Scheduling of Missions with Constrained Tasks for Heterogeneous Multi-Robot Systems

Gricel Nidteja Vázquez Flores

PhD

University of York

Computer Science

April 2024

Coming together is the beginning.
Keeping together is progress.
Working together is success.

Henry Ford

Abstract

Applications of multi-robot systems (MRS) have been widely considered in domains ranging from healthcare and manufacturing to search-and-rescue. These applications involve the execution of complex missions comprising interdependent tasks with complicated constraints and conflicting optimisation objectives. One of the most challenging MRS problems is therefore the synthesis of robot plans that comply with these complexities. Despite significant research, existing solutions do not provide end-to-end methodologies capable of addressing this MRS planning problem for realistic sets of complex constraints and optimisation objectives. Furthermore, current solutions rarely employ formal methods to provide guarantees on the compliance of their MRS plans with such requirements.

This thesis addresses the limitations summarised above by proposing an end-to-end, tool-supported MRS task allocation and scheduling approach (KANOA). KANOA tackles the allocation of mission tasks to, and the synthesis of *plans* (i.e., task schedules) for, the heterogeneous robots of an MRS by using a combination of formal methods over several stages. To that end, KANOA comes with a domain-specific language (DSL) for the high-level description of MRS planning problems with: (i) realistic constraints on the sequencing of tasks, and on permitted robot workloads and areas of operation; and (ii) conflicting optimisation objectives (maximising the probability of successful mission completion, minimising the robot idling time, and/or minimising the mission duration).

In the first stage of the KANOA approach, this DSL is used to specify the MRS planning problem of interest, and to formalise it through mapping to Z notation. In a second stage, KANOA uses this Z-notation encoding to formulate the allocation of the mission tasks to the robots of the MRS as a constraint-solving problem, which is then solved through model synthesis using the Alloy model finder. Finally, KANOA employs a combination of probabilistic model checking and closed-form model analysis to schedule (i.e., to order and decide the start time of) the tasks allocated to each robot. The combined set of robot plans generated in this last KANOA stage is guaranteed to satisfy the MRS mission constraints, and to be Pareto-optimal with respect to its optimisation objectives.

Contents

List of Figures

List of Tables

To my lovely parents,
Rosalina and Misael.

Acknowledgements

I would like to thank my supervisors, Prof. Radu Calinescu and Dr Javier Cámara, for all the effort and time invested in guiding this research contribution. His thought-provoking feedback and ideas, besides his efficiency and consistency throughout my PhD, have made this thesis's materialisation possible and shaped me as a researcher. I am very grateful to have them as my supervisors.

Thank you to my assessor, Prof. Ana Cavalcanti. I learned and continue learning from her professional ethics and communication skills, as well as being supportive in every meeting we had.

Thank you to my colleagues in the ISA building and CS department for being part of my PhD journey—Brendan Devlin-Hill, Qi Zhang, Dr Xinwei Fang, Dr Calum Imrie, Dr Ioannis Stefanakos, Dr Sinem Getir Y. and Dr Chiara Picardi— and beyond—Dr Marie Farrell, Tom Pressburger and Max Fan; Dr Claudio Menghi and Prof. Patrizio Pelliccione; Marc Carwehl and Prof. Genaina Rodrigues

Thank you, Dr Anastasia Mavridou, for the great opportunity to spend a summer at NASA's Ames Research Center, at the heart of Silicon Valley, California, and learn from your brilliant work on the elicitation of formal requirements through FRET.

Thank you to Dr Jose Guadalupe Rico Espino for the encouragement to continue in research from my early research career days back in Mexico. To my bachelor's teachers, Alexander Bell Mejia and Miguel Gonzalez Ortiz, and all their interesting classes in maths and engineering.

Thank you, mom and dad, Rosalina and Misael, for being my driving force with your endless love and support all these years and the years to come. To my sibling, M., M. and K., for cheering me up from far away.

I would also like to thank the Stackoverflow.com community for helping me solve many programming bottlenecks, and not very thankful to ChatGPT for refusing to write my thesis.

Finally, I thank the Mexican Organization of the National Council of Science and Technology, CONACYT, for the financial support and the opportunity to continue my professional and personal growth. I believe that science is the door to a better future and that investment in science is an investment in the progress of society.

Declaration

Except where stated, all of the work contained in this thesis represents the original contribution of the author. I declare that this thesis is a presentation of original work and I am the sole author. This work has not previously been presented for a degree or other qualification at this University or elsewhere. All sources are acknowledged as references.

Previous versions and parts of the research described in this thesis have been previously published in:

- Vazquez, G., Mavridou A., Farrell M., Pressburger T. & Calinescu R. "Robotics: A New Mission for FRET Requirements", accepted at NASA Formal Methods (NFM) Symposium, 2024.
- Menghi C., Tsigkanos C., Askarpour M., Pelliccione P., Vazquez G., Calinescu R. and García S., "Mission Specification Patterns for Mobile Robots: Providing Support for Quantitative Properties", Transitions of Software Engineering (TSE) 2022.
- Vazquez G., Calinescu R. and Cámara J., "Scheduling of Missions with Constrained Tasks for Heterogeneous Robot Systems", Formal Methods and Autonomous Systems (FMAS), Berlin, Germany, 2022.
- Vazquez G., Calinescu R., and Cámara J., "Scheduling multi-robot missions with joint tasks and heterogeneous robot teams," in Towards Autonomous Robotic Systems: 22nd Annual Conference, (TAROS), Lincoln, UK, Proceedings 22: Springer International Publishing, pp. 354-359, September 8–10, 2021.
- Vazquez G., "Automated Scheduling of Multi-Robot System Missions: An Architectural Perspective," in European Conference on Software Architecture (ECSA), 2021.

Part of Chapter 4 is joint work with Claudio Menghi, Christos Tsigkanos, Mehrnoosh Askarpour, Patrizio Pelliccione and Sergio García. We clarify in this chapter's introduction our contributions to this collaborative project.

