action of the merge operator, $\mathcal{M}(\cdot)$, where X and Y are composed of independent durations. A merge of them occurs if their combined duration, $\max(X, Y)$, can be expressed with independent operands only.

11.3.3 Plan Duration Bounds

Exact analytical computation of $K(\tau)$ is challenging due to the possibility of dependencies among completion durations, $\mathcal{C}(\tau)$. Instead, the estimate $K_e(\tau)$ is computed as

$$K_e(\tau) = \max\big(\mathcal{C}_{\text{ind}}(\tau)\big), \tag{11.6}$$

where $\mathcal{C}_{\text{ind}}(\tau)$ is the same set as $\mathcal{C}(\tau)$ under the assumption of independence. We define a lower bound $K_l(\tau)$ with CDF

$$F_{K_l(\tau)} = \min_{x \in \mathcal{C}_{\text{ind}}(\tau)} F_x, \tag{11.7}$$

where F_x is the CDF of x. Note that the computed completion durations of all critical paths $\mathcal{C}(\tau)$ may generally have shared tasks that will make the duration of the individual path dependent on the others. We show that considering the duration of the different critical paths as independent random variables, $\mathcal{C}_{\text{ind}}(\tau)$, provides *safe bounds*, in terms of stochastic dominance, for the case of dependent variables.

Theorem 1. *Assume that $K(\tau)$ from Eq. 11.3 represents the exact task duration and $K_e(\tau)$ from Eq. 11.6 is its estimate. Further, let $K_l(\tau)$ be defined as above. Then,*

$$K_l(\tau) \leq_{st} K(\tau) \leq_{st} K_e(\tau),$$

where $\leq_{st}$ denotes the stochastic order.

Proof. Since $\mathcal{C}(\tau)$ is composed of nonnegative, continuous random variables combined via $+$ and $\max$ operators, the resulting durations are exact and positively dependent (i.e., they are non-decreasing functions of any shared random variable). Eq. (11.7) and Theorem 2 (in the Appendix) imply that if $\mathcal{C}(\tau)_{com}$ denotes a comonotonic version of $\mathcal{C}(\tau)$, then

$$\max(\mathcal{C}(\tau)_{com}) \leq_{st} \max(\mathcal{C}(\tau)) \leq_{st} \max(\mathcal{C}_{\text{ind}}(\tau)).$$

Thus, $K_l(\tau) \leq_{st} K(\tau) \leq_{st} K_e(\tau)$. □

11.3.4 Human Preferences and Ergonomic Constraints

For a human agent a, task allocation considers ergonomic and motivational factors. An idle time quota, $Q_a^I \in [0,1]$, can be specified, defining a guaranteed minimum level of free time that can be spent on breaks or other (non-planned) tasks. Preferred activities and roles are specified with indicator $\mathcal{L}_{a,\tau,f_\tau}$ to be 1 if task τ with team formation f_τ is preferred, and 0 otherwise. An activity time quota, $Q_a^P \in [0,1]$, defines the desired proportion of the active time to be spent on preferred activities and roles. The total activity time for an agent, including planned tasks and routing, is given by

$$T_a = \sum_{i=1}^{n_a} \Big(R_{a,\tau_{i-1},\tau_i} + A_{\tau_i,f_{\tau_i},F_{\tau_i}} \Big),$$

and the total preferred activity time is

$$P_a = \sum_{i=1}^{n_a} \mathcal{L}_{a,\tau_i,f_{\tau_i}} \Big(R_{a,\tau_{i-1},\tau_i} + A_{\tau_i,f_{\tau_i},F_{\tau_i}} \Big).$$

We require that the estimated median idle time quota for human agent a, denoted by

$$Q_a^{I*} = \frac{p_{50}(K_l(\tau_L)) - p_{50}(T_a)}{p_{50}(K_e(\tau_L))}, \tag{11.8}$$

satisfies

$$Q_a^{I*} \geq Q_a^I, \quad \forall\, a \in H, \tag{11.9}$$

where $p_{50}(\cdot)$ returns the median and $H \subseteq A$ is the set of human agents. Similarly, the median preferred activity time quota

$$Q_a^{P*} = \frac{p_{50}(P_a)}{p_{50}(T_a)}$$

should satisfy

$$Q_a^{P*} \geq Q_a^P, \quad \forall\, a \in H, \tag{11.10}$$

Unlike constraint Eq. (11.9), this is considered a soft constraint.