Chapter 1

Introduction

Robots can access environments beyond human reach, such as outer space [3], undersea regions, and hazardous environments [4]. They can also excel at perpetually performing routine tasks that might be tedious for humans, such as continuously moving goods around a warehouse [5] and providing support for personal assistance and rehabilitation [6, 7]. These situations, however, require a high level of robot coordination and robustness to deal with the uncertainties that are common in real-world environments. Multi-robot systems possess multiple benefits compared to single-robot ones. Some of these are [8, 9]: (a) resolving task complexity when a single robot requires complex skills or might be unable to complete a task on its own, this might include the distributed nature of the tasks; (b) improving reliability through redundancy by having more than one robot available; (c) increasing the overall performance as completion times can be drastically reduced by robots advancing task in parallel; and (d) simplifying robots' design by dividing the necessary capabilities to complete the tasks into simpler and cheaper robots. Over the past few years, MRSs have been employed in different environments, including agriculture [10, 11], pick-up and delivery in distribution centers [12, 13] and search-and-rescue [14–16]. As new and more complex applications are considered for systems with multiple robots, so is the need to incorporate a broader range of constraints in the definition of the missions associated with these applications. These constraints apply to the robots, tasks and environment. For example, they may restrict the space where each of the robots can work, or they may specify intricate task ordering requirements. Additional requirements for these applications may involve the optimisation of mission objectives such as maximising the probability of completing the mission without failing to travel between locations or completing the tasks, or minimising the mission execution time.

Guarantees that a correct plan will be synthesised under a complex set of constraints must also be considered. In this context, we investigate a variant of the multi-robot system task allocation and scheduling problem whose requirements consist of a complex set of constraints *and* multiple conflicting optimisation objectives. We provide solutions for the definition of tasks, task allocation and scheduling problems, and the synthesis of individual robot plans. We propose a framework that eases the adoption of new constraints and requirements as needed for the application at hand, while providing constraint-compliance guarantees and Pareto-optimal trade-offs between the optimisation objectives.

This chapter is divided as follows. Section 1.1 presents the thesis' motivation, Section 1.2 shows the research objectives and Section 1.3 describes the thesis contributions.

1.1 Motivation

The motivation behind this thesis stems from the challenging and error-prone task of elicitating formal specifications for multi-robot systems, particularly in addressing uncertainty both within these specifications and in the context of task allocation and scheduling. We are interested in the allocation and scheduling of tasks as solving these two interrelated problems is essential for the deployment of robots, yet non-trivial. This becomes increasingly complex as the number of requirements increases, both in terms of their specification and the mechanisms to find solutions. In this section, we further elaborate on the thesis's motivation.

Uncertainty in multi-robot applications. In the deployment of mobile robots, uncertainties may arise from the oversimplified models underpinning the operation of the robots, model drifting between the actual and the expected behaviour of the robots, the inaccurate specification of goals, and so on [17–19]. This wide variety of sources of uncertainty has driven the design of new taxonomies expressly for robotic systems [20], but also in related research areas such as that explored by the self-adaptative systems research community [19, 21]. Currently, many researchers consider the modelling of certain types of uncertainties using probability theory [22, 22–25] and stochastic algorithms such as simulated annealing [9]. As more complex applications for multi-robot systems (MRS) are proposed, there is a growing need to formalise and incorporate these uncertainties systematically into the underlying MRS models and specifications.

Another approach for reasoning about the probabilistic behaviour of MRS systematically involves the use of formal methods—rigorous, unambiguous and verifiable techniques applied in the design and implementation of systems [26]. Model checking, a type of formal method, has been successfully used in robotics for task allocation and planning, and for the synthesis of individual robot controllers [27, 28]. Model checking enables the verification of a set of properties on all possible states of a system model [29]. Within the family of model checking techniques, *probabilistic model checking* is a formal technique for analysing systems that exhibit stochastic behaviour under a set of properties defined in probabilistic variants of temporal logic [29]. For models that allow the specification of nondeterministic choices, such as Markov decision processes (MPDs), one can synthesise a policy (i.e., a solution to nondeterministic actions) by optimising some model-related value such as probabilities or rewards (e.g., probability of success, expected energy consumption) while satisfying some temporal specification [29].

These characteristics encouraged the use of probabilistic model checking in multi-robot systems to synthesise robot plans (policies) under a set of formally-defined properties [30]. To give an example, a team of robots helping in a healthcare setting may be required to sanitize patient rooms [31], deliver medical supplies [32], move furniture and medical equipment [7], and help nurses to dress patients [33]. Given the different robot capabilities and mission tasks, the robots should find a way to cooperate to accomplish the whole mission. The robot team must complete the tasks with a high probability of success, within a certain time window, optimising the available energy and travelling costs. Finally, probabilistic behaviour may appear, for example, when robots may fail to travel between the required locations or to complete their tasks with certain (small) probabilities estimated through testing.

Specification of MRS missions and requirements. Beyond modelling the spectrum of uncertainties in MRS, another challenge is capturing their numerous functional and non-functional requirements. These requirements can depend on the type of robots; for example, in a heterogeneous MRS the robots possess different capabilities, and must only be allocated tasks that they are capable of carrying out. In the case of mobile robots, we also assume that tasks require them to move within feasible spaces while avoiding the obstacle space. Before deployment, at design time, the description of the MRS problem must include all the necessary robotic mission requirements and relevant models of uncertainties. Hence, there is a need for languages that support the specification of a wide range of MRS requirements in an unambiguous, formal manner, but at the same time, remain user-friendly to ease their adoption.

The specification of tasks has been widely studied within the planning community with well-known solutions including Hierarchical Task Networks (HTN) and actions description languages (ADLs) such as STRIPS, PDDL, and RDDL [34]. Some extensions to these solutions, such as PDDL 1.0 [35] allow the modelling of a probabilistic distribution over the set of possible effects to an action. However, these specification languages cannot be used to describe how additional probabilistic behaviour (for instance, the probability of a robot failing to travel to a location or the probability of multiple robots succeeding to complete a task).

Another approach to the formalisation of robot tasks, as well as functional requirements, is the use of temporal logic such as Linear Temporal Logic (LTL) and Computational Tree Logic (CTL) [29]. Temporal logic provides unambiguous languages for the specification of requirements. In robotics, they have been successfully applied in the specification of robotic missions, as well as in the description of functional and non-functional mission requirements. A catalogue of qualitative robotic mission requirements specified in LTL and CTL is provided in [2,36]. One of the disadvantages of using temporal logic is that their syntax is foreign to potential users of robotic systems. Moreover, the elicitation of requirements written in such types of logic languages are susceptible to errors due to their intricate semantics, even for domain experts. Hence, higher-level domain-specific knowledge is proposed to ease their adoption [2]. A second disadvantage is that LTL and CTL do not support the modelling of important probabilistic and quantitative requirements such as "*the probability of all robots completing the mission without failure should be at least 0.99*" or "*the mission execution time should be minimised*". These properties can be specified in a type of probabilistic logic called *probabilistic computational tree logic* (PCTL) extended with *rewards*. Rewards-extended PCTL is exploited for the description of robotic mission requirements within this thesis.

Allocation of tasks to robots. When discussing the implementation of a multi-robot system, we delve into a range of challenges that need to be addressed. One of these is how to partition the tasks of an MRS mission among a team of heterogeneous robots. This issue, in itself, implicates a series of problems. Some of these include defining the tasks assigned to the robots and any task constraints; defining the set of constraints applicable to the allocation of tasks (such as bounding the number of tasks acceptable by each robot, or pre-allocating some of these tasks to a robot); and modelling as part of the task allocation problem the heterogeneity of the robots, including by defining the types of tasks that each robot can perform.

Despite the availability of numerous solutions for the MRS task allocation problem, ongoing research in this area remains active due to the variations and complexity inherent in this problem. For instance, regarding the hetereogeneity of the robots, some robots may be better suited for pick-and-place tasks than others as they have newer object detection sensors, better grippers or are equipped with more dexterous robot arms. Other robots may entirely lack the capability to perform these tasks. For this example, deploying the most suitable robots results in a higher probability of completing the mission successfully.

The task allocation problem can be solved separately or jointly with the generation of individual robot plans. A *robot plan* describes what action a robot must be doing at any point in time until the mission is completed. The process of synthesising robot plans can be viewed as a sequence of problems to be solved. As mentioned before, solutions consider one of three options: (a) synthesize robot plans by first resolving the allocation of tasks to robots and then addressing the scheduling of these tasks to create individual robot plans that comply with a given set of constraints; (b) model the allocation and scheduling of tasks jointly and solve them concurrently; or (c) assume that the tasks are pre-allocated to robots, transforming the problem into a scheduling problem. By scheduling problem, we mean that each robot must know in which order to perform its tasks while complying with a set of mission requirements. These constraints may involve maintaining some prescribed ordering of the task [37] or completing the mission within a specified time limit [30].

Partitioning the plan synthesis into two distinct problems, namely task allocation and scheduling, mitigates the complexity associated with the use of a potentially very large monolithic model. Additionally, we argue that using this approach supports the modelling and analysis of a broader set of requirements, as some constraints may belong exclusively to either the task allocation or the scheduling part. For instance, if the decision is made to restrict the number of tasks assigned to each robot to promote a "fair" distribution of tasks, this constraint is relevant to the allocation problem but not the scheduling aspect. Consequently, in this thesis, we exploit these advantages and approach the synthesis of robot plans as a combination of two distinct problems: allocation and scheduling of tasks.

In recent years, extensive research has been conducted on the task allocation problem. Well-known multi-robot task allocation (MRTA) algorithms include the optimal assignment problem [9] and market-based approaches [9, 38]. However, the current solutions do not provide an easy solution to adapt their problem formulation in order to systematically add new types of requirements; for example, by providing repositories of such requirements. Moreover, they do not provide guarantees that an allocation can be found when faced with a complex set of mission constraints.

Scheduling of tasks and robot-plan synthesis. Task allocation and scheduling problems solve the *who* and *in which order* questions, respectively. These problems are known to be strongly NP-hard[1] [39]. To schedule n_t tasks assigned to a robot, there are $n_t!$ possible solutions. For n_r allocated with the same number of tasks, the solution space grows to $n_r \times n_t!$. Moreover, once the permutation is done, we must ensure that task dependencies among robots comply, for example, as two or more robots must meet at the same time and space. We refer to this problem as plan synthesis. Synthesizing robot plans is a non-trivial problem, yet necessary for robots to know *what to do* at every moment during deployment. The plan synthesis complexity arises from existing task constraints, the coordination of robots in space and time, multiple conflicting optimisation objectives, time and spatial constraints [8], times required to travel between tasks, and uncertainties in the system that have an impact on the completion of the tasks [41].

Once the tasks are allocated to the robots, we must deal with how to schedule the tasks assigned to each robot. Some constraints relevant to the scheduling problem involve the sequencing of tasks. For instance, a robot must first pick up the medicine before delivering it to a patient. Other requirements relevant to the scheduling phase may include predetermined completion time windows, requiring the completion of tasks within specific time frames; limitations on the available time; or constraints on the total travelling cost incurred by the robots as they move between task locations. Scheduling the tasks assigned to each robot generates an ordered sequence of these tasks. However, the plans intended for the deployment of the robots must also account for travel and idle actions. The need for idling arises, for instance, when two robots are instructed to meet at a location in order to carry out a joint task (such as moving a piece of furniture that can only be lifted by two robots working together), but one arrives at this location earlier, requiring it to idle until the second robot arrives.

[1]The only not NP-hard is when no dependencies exist between tasks, robots execute a single task (ST) at a time, tasks require a single robot (SR), and the tasks are instantaneously assigned (IA) without future tasks information [39, 40]. It's important to highlight that in these studies, task allocation indirectly solves the scheduling of tasks.

Different variations of the scheduling problem have resulted in a wide range of problem formulations and solutions within and outside the robotics research community. Two of the non-robotics research areas that tackled this problem are the scheduling of computation tasks (e.g., on a computer cluster) and the scheduling of manufacturing processes [42]. Classical scheduling techniques, such as the job shop problem [43] and multiple mixed integer programming [41,43], have also been studied for MRS. Planners such as STRIPS [44] and Hierarchical Task Network planners [45] provide another set of solutions to the robotic task-scheduling problem. However, these solutions are very restrictive in their problem formulations, so changes to accommodate the types of constraints we are interested in in this thesis require significant research work. Moreover, even defining the problem to be solved requires domain experts to specify all the intricate relations between tasks, pre-conditions, actions, etc. We aim to simplify the specification of multi-robot task scheduling problems and facilitate their automatic translation into (possibly probabilistic) formal models and requirements; indirectly mitigating the error-prone process of their elicitation.