11.3.5 Objective Function

The primary objective is to minimize the makespan, represented by $K_e(\tau_L)$—the upper bound completion time of the final task τ_L. Given a specified risk level $k \in [0,100]$, we aim to minimize the k-th percentile of the makespan, $p_k(K_e(\tau_L))$. A low k promotes plans with better chances to reach

lower makespans (higher risk) while a high k promotes plans with better chances to avoid higher makespans (lower risk). To account for human-centric constraints, penalty terms are added when the estimated quotas in Eqs. (11.9) and (11.10) are not met. The objective function to be minimized is:

$$J = p_k(K_e(\tau_L)) \; + \sum_{a \in \{H | n_a > 0\}} \Big[\left[Q_a^P - Q_a^{P*}\right]^+ \; p_{50}(K_e(\tau_L)) \\ + \; \left[Q_a^I - Q_a^{I*}\right]^+ \; p_{50}(T_a) \Big], \tag{11.11}$$

where $[x]^+ := \max(x, 0)$.

11.4 Solving the Scheduling Problem

Due to the stochastic representation of the makespan, developing an exact solution algorithm for the scheduling problem is highly challenging. Consequently, we adopt variants of two metaheuristic approaches to search for sub-optimal solutions, namely a GRASP algorithm with a novel efficient search heuristics and a GA approach inspired by [19]. These approaches are designed to explore the solution space while handling uncertainty, PCs, deadlocks and human preferences.

11.4.1 GRASP Algorithm

The GRASP algorithm builds a solution incrementally, starting from an empty solution and in each step selecting and scheduling a *combination* of one task, one related team formation, and a matching agent set. A task is selectable if unscheduled and all of the task's predecessors are already scheduled in the partially built solution. Selection is guided by a heuristic function (see Eq. (11.12)) that for any selectable combination estimates the impact of scheduling the related task at the end of the sequences of the related agents in the partial solution. From the best-ranked alternatives, a randomized selection is made, with higher-ranked combinations having higher selection probabilities. The selected task is denoted by τ_i for step $i \in \{1, 2, \ldots, |T|\}$. After the last step $|T|$, the resulting solution is repaired for deadlocks (see Section 11.4.3) and compared against the current best solution using the objective function defined in Eq. (11.11).

GRASP Heuristic Function In step i, the heuristic function $\mathcal{H}(\tau_i, f_{\tau_i}, F_{\tau_i})$ estimates the incremental cost of scheduling a selectable combination for τ_i,

f_{τ_i} and F_{τ_i}. It sums estimates of makespan increase and deviations from preferred idle and activity times using (non-stochastic) median values of task and routing durations, defined as:

$$\mathcal{H}(\tau_i, f_{\tau_i}, F_{\tau_i}) = \left[C(\tau_i) - \max_{\{\tau_j | j<i\}} C(\tau_j)\right]^+ + \sum_{a \in F_{\tau_i} \cap H} \left[\left[T_a - C(\tau_i)(1 - Q_a^I)\right]^+ + \left[Q_a^P T_a - P_a\right]^+\right], \tag{11.12}$$

where

$$C(\tau) = \max\Big\{S(\tau) + p_{50}(A_{\tau, f_\tau, F_\tau}),\ \max_{t \in \mathrm{P}_\tau^{\mathrm{cpc}}} C(t),\ \max_{t \in \mathrm{P}_\tau^{\mathrm{spc}}} S(t)\Big\},$$

$$S(\tau) = \max\left\{\max_{t \in \mathrm{P}_\tau^{\mathrm{apt}}} \Big(C(t) + \max_{a \in F_\tau \cap F_t} p_{50}(R_{a,t,\tau})\Big),\ \max_{t \in \mathrm{P}_\tau^{\mathrm{cps}}} C(t),\ \max_{t \in \mathrm{P}_\tau^{\mathrm{sps}}} S(t)\right\}$$

and $S(\tau_0^a) = C(\tau_0^a) = 0\ \forall a \in A$.

11.4.2 Genetic Algorithm (GA)

In the GA framework, an *individual* represents a feasible solution and is encoded by a chromosome that includes grounded decision variables, such as the sequence of tasks for each agent (with multi-robot tasks appearing in multiple agent sequences) and the selected team formation for each task. The algorithm proceeds as follows:

1. **Initialization:** An initial *generation*, i.e., a first population of individuals, is generated by assigning tasks in a random order to randomly selected team formations with corresponding agent sets. These assignments are then adjusted to ensure that all intra-schedule PCs are satisfied. Deadlocks are resolved as described in Section 11.4.3.

2. **Creating Next Generation:** A set of mutation operators is applied to randomly selected individuals from the current generation. Although these mutations preserve intra-schedule PCs, they may temporarily introduce deadlocks, which are subsequently repaired.