As the need for more complex scheduling problem formulations grows, so does the need to make these approaches flexible enough to capture a greater diversity of mission-related requirements, constraints and optimisation objectives. For example, consider a safety-critical search-and-rescue scenario where a fleet of robots must locate individuals who are in danger. To make the system more resilient, designers must model multiple uncertainties coming from different sources, such as the unknown distance to the victims, the uneven terrain reducing the probability of the robots travelling successfully between search locations, etc. These different aspects have to be captured in the problem description and the solution for the synthesis of individual robot plans. Furthermore, when multiple solutions and conflicting optimisation objectives are present, achieving effective trade-offs between these objectives becomes important.

Theoretically, addressing the permutation of tasks across N robots falls in the category of non-deterministic polynomial problems. For instance, in the shop floor problem, there are $n!m$ possibilities to schedule n tasks (pre-allocated) to m robots [46]. As the applications of multi-robot systems become more challenging and demand the use of more complex models for task allocation and scheduling, the scalability issue must be carefully assessed when proposing new task scheduling techniques for multi-robot systems. Some solutions recur to heuristic techniques to sample the solution space [47–49]. Some meta-heuristics sampling techniques, such as Genetic Algorithms (GA), can guide the search to obtain Pareto-optimal solutions. In this thesis, we employ GA and a series of specialised models to obtain Pareto-optimal robot plans that optimise one or more conflicting optimisation objectives, while complying with a set of complex mission constraints.

1.2 Research Objectives

We have described the key challenges faced in the MRS design pipeline: from the specification of the problem, to the MRS mission task allocation and scheduling and the synthesis of individual robot plans. This thesis presents research that addresses multiple limitations in the way in which existing solutions handle these challenges. Given a high-level description of a series of tasks to be performed by a group of heterogeneous mobile robots, with multiple constraints and optimisation objectives to be considered in the synthesis of the robot plans, the research objectives of this thesis are as follows:

1. *To ease the definition of an MRS task allocation and scheduling problem for realistic sets of complex constraints and optimisation objectives, including the description of robots exhibiting probabilistic behaviour.*
2. *To devise an MRS task allocation method capable of managing a diverse set of constraints, including the heterogeneity of robots' capabilities and spatial limitations, and to design a subsequent task scheduling method that generates individual robot plans optimised to meet multiple mission-critical requirements.*
3. *To integrate the methods from Objective 2 into an end-to-end methodology for MRS task allocation and scheduling that employs formal methods at each stage of its robot schedule generation process.*
4. To provide tool support for the adoption and testing of the developed techniques for the definition of MRS specifications and the generation of task schedules.

1.3 Contributions and Thesis Structure

To achieve the research objectives summarised in the previous section, this thesis introduces KANOA, a novel end-to-end framework for the synthesis of multi-robot plans from high-level mission specifications. KANOA allows the specification of multiple optimisation objectives and a complex set of functional and non-functional mission-related requirements applicable at different stages of the MRS plan synthesis, and employs formal methods in novel ways at each stage of the MRS task allocation and scheduling process.

The main contributions of the research presented in this thesis are as follows:

- **A systematic review on the state-of-the-art research on allocation and scheduling in robotic systems** (Chapter 3). Chapter 3 presents a comprehensive review of the research landscape on task allocation and scheduling for robotic systems. Through this systematic review, we untangle the intricate variety of terms used for variations of the MRS task allocation and scheduling problems and their proposed solutions. The review identifies key trends, challenges, and the algorithms and methodologies that have shaped the field in recent years. We also identify a series of research gaps to establish a foundation for future work.

- **Formalisation of a repository of robotic mission patterns from high-level specifications** (Chapter 4). We present the formalisation of QUARTET, an extensive repository of quantitative mission patterns for mobile robots. We use probabilistic computational tree logic augmented with rewards to unambiguously define quantitative requirements applicable to functional and non-functional aspects of the system. A tool supporting the translation from a higher-level language into this logic is also delivered. As this contribution was carried out as part of a collaborative project [1], Section 4.1.1 describes the components of this research that were carried out as part of our thesis.

- **KANOA problem specification for multi-robot systems** (Chapter 4). In addition to the previous contribution, Chapter 4 describes the multi-robot plan synthesis problem considered by KANOA. Designed in a separation-of-concerns fashion, KANOA's problem specification comprises separate modules for defining the MRS world (i.e., environment), robots, tasks and mission. This modular specification allows for the definition of a vast range of both functional and non-functional requirements, providing designers the flexibility to integrate them based on the specific part to which these requirements apply. Furthermore, we introduce a user-friendly domain specification language designed to facilitate the adoption of KANOA, we formalize KANOA's problem using the Z notation, and we present a scenario of robots moving surgical equipment and visiting patients in a hospital case study that serves as a motivating scenario throughout this thesis.

- **Task allocation as a constraint-solving problem** (Chapter 5). We specify KANOA's task allocation as a constraint-solving problem. This enables the use of a wide range of optimised techniques and tools for the generation of feasible allocation instances, such as the Alloy Analyser constraint solver. Compared to other techniques, the aim is to guarantee that allocations comply with any applicable (to the task allocation) requirements is to be found even in the presence of multiple constraints. We demonstrate the applicability of our solution in the hospital case study.

- **Task scheduling and plan generation** (Chapter 6). KANOA uses a combination of formal techniques for the generation of a set of Pareto-optimal robot plans. First, during a pre-scheduling phase, robots are organized into groups based on whether they have shared dependencies in their allocated tasks. Next, in a scheduling and plan synthesis phase, we assemble: (a) a discrete-time Markov chain (DTMC) to reason about the minimisation of the total robot idle time; (b) an MDP to devise plans that maximise the probability of mission success; and (c) an analytic model to compute the total travelling cost. We use probabilistic model checking to analyse key properties (specified in probabilistic temporal logic augmented with rewards) of the first two models. Inspired by the EvoChecker approach [50], we employ multiobjective optimization genetic algorithms to automate the generation of robot plans. This produces Pareto-optimal solutions for each task allocation, from which the plan for deployment of the robots is eventually selected.

- **KANOA's tool, framework and evaluation** (Chapter 7). Finally, we present the tool-supported KANOA's framework for the synthesis of robot plans from the high-level specifications described in the KANOA's DSL, and we evaluate KANOA in Chapter 7.

Defining the contributions' scope and impact. The open problem addressed in this thesis involves providing methods to specify unambiguous MRS requirements and ways to solve specific formulations of the MRS allocation and scheduling problems, leading to the synthesis of formally verified multi-robot plans under complex constraints and optimisation objectives. These include (a) guaranteeing the overall mission success, accounting for the possibility of robots probabilistically failing to complete tasks, or to travel between locations and possible task retries; (b) addressing task ordering constraints and handling joint tasks where multiple robots must coordinate in time and space to work together; (c) incorporating time constraints for the initiation and completion of tasks, as well as for the overall mission duration; (d) optimising mission-related deterministic and probabilistic metrics, such as maximising the probability of mission success and minimising the total travelling cost.

Following the well-known Gerkey & Mataric's taxonomy [39, 51] for task allocation problems, our work targets single-task, multi-robot time-extended assignment (ST-MR-TA). Therefore, we make the following **assumptions** for the types of MRS problems targeted within this thesis:

- single tasks: each robot is capable of executing at most one task at a time (hence, advancing multiple tasks at a time requires more than one robot);
- multi-robot tasks: some tasks might require multiple robots to be completed;
- time-extended assignment: the allocation of tasks considers information of all tasks to be assigned to the robots and their time and ordering constraints;
- centralised architecture: there exists a single coordinator in charge of the task allocation and scheduling computation;

- collision-free behaviour: a lower-level behavioural controller is in place to avoid collisions between robots or the environment;
- motion planning independent: a planning system is in place to ensure the robots' movement between task locations;
- number of retries allowed: the number of retries allowed for each task is predefined and cannot be modified;
- stationary probability: the probability of robots failing while travelling and completing tasks are assumed stationary and known in advance.

Providing guarantees for probabilistic and non-deterministic plan properties can be computationally expensive and time-consuming [29]. Recognising that the formal verification of plans involves high computational overheads, this thesis focuses on the synthesis of robot plans during the design phase of the multi-robot system (MRS) development lifecycle. This stage occurs before the runtime activities required post-deployment, when time is critical, and new schedules must be generated within seconds [52].

Our work was done with robotics and systems engineering stakeholders in mind. Hence, we aimed to advance the state-of-the-art by developing methods for generating robot plans that meet a specified list of requirements while ensuring a degree of optimality; and proposing a methodology that combines these methods, supporting their adoption and future development.

The remainder of this thesis is organised as follows. Chapter 2 presents the background concepts and terminology used in the rest of the thesis. Chapter 3 provides a systematic review of the task allocation and scheduling problems in multi-robot systems. Chapter 4 presents a repository of quantitative robotic mission specifications, and the problem specification for task allocation and scheduling multi-robot systems, called KANOA. Chapter 5 describes KANOA's task allocation process where robots are assigned a set of tasks complying with any applicable constraint from the problem specification. The pre-allocation and task allocation are described in Sections 5.2 and 5.3, respectively. Chapter 6 describes KANOA's task scheduling, a hybrid approach combining meta-heuristic search to explore the solution space, and probabilistic model checking (PMC) to quantitatively reason about the quality of the solutions. The pre-allocation and task allocation are described in Sections 6.2 and 6.3, respectively. KANOA's tool implementation and evaluation are presented in Chapter 7. Finally, Chapter 8 concludes the thesis by summarising the findings of this research and providing directions for future work.

Chapter 2

Background

The study of robotic systems demands a multifaceted approach to the description, design, and analysis of their intricate components, functionalities and missions. In this chapter, we present a series of techniques successfully used within this thesis for this purpose. Section 2.1 presents the fragment of Z notation used in this thesis for the formal description of multi-robot systems. Section 2.2 introduces a constraint solver called Alloy Analyzer, and its associated Alloy language for the declarative description of systems. The Alloy Analyzer is used in the thesis for the generation of feasible allocations of tasks to robots. Section 2.3 presents background on the verification of probabilistic systems, specifically on probabilistic model checking (PMC), a technique supporting the analysis of systems with stochastic behaviour. This section also describes two types of probabilistic models (Sections 2.3.1 and 2.3.2) and logic languages (Section 2.3.3), which we will use for the description of robotic mission requirements and optimisation objectives. Section 2.3.2 also presents the synthesis of Markov decision process (MDP) *adversaries* (also known as strategies or policies), which are used within this thesis for the synthesis of individual robot plans in a multi-robot mission. Section 2.3.4 introduces PMC, and describes PRISM, a tool (and language) for the automatic verification of models through PMC. Lastly, Section 2.4 concludes the chapter with a description of evolutionary-guided synthesis of MDP adversaries, a method that we exploit to generate robot plans in the thesis.

2.1 Z Notation

Z notation is a formal language and a style for the description of systems based upon mathematical logic, set theory and the concept of schema representations as a key feature. The advantage of using Z notation for this purpose is that it is precise and unambiguous, avoiding the mistake of hidden information and details flaws that are difficult to foresee in other documentation methods such as text, pictures or diagrams [53].

The Z notation includes a mechanism for defining and combining schemas by schema calculus. Hence, large specifications can be partitioned into multiple schemas and built up in stages, easing readability, reusability of components and maintenance.

In this section, we introduce Z notation concepts that are later used in the definition of multi-robot mission specifications. We do not provide a full description of the capabilities and syntax of the Z notation, but rather an introduction detailed enough to understand its application in Section 4.2.

We introduce the concepts with a simple example. For our example, there exist robots described by an identifier and a location, and each location also has an identifier. We declare these basic types as:

$$[\mathit{RobotID}, \mathit{LocationID}]$$

These objects represent sets self-defined that do not need any further reference to the type of objects they contain. We define a schema to describe a state space of the system. A schema contains two parts, separated by a central diving line in the Z graphical notation. The top part allows for the declaration of variables, and the bottom defines the relationships and constraints for the values of these variables, stated as predicates.

$$\begin{array}{|l}
\multicolumn{1}{l}{\mathit{SchemaName}} \\ \hline
\mathit{variabledeclarations} \\ \hline
\mathit{predicates} \\ \hline
\end{array}$$

In our example, we are interested in defining robots and location schemas: the robot schema contains two variables. The robot identifier *id* associated with variables in the set *RobotID*, and the velocity of the robot which is a value in the set of natural numbers $\mathbb{N}$ and is constrained to under 10 units per minute.

$$\begin{array}{|l}
\multicolumn{1}{l}{\mathit{Robots}} \\ \hline
\mathit{id} : \mathit{RobotID} \\
\mathit{velocity} : \mathbb{N} \\ \hline
\mathit{velocity} < 10 \\ \hline
\end{array}$$