3. **Evaluation and Sorting:** The new generation is evaluated using the objective function (see Eq. 11.11) and sorted accordingly. An elitism strategy retains a small fraction of the best-performing individuals from the previous generation, replacing the least fit individuals.

Steps 2-3 are repeated until convergence is achieved. To enhance diversity, four mutation operators are used. *Task-to-Idle* moves a task from one agent to another with most surplus idle time, inserting it at a valid position. *Team Formation* changes a task's team formation—retaining current agents if possible, otherwise reassigning them. *Task Insert* transfers a task to a different agent compatible with the team formation. *Task Swap* exchanges tasks between two agents while potentially changing their team formations.

11.4.3 Deadlock Search and Repair

Deadlocks occur when agent schedules introduce cyclic dependencies that violate the acyclic structure of the task network. To eliminate these deadlocks (see Algorithm 7 for detailed procedure), we employ a two-stage procedure.

Dependency Order (DO) Search: Starting from the initial task τ_F, the algorithm recursively tries to determine a valid dependency order for all tasks, from τ_F to τ_L, taking into account both PCs and agent schedules. This process identifies *lock-sets* L_t, which consist of tasks that cannot be scheduled until task t's dependency order is decided.

Detect and Repair Cycles: Starts with a DO search. If circular dependencies are detected (i.e., some lock-sets remain unresolved), a repair step is invoked to identify a pair of tasks t and $x \in L_t$ that are assigned to the same agent and lack a definitive ordering. Their order is swapped in the agent's schedule and if x is a multi-agent task, the order of x with a corresponding task t' in another agent's schedule may also be swapped to avoid creating a new cycle. The DO search is reinitiated and the process is rerun iteratively, until no lock-sets remain and a valid dependency order is found.

Algorithm 7 Detect and repair deadlocks

function DETECTANDREPAIRCYCLES
  $DOfound \leftarrow$ **false**
  while $\neg DOfound$ **do**
    for $t \in T$ **do** $L_t \leftarrow \emptyset$ ▷ Reset lock sets
    end for
    for $t \in T$ **do** $D_t \leftarrow$ **false** ▷ Reset DO flags
    end for
    $d \leftarrow 0$ ▷ Reset number of ordered tasks
    SEARCHDO(τ_F) ▷ Search from the first task
    if $\exists a, x, t | x = \tau_i^a \wedge t = \tau_j^a \wedge i < j \wedge x \in L_t \wedge x \not\asymp t$ **then**
      if $\exists$ alternative such a, x, t combinations: a', x', t' **then**
        Select $a, x, t | x \not\asymp x'$ for all combinations
      end if
      SWAPTASKORDER(a,x,t) ▷ Remove cycle with agent a
      for $\{a', x', t' | x' = x \wedge a' \neq a \wedge (t \prec t' \vee t = t')\}$ **do**
        SWAPTASKORDER(a',x,t') ▷ Avoid new cycle with a'
      end for
    else $DOfound \leftarrow$ **true** ▷ Done. No cycles in schedule
    end if
  end while
end function
function SEARCHDO(τ)
  $D_\tau \leftarrow$ **true** ▷ DO for τ is decided
  $d \leftarrow d + 1$
  $\mathcal{D}_d = \tau$ ▷ Store the dependency order of tasks
  if $\exists \tau_o | \tau \in \mathrm{P}_{\tau_o}^{\mathrm{cps}} \wedge \neg$SAMEACTIVITY($\tau_o$,$\tau$) **then**
    $\chi = \{t | t \in \mathrm{P}_{\tau_o}^{\mathrm{apt}} \cup \mathrm{P}_{\tau_o}^{\mathrm{cps}} \cup \mathrm{P}_{\tau_o}^{\mathrm{sps}} \cup \mathrm{P}_{\tau_o}^{\mathrm{cpc}} \cup \mathrm{P}_{\tau_o}^{\mathrm{cps}} \wedge \neg D_t\}$
    for $t \in \chi$ **do** $L_t \leftarrow L_t \cup \tau_o$ ▷ τ_o is locked by t
    end for
    if $\chi = \emptyset$ **then** SEARCHDO(τ_o) ▷ Search the unlocked τ_o
    end if
  end if
  if $\exists \tau_s | \tau \in \mathrm{P}_{\tau_s}^{\mathrm{cps}} \cup \mathrm{P}_{\tau_s}^{\mathrm{sps}} \cup \mathrm{P}_{\tau_s}^{\mathrm{cpc}} \wedge$SAMEACTIVITY($\tau_s$,$\tau$) **then**
    $\chi = \{t | t \in \mathrm{P}_{\tau_s}^{\mathrm{apt}} \cup \mathrm{P}_{\tau_s}^{\mathrm{cps}} \cup \mathrm{P}_{\tau_s}^{\mathrm{sps}} \cup \mathrm{P}_{\tau_s}^{\mathrm{cpc}} \cup \mathrm{P}_{\tau_s}^{\mathrm{cps}} \wedge \neg D_t\}$
    for $t \in \chi$ **do** $L_t \leftarrow L_t \cup \tau_s$ ▷ τ_s is locked by t
    end for
    if $\chi = \emptyset$ **then** SEARCHDO(τ_s) ▷ Search the unlocked τ_s
    end if
  end if
  for $t \in L_\tau$ **do**
    $L_\tau \leftarrow L_\tau \setminus t$ ▷ τ no longer locks t
    if $\nexists x | t \in L_x$ **then** SEARCHDO(t) ▷ Search the unlocked t
    end if
  end for
end function

Table 11.2: Use case settings.

Use case	#T	#R	#H	Use case	#T	#R	#H
1A	11	2	1	1B	11	3	2
2A	22	2	1	2B	22	3	2
3A	33	2	1	3B	33	3	2
4A	44	2	1	4B	44	3	2
5A	55	2	1	5B	55	3	2

*(#T - no of tasks; #R - no of robots; #H - no of humans)

11.5 Evaluation

This section evaluates the proposed stochastic computation approach using Monte Carlo simulations and a deterministic counterpart. It also compares GRASP with two variants of the GA approach, for solving the scheduling problem. The Python code used to obtain the results is available online[1].

[1] https://github.com/anders-lager/stochastic-scheduling

Algorithm 8 Deterministic makespan computation

function COMPUTEMAKESPAN
  $S_0 \leftarrow 0$; $C_0 \leftarrow 0$; $i \leftarrow 0$
  while $i \leq |T|$ **do**
    $i \leftarrow i + 1$
    $\tau \leftarrow \mathcal{D}_i$ ▷ Visit tasks in dependency order
    for $a \in F_\tau$ **do**
      $t \leftarrow$AGENTPREVIOUSTASK(a,τ)
      $S_i = \max(C_t + R_{a,t,\tau}, S_i)$
    end for
    for $t \in \mathrm{P}_\tau^{\mathrm{cps}}$ **do** $S_i = \max(C_t, S_i)$
    end for
    for $t \in \mathrm{P}_\tau^{\mathrm{sps}}$ **do** $S_i = \max(S_t, S_i)$
    end for
    $C_i = S_i + A_{\tau,f_\tau,F_\tau}$
    for $t \in \mathrm{P}_\tau^{\mathrm{cpc}}$ **do** $C_i = \max(C_t, C_i)$
    end for
    for $t \in \mathrm{P}_\tau^{\mathrm{spc}}$ **do** $C_i = \max(S_t, C_i)$
    end for
  end while
  return $C_{|T|}$
end function