The location schema has only two variables describing its identifier and its (x, y) coordinates.

$$\begin{array}{|l}
\multicolumn{1}{l}{\mathit{Location}} \\ \hline
\mathit{id} : \mathit{LocationID} \\
x, y : \mathbb{N} \\ \hline
\end{array}$$

Note that the set of natural numbers $\mathbb{N}$ is predefined in Z notation and does not need to be explicitly defined. We can extend the *Location* schema with a relationship where, for any two locations, *their identifiers are the same if and only if the two locations are the same*,

$$\forall\, l1, l2 : \mathit{Location} \bullet l1.\mathit{id} = l2.\mathit{id} \Leftrightarrow l1 = l2$$

We can also define relations between schema variables. For example, the world model containing all locations and the distance between two locations is specified as,

$$\begin{array}{|l}
\textit{World} \\ \hline
\textit{locations} : \mathbb{P}\,\textit{Location} \\
\textit{dist} : \textit{LocationID} \times \textit{LocationID} \twoheadrightarrow \mathbb{N} \\ \hline
\end{array}$$

where *LocationID* $\times$ *LocationID* defines the set of pairs of location identifiers, and $\twoheadrightarrow$ is a partial function mapping it to a natural number. The operator $\mathbb{P}$ is the power set operator, hence *locations* is defined as a set of *Location*.

We can create axiomatic definitions for each robot identifier, locations and location identifier instances as follows,

$$\begin{array}{|l}
r1, r2, r3, r4, r5 : \textit{RobotID} \\
loc1, loc2, loc3, loc4, loc5 : \textit{LocationID} \\
LOC1, LOC2, LOC3, LOC4, LOC5 : \textit{Location} \\ \hline
LOC1.id = loc1 \land LOC1.x = 0 \land LOC1.y = 0
\end{array}$$

In the last schema, we also wrote as an example the values of *Location LOC1* such that its elements are $id = loc1$, $x = 0$ and $y = 0$.

Z notation for MRS. We decided to use Z notation to formalise the different building blocks of the task allocation and scheduling problem specification. Its strong mathematical foundation allows for the precise definition of system components and their interactions. By employing Z notation, we can rigorously specify the different elements of the MRS, such as tasks, robots, locations, and the constraints that govern their interactions. The use of separate schemas in Z notation facilitates a clear separation of concerns, making the system's structure more understandable and manageable. This separation allows us to independently validate individual components (through tools such as Fuzz [54]), ensuring that each part of the system adheres to its specified behaviour before integrating them into the larger system.

2.2 Alloy Language and Alloy Analyzer

Alloy is a formal specification language (supported by the *Alloy analyzer* tool) that has its roots in the Z specification language and that can describe structural properties of systems succinctly. Alloy is defined in terms of simple relational semantics, and constraints, employing constructs that are common in object-oriented notations. These constructs are based on abstract models that are defined in terms of *data domains*, and *operations* between those domains [55].

We introduce some of the standard features of the Alloy specification language using examples from the multi-robot system domain. Suppose that we want to specify a model of the domain at an abstract level, in which we have robots (and robot capabilities), rooms (e.g., within a hospital) where the robots need to perform a mission, and tasks. We can start by indicating the existence of a set (of atoms) for robots and their capabilities, which in Alloy are specified using *signatures*,

```
abstract sig Robot {hascapability: some Capability}
abstract sig Capability {belongsto: one Robot}
```

In the signature definitions above, Robot.hascapability denotes a relation with a non-empty set of capabilities that a robot possesses. The keyword some in Alloy indicates that the set of capabilities denoted by hascapability has to contain at least one atom (we assume that it would not make sense to have a robot without any capabilities). In contrast, Capability.belongsto denotes a relation that states that a capability must belong to one (and only one) robot (as indicated by the keyword one in the declaration). The keyword abstract in the declaration of both signatures indicates that new signatures can *extend* them, inheriting all their characteristics. In our example, we can for instance, create concrete robots and capabilities (e.g., r1, r2 and c1, c2) that instantiate their corresponding abstract signatures,

```
sig r1, r2 extends Robot { ... }
sig c1, c2 extends Capability { ... }
```

Alloy models can incorporate relational logic sentences in the form of predicates and *facts* (i.e., expressions that always have to be satisfied) that place explicit constraints on the model. Hence, when the Alloy analyzer searches for examples that satisfy the structural properties described in the Alloy model, it discards any which violate any fact. In our example, we can write a fact which ensures that if a capability c belongs to a specific robot r, that robot indeed has c among its capabilities (and the other way around),

```
fact { all c:Capability, r:Robot |
r in c.belongsto <=> c in r.hascapability }
```

The set of integers must be explicitly declared if the Alloy model constrains any reference to this set. It is added into the model by the statement,

```
open util/integer
```

The Alloy analyzer can look for examples of structures that satisfy all the relational constraints in the model within a finite *scope*. All Alloy models are bounded models; i.e., they require a maximum size. These can be explicitly defined stating the maximum number of atoms of each signature that a solution can consider. If not specified, the analyzer will compute up to three of each signature and any number of relations. The scope can be modified within the command **run**, which starts the analysis of the Alloy model. For example,

```
run predicate1 for exactly 6 Capability, 2 Robot
```

generates a search space for the solutions with exactly 6 atoms of type Capability and up to 2 robots.

A special set is the **set of integers**. Its scope is added into the **run** statement as the bitwidth of the range of values needed for the model; for an integer scope n, Alloy generates 2^n values ranging from $-2^n/2$ to $(-2^n/2) - 1$. For example, the command

```
run predicate1 for 6 Int
```

sets n to 6, and therefore adds into the model the set of integers ranging from -32 to 31.

Alloy Analyzer. Alloy analyzer provides tool support for the specification and analysis of Alloy models. Developed by the Software Design Group at MIT, the tool uses Kodkod [56] as its model finding engine, and can be configured to employ one of several SAT solvers for its constraint solving. The SAT4J SAT solver is selected by default. For faster performance on small problems, MiniSat and ZChaff solvers are available (for some operating systems) [57]. For more information, we direct the reader to [58] and the Alloy website [59].