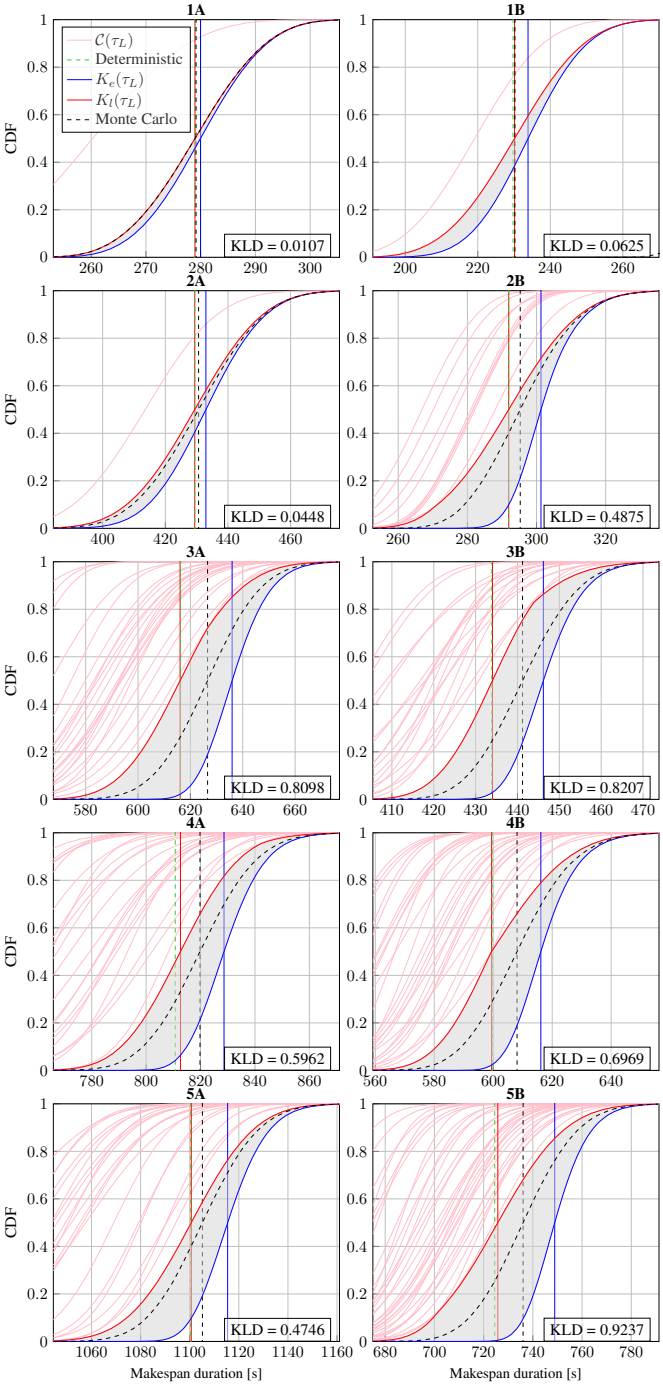

Figure 11.2: Makespans of best schedules found.

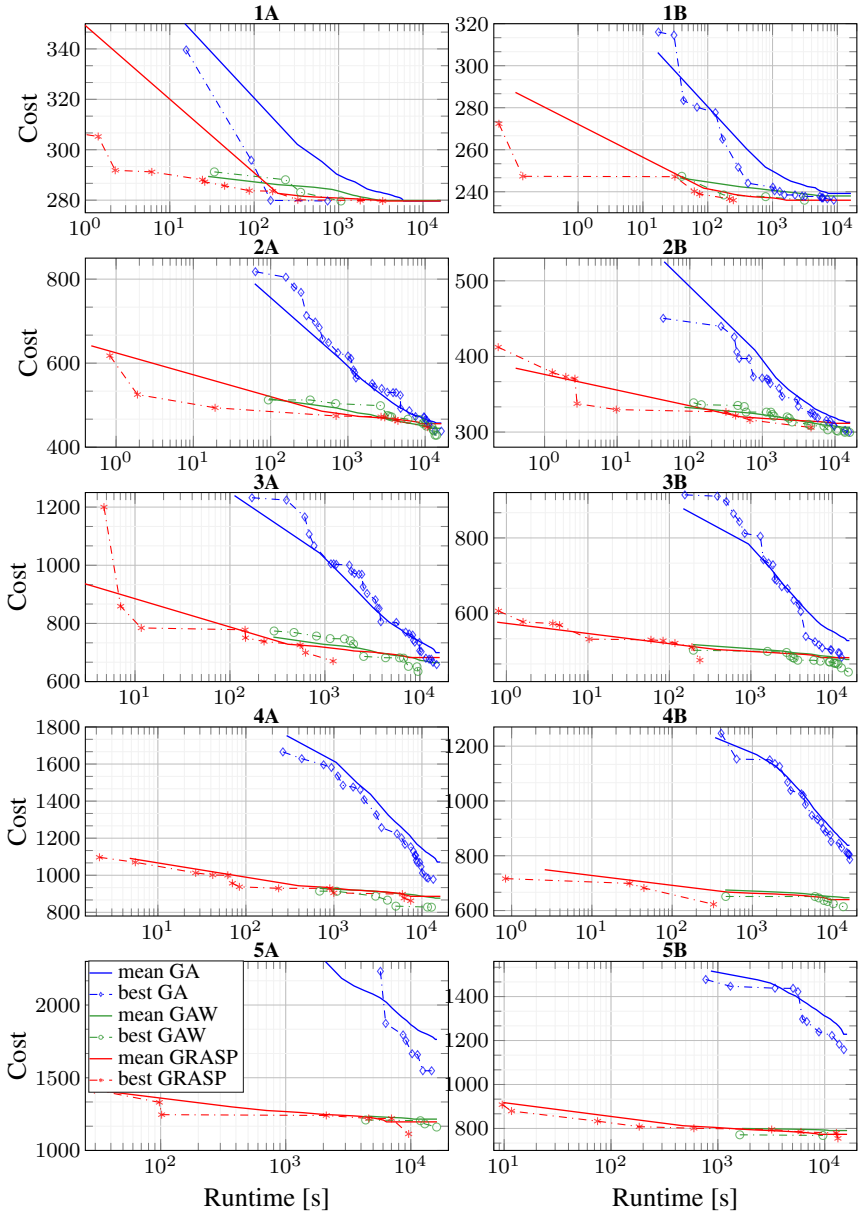

Figure 11.3: Convergence of scheduling algorithms.

11.5.1 Numerical Results

Ten use cases, summarized in Table 11.2, are set up with different numbers of tasks and agents. Routing and task durations are modeled as uniform distributions with randomized intervals. While these data are simplified, the methodology supports a real-world scenario with unrestricted distributions modeled from empirical data. All use cases are based on a process model exemplified in Section 11.2. For use cases 2A and 2B, each with 22 tasks, the constructed task network is illustrated in Figure 11.1. For larger problem instances, the increase of tasks is accomplished by an increased length of the order list, which expands the task network vertically. All tasks have 3 alternative team formations with different agent combinations: a single robot, a single human, or a robot with a human supervisor. Assemble tasks have a 4th team formation with a collaborating robot-human pair. Desired human idle time is 40%, and all humans prefer *Assemble* activities in any team formation and the role of supervisor for all activities except *Fetch*. In a real-world scenario, the methodology enables different value selection strategies, e.g., from direct human input. The selected risk value, $k = 50$, i.e., a medium risk in the objective Eq. (11.11).

MC simulations are used to validate the makespan computation approach by representing the ground truth. An MC distribution is generated from 100 thousands of deterministic makespan computations, using Algorithm 8, where input task and routing durations are randomized from their modeled distributions. To account for PCs, Algorithm 8 accumulates task and routing durations in the dependency order identified by the deadlock Algorithm 7. Makespans for the best schedules found (with any algorithm) are illustrated in Figure 11.2. They include makespan bounds, completion durations for critical paths, and MC distributions. The difference of the makespan bounds, $K_l(\tau_L)$ and $K_e(\tau_L)$, is illustrated by the gray area between their CDFs and by their Kullback-Leibler Divergence (KLD). Additionally, the graphs indicate the makepan of a deterministic approach computed with Algorithm 8 using median values of the modeled input task and routing durations.

The parameters of GRASP and GA were manually tuned to limit the convergence time while providing good solutions. GRASP randomly selects a combination from the 4 best heuristically ranked alternatives, with selection probabilities 8/15, 4/15, 2/15, and 1/15 from the best to the fourth best. GA uses a population of 200 with 5% elites, and each mutation type occurs with a probability of 30%. The algorithms are run 20 times for each use case with different seeds. Each run is limited to 270 minutes. A comparative analysis of algorithmic convergence over time is presented in Figure 11.3. It includes a warm-started variant of GA, denoted GAW, whose initial generation is created

with GRASP solutions.

11.5.2 Evaluation discussion

The makespan graphs show the MC distribution being wedged between $K_l(\tau_L)$ and $K_e(\tau_L)$ for all use cases, as expected from Theorem 1. Consequently, the usage of $K_e(\tau_L)$ in the objective, Eq 11.11, guarantees the makespan is never underestimated for a schedule. Additionally, the usage of $K_l(\tau_L)$ for the estimate of idle time quota, Eq. 11.8, guarantees human idle time never is overestimated. The estimation accuracy, e.g., represented by the KLD value, depends on the $\mathcal{C}(\tau_F)$ distributions and their degree of mutual dependence. For these examples, the accuracy has a variation among use cases without a clear correlation with problem size. The deterministic approach tends to underestimate the median makespan and generally suffers from the missing representation of uncertainty. The fast convergence of GRASP compared to GA demonstrates the efficiency of the proposed GRASP heuristics. The capacity of GRASP to find good solutions faster is demonstrated in all use cases. The GAW approach clearly benefits from the GRASP performance, with a convergence starting from a more optimized population than GA. In general, GA is outperformed by both GRASP and GAW. For the smallest use cases $1A$ and $1B$, all algorithms converge to the solutions of equal cost, but the actual plans are not necessarily the same. GAW finds the best solutions for intermediate use cases $2A$ through $4B$, while GRASP finds the best solutions for $5A$ and $5B$. On average, the difference between GRASP and GAW is generally small without a clear advantage to either of them unless a really fast solution is needed, which would make GRASP the better alternative.