Alloy for MRS. The Alloy Analyzer is a powerful tool for exploring and checking models of systems defined by relational logic. In the context of MRS and task allocation, Alloy's ability to automatically generate instances of the system and check for constraint satisfaction is valuable. Specially, for the task allocation problem. The Alloy Analyzer can quickly explore the state space of the system and generate 'correct' allocations, which is crucial in MRS due to the complexity of interactions between multiple robots and tasks, where unexpected behaviours can easily arise.

Using Z notation in conjunction with the Alloy Analyzer combines the strengths of both formal methods. Z notation provides a rigorous mathematical framework for specifying the components and constraints of the MRS task allocation problem, ensuring that the system's design is precise and unambiguous. Once the system is specified in Z, he Alloy Analyzer can generate allocation instances and verify the specifications by exploring different scenarios and ensuring the constraints are properly applied. This combination allows for a comprehensive approach to the problem: Z notation ensures that the system is correctly and clearly defined, while the Alloy Analyzer ensures the generation of correct solutions, such as for the task allocation problem, and verifies that their deployment leads to the expected behaviour of the robots.

2.3 Verification of Probabilistic Systems

Since certain aspects of a robotic system and its environment can be described with probabilities, it is reasonable to represent this behaviour through the modelling of a stochastic process—see, for instance, Table 3.5 later in the thesis for a wide range of uncertainties encountered in robotic systems, many of which can be modelled using probability theory. In this section, we introduce several types of probabilistic models, their verification through a technique called probabilistic model checking, and a tool for their automatic verification called PRISM. As we are using the PRISM modelling language and tool later in the thesis, this section adopts the notation and definitions for the probabilistic models and probabilistic model checking from [60], and [61] for the synthesis of adversaries.

Further information about this large topic is available in [29].

Formal verification is a systematic methodology that uses mathematical reasoning to obtain guarantees about the correctness of a system. A well-known technique within this field successfully applied for robotic systems is model checking, which involves constructing and analysing a system model, typically presented as a finite state automaton characterised by states representing different configurations and transitions capturing the evolution of the system over time. The verification of the system is done under a property defined in temporal logic, such as "the robotic system eventually finishes the mission" or "no object collisions happen until the mission is completed".

Probabilistic model checking (PMC) is a type of formal verification enabling quantitative reasoning about systems, for example by considering probabilistic and timed aspects of a robotic system. It is a generalisation of model checking for the analysis of probabilistic models, such as Markov models. **Markov models** are a special category of stochastic processes which satisfy the Markov property. This property dictates that future events are independent of the past, with the present influencing only the immediate subsequent event. We describe next two widely used types of models that follow the Markov property: discrete-time Markov chains (DTMCs), used to model probabilistic behaviour; and the Markov decision processes (MDPs), an extension of DTMCs that supports the modelling of nondeterministic actions.

2.3.1 Discrete-Time Markov Chains

A discrete-time Markov chain is a mathematical model used to describe a system that undergoes transitions between a set of distinct states in discrete time steps. In a homogeneous DTMC, the transition probabilities remain constant over time. This means that the system's behaviour does not change as time progresses. The evolution of a DTMC can be described through a sequence of random variables representing the system's state at each discrete time step.

Definition 2.3.1 (Discrete-time Markov chain (DTMC)) *A DTMC is a tuple $\mathcal{D} = (S, \iota_{init}, \mathbf{P}, L)$, where S is a (countable) set of states and $\iota_{init} \in S$ the initial state; $\mathbf{P} : S \times S \to [0,1]$ is a transition probability matrix such that the exiting probability of state $s \in S$ is given by $\sum_{s' \in S} \mathbf{P}(s, s') = 1$; and $L : S \to 2^{AP}$ is a labelling function assigning a set of atomic propositions from a set AP to each state $s \in S$.*

In this definition, 2^{AP} denotes the power set of AP. An atomic proposition defines a relevant characteristic of the robotic system. This describes simple Boolean statements such as "the robot r1 is at location at1", evaluating to true in states where the premise holds, i.e., if robot r1 is at this given location. Another example of an atomic proposition is "the mission has been completed", evaluating to true in states where all the tasks assigned to the robots have been successfully completed.

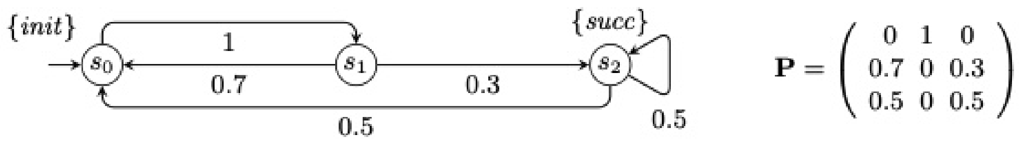

{init}
{succ}
1
0.7
0.3
0.5
0.5
$\mathbf{P} = \begin{pmatrix} 0 & 1 & 0 \\ 0.7 & 0 & 0.3 \\ 0.5 & 0 & 0.5 \end{pmatrix}$

Figure 2.1: Representation of a DTMC and its transition matrix, **P**. Taken from [60].

Figure 2.1 shows a DTMC consisting of three states $S = \{s_0, s_1, s_2\}$, with initial state $\iota_{init} = s_0$, $L(s_0) = \{init\}$, $L(s_1) = \{\}$, $L(s_2) = \{succ\}$, and transition probability displayed on each state transition (arrows). The transition probabilities are also shown as the **P** matrix on the right. This matrix contains the transition probabilities for all state pairs, where the row number gives the initial state and the column number gives the final state of the transition (starting at 0).

A path (finite or infinite) represents one possible execution of $\mathcal{D}$, i.e., a sequence of states $s_0, s_1, s_2, ...$ for which the probability of transitioning between consecutive states is greater than 0, $\mathbf{P}(s_i, s_{i+1}) > 0$. We denote the set of all finite and infinite paths from state s as $FPath_{\mathcal{D},s}$ and $IPath_{\mathcal{D},s}$, respectively. We use $\rho \in FPath_{\mathcal{D},s}\pi$ to denote a finite path and $\pi \in IPath_{\mathcal{D},s}$ to represent an infinite path.