11.6 Conclusion

In this paper, we presented a framework for scheduling in human-robot collaborative manufacturing environments. Our approach models the complex interplay between stochastic task and routing durations, precedence constraints, and human-centric factors such as ergonomic preferences and idle time requirements. By formulating the scheduling problem as an optimization model that accounts for uncertainty and potential deadlocks, we have established a robust basis for effective multi-agent task allocation.

Our contributions demonstrate that the integration of stochastic modeling with human-centric scheduling can significantly enhance the reliability of scheduling strategies in resilient manufacturing systems by producing realistic plans. Future work will focus on extending the framework to support real-time

scheduling adjustments and incorporating more detailed risk models, as well as validating the approach in real-world industrial settings.

Overall, the proposed framework advances the state of the art in collaborative robotics by providing a scalable, adaptable solution to the challenges inherent in human-robot manufacturing systems.

Bibliography

[1] Luca Gualtieri, Ilaria Palomba, Erich J Wehrle, and Renato Vidoni. The opportunities and challenges of sme manufacturing automation: safety and ergonomics in human–robot collaboration. *Industry 4.0 for SMEs: Challenges, opportunities and requirements*, pages 105–144, 2020.

[2] Brian P Gerkey and Maja J Matarić. A formal analysis and taxonomy of task allocation in multi-robot systems. *The Int. Journal of Robotics Research*, 23(9):939–954, 2004.

[3] Andrew W Palmer, Andrew J Hill, and Steven J Scheding. Stochastic collection and replenishment (SCAR): Objective functions. In *IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, pages 3324–3331, 2013.

[4] Anders Lager, Branko Miloradović, Giacomo Spampinato, Thomas Nolte, and Alessandro V. Papadopoulos. Risk-aware planning of collaborative mobile robot applications with uncertain task durations. In *IEEE Int. Conf. Robot and Human Interactive Communication*, pages 1191–1198, 2024.

[5] Michael Saint-Guillain, Tiago Vaquero, Steve Chien, Jagriti Agrawal, and Jordan Abrahams. Probabilistic temporal networks with ordinary distributions: Theory, robustness and expected utility. *Journal of Artificial Intelligence Research*, 71:1091–1136, 2021.

[6] Shaobo Zhang, Yi Chen, Jun Zhang, and Yunyi Jia. Real-time adaptive assembly scheduling in human-multi-robot collaboration according to human capability. In *IEEE Int. Conf. Robotics & Automation*, pages 3860–3866, 2020.

[7] Shushman Choudhury, Jayesh K Gupta, Mykel J Kochenderfer, Dorsa Sadigh, and Jeannette Bohg. Dynamic multi-robot task allocation under uncertainty and temporal constraints. *Autonomous Robots*, 46(1):231–247, 2022.

[8] Stephen B Stancliff, John Dolan, and Ashitey Trebi-Ollennu. Planning to fail: Reliability needs to be considered a priori in multirobot task allocation. In *IEEE Int. Conf. Systems, Man and Cybernetics*, pages 2362–2367, 2009.

[9] Costanza Messeri, Gabriele Masotti, Andrea Maria Zanchettin, and Paolo Rocco. Human-robot collaboration: Optimizing stress and productiv-

ity based on game theory. *IEEE Robotics and Automation Letters*, 6(4):8061–8068, 2021.

[10] Yee Yeng Liau and Kwangyeol Ryu. Genetic algorithm-based task allocation in multiple modes of human–robot collaboration systems with two cobots. *The Int. Journal of Advanced Manufacturing Technology*, 119(11):7291–7309, 2022.

[11] Michele Lombardi and Michela Milano. Optimal methods for resource allocation and scheduling: a cross-disciplinary survey. *Constraints*, 2012.

[12] Thomas A Feo, Mauricio GC Resende, and Stuart H Smith. A greedy randomized adaptive search procedure for maximum independent set. *Operations Research*, 42(5):860–878, 1994.

[13] Branko Miloradović, Baran Çürüklü, Mikael Ekström, and Alessandro Vittorio Papadopoulos. GMP: A genetic mission planner for heterogeneous multirobot system applications. *IEEE Transactions on Cybernetics*, 52(10), 2021.

[14] Carlos Ocampo-Martinez et al. Energy efficiency in discrete-manufacturing systems: Insights, trends, and control strategies. *Journal of Manufacturing Systems*, 52:131–145, 2019.

[15] Ernesto Nunes, Marie Manner, Hakim Mitiche, and Maria Gini. A taxonomy for task allocation problems with temporal and ordering constraints. *Robotics and Autonomous Systems*, 90:55–70, 2017.