We can **reason about DTMCs** by determining the probability that a certain path is taken. We define the probability of a finite path $\rho = s_0...s_n$ as $\mathbf{P}(\rho) ::= \Pi_{i=0}^{n-1}\mathbf{P}(s_i, s_{i+1})$. For infinite paths, we begin by finding the *basic cylinder* C_p for each finite path $\rho \in FPath_{\mathcal{D},s}$, consisting of all infinite paths starting with ρ. Using properties of cylinders, a probability space $(IPath_{\mathcal{D},s}, \mathcal{F}_{\mathcal{D},s}, Pr_{\mathcal{D},s})$ is constructed; where $\mathcal{F}_{\mathcal{D},s}$ is the smallest *sigma*-algebra generated by the set of basic cylinders $\{C_\rho \mid \rho \in FPath_{\mathcal{D},s}\}$ and $Pr_{\mathcal{D},s}$ is a measure such that $Pr_{\mathcal{D},s}(C_\rho) = \mathbf{P}(\rho), \forall\, \rho \in FPath_{\mathcal{D},s}$.

For example in Figure 2.1, we can compute the probability that a path contains s_2 as,

$$\begin{aligned} Pr_{\mathcal{D},s_0}(\{\pi \text{ contains } s_2\}) &= \Sigma_{n=1}^{\infty} Pr_{\mathcal{D},s}(\{\pi \text{ starts with } (s0s1)^n s2\}) \\ &= \Sigma_{n=1}^{\infty} 1 \times (0.7 \times 1)^{n-1} \times 0.3 = 1 \end{aligned}$$

We can rephrase the last query as, "what is the probability that in the future s_2 happens?", as described later in Section 2.3.3.

2.3.2 Markov Decision Processes

Markov decision processes (MDPs) are a formalism for the modelling of systems that supports probabilistic and nondeterministic behaviour to be captured in the model [60]. MDPs are state-transition models where transitions are made in two stages. Given a state in an MDP, first, an action is taken from a set of actions available. Then, a (probabilistic) transition is made to a possible next state following a probability distribution that depends on the action selected in the first stage [60].

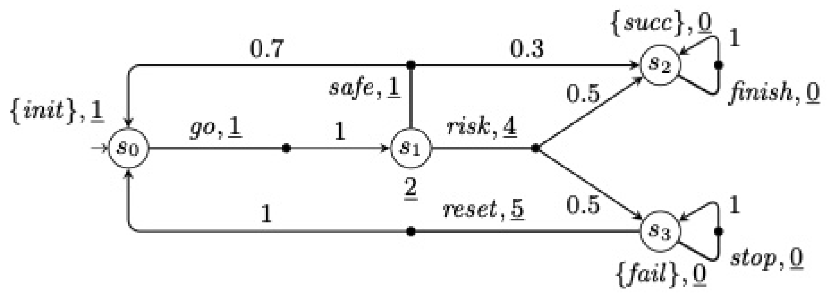


Figure 2.2: Representation of an MDP with rewards (underlined numbers). Taken from [60].

Definition 2.3.2 (Markov decision process (MDP)) *An MDP is a tuple $\mathcal{M} = (S, \iota_{init}, \alpha_{\mathcal{M}}, \delta_{\mathcal{M}}, L)$, where S is a finite set of states and $\iota_{init} \in S$ the initial state; $\alpha_{\mathcal{M}}$ is a set of actions; $\delta_{\mathcal{M}} : S \times \alpha_{\mathcal{M}} \to Dist(S)$ is a (partial) probabilistic function that maps state-action pairs to distribution functions over the states; and $L : S \to 2^{AP}$ is a labelling function assigning a set of atomic propositions from a set AP to each state $s \in S$.*

In this definition, $Dist(S)$ represents the set of distributions $\mu \in Dist(S)$ over S, where, $\mu = \{s_0 \mapsto x_0, ..., s_n \mapsto x_n\}$ denotes the (discrete) distribution that chooses s_i with probability $x_i \in [0, 1]$, where $\mu : S \to [0, 1]$ satisfying $\Sigma_{s \in S}\, \mu(s) = 1$. For each state s, we use the notation $A(s)$ for the set of actions available in state s: $A(s) ::= \{a \in \alpha_{\mathcal{M}} \mid (s, a) \in \operatorname{dom} \delta_{\mathcal{M}}\}$.

Reward structures are used to associate quantities to MDP states and/or transitions, e.g., to track the energy consumption or calculate the minimum travelling distance of a robot from an initial to a final location; hence, rewards are often referred as *costs*.

Definition 2.3.3 (Reward structure) *A reward structure over an MDP $\mathcal{M}$ is defined as a tuple $r = (r_{state}, r_{action})$, where $r_{state} : S \to \mathbb{R}_{\geq 0}$ is an* ***state reward function*** *and $r_{action} : S \times \alpha_{\mathcal{M}} \to \mathbb{R}_{\geq 0}$ is an* ***action reward function****.*

A **finite path** ρ of an MDP is a sequence of states-actions ending in a state, e.g. $\rho = s_0, a_0, s_1, a_1, ..., a_{n-1}, s_n$, where the last state is retrieved by the function $last(\rho) = s_n$ and $\mid \rho \mid = n$ denotes the length of the path. The set of all finite paths of an MDP $\mathcal{M}$ is denoted by $FPath_{\mathcal{M},s}$.

Figure 2.2 shows an MDP $\mathcal{M}$ comprising four states $S = \{s_0, s_1, s_2, s_3\}$ with initial state $\iota_{init} = s_0$ and action space $\alpha_{\mathcal{M}} = \{go, risk, safe, finish, reset, stop\}$. The probabilistic function has values depicted before a state is reached (inward arrow) from an (action, state) pair. For example, $\delta_{\mathcal{M}}(s_0, go) = [s1 \mapsto 1]$, $\delta_{\mathcal{M}}(s_1, risk) = [s2 \mapsto 0.5, s3 \mapsto 0.5]$. We note that the MDP actions are only available in specific states, e.g., $(s_1, go) \notin \operatorname{dom} \delta$ and $A(s_1) = \{risk, safe\}$. The labels of states are depicted in curly brackets, for example, $L(s_0) = \{init\}$ and $L(s_1) = \varnothing$. State and action rewards are also depicted in Figure 2.2

by underlined numbers close to the states labels and actions; for instance $r_{state}(s_1) = 2$ and $r_{action}(s3, reset) = 5$. A finite path of MDP $\mathcal{M}$ is $\rho = s_0 \xrightarrow{go} s_1 \xrightarrow{risk} s_2$, and an infinite path $\pi = s_0 \xrightarrow{go} s_1 \xrightarrow{risk} s_2 \xrightarrow{finish} s_2 \xrightarrow{finish} \ldots