[16] Branko Miloradović, Mirgita Frasheri, Baran Cürüklü, Mikael Ekström, and Alessandro Vittorio Papadopoulos. TAMER: Task allocation in multi-robot systems through an entity-relationship model. In *Int. Conf. Principles and Practice of Multi-Agent Systems*, pages 478–486, 2019.

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

[18] Moshe Shaked and J. George Shanthikumar, editors. *Univariate Stochastic Orders*, pages 3–79. Springer New York, 2007.

[19] Branko Miloradović, Baran Çürüklü, Mikael Ekström, and Alessandro V. Papadopoulos. A genetic algorithm approach to multi-agent mission planning problems. In *Operations Research and Enterprise Systems*, pages 109–134, 2020.

Appendix

Definition 9 (Comonotonicity)**.** *Two random variables X and Y are said to be **comonotonic** if there exists a common underlying random variable U and two non-decreasing functions f and g such that:*

$$X = f(U) \quad \textit{and} \quad Y = g(U),$$

For comonotonicity, there are a few relevant properties, including the following.

Joint Cumulative Distribution Function: The joint cumulative distribution function of X and Y is given by:

$$F_{X,Y}(x, y) = \min(F_X(x), F_Y(y)),$$

where $F_X(x)$ and $F_Y(y)$ are the marginal cumulative distribution functions of X and Y, respectively.

Copula: The copula of comonotonic random variables corresponds to the *Fréchet–Hoeffding upper bound*:

$$C(u, v) = \min(u, v), \quad \text{for } u, v \in [0, 1].$$

Maximum Positive Dependence: Comonotonicity represents the strongest form of positive dependence:

- The rank correlation (Spearman's ρ) and linear correlation (ρ) are maximal.
- The conditional distribution of Y given $X = x$ is deterministic.

Theorem 2 (Stochastic Dominance of Maxima)**.** *Let X and Y be nonnegative random variables (not necessarily identically distributed) with continuous cumulative distribution functions (CDFs) F_X and F_Y, respectively. Consider three random vectors (X_{ind}, Y_{ind}), (X_{com}, Y_{com}), and (X_{dep}, Y_{dep}) such that:*

1. *$X_{ind} \sim X$, $Y_{ind} \sim Y$, and X_{ind} and Y_{ind} are independent.*
2. *(X_{com}, Y_{com}) is a comonotone coupling of X and Y.*
3. *(X_{dep}, Y_{dep}) is any other positively dependent coupling of X and Y with copula C_{dep} satisfying $\forall u, v \in [0, 1]$:*

$$C_{ind}(u,v) = uv \leq C_{dep}(u,v) \leq \min(u,v) = C_{com}(u,v).$$

Let us define the maxima $Z_{ind} := \max(X_{ind}, Y_{ind})$, $Z_{com} := \max(X_{com}, Y_{com})$, *and* $Z_{dep} := \max(X_{dep}, Y_{dep})$.

Then the following stochastic order holds:

$$Z_{com} \leq_{st} Z_{dep} \leq_{st} Z_{ind},$$

which means that for all $z \geq 0$*:*

$$P(Z_{ind} > z) \geq P(Z_{dep} > z) \geq P(Z_{com} > z).$$

Proof. Independent case:

$$P(Z_{\text{ind}} \leq z) = P(X_{\text{ind}} \leq z, Y_{\text{ind}} \leq z).$$

Since X_{ind} and Y_{ind} are independent:

$$F_{Z_{\text{ind}}}(z) = F_X(z)F_Y(z).$$

Comonotonic case:

$$P(Z_{\text{com}} \leq z) = P(F_X^{-1}(U) \leq z, F_Y^{-1}(U) \leq z).$$

Since $F_X^{-1}(U) \leq z \iff U \leq F_X(z)$ and similarly for Y, it follows that:

$$F_{Z_{\text{com}}}(z) = \min(F_X(z), F_Y(z)).$$

Positively Dependent case: For any positively dependent coupling:

$$F_{Z_{\text{dep}}}(z) = C_{\text{dep}}(F_X(z), F_Y(z)).$$

Given the bounds on the copula:

$$F_X(z)F_Y(z) \leq C_{\text{dep}}(F_X(z), F_Y(z)) \leq \min(F_X(z), F_Y(z)).$$

Therefore, we conclude that:

$$F_{Z_{\text{ind}}}(z) \leq F_{Z_{\text{dep}}}(z) \leq F_{Z_{\text{com}}}(z),$$

which implies:

$$P(Z_{\text{ind}} > z) \geq P(Z_{\text{dep}} > z) \geq P(Z_{\text{com}} > z), \forall z \geq 0.$$

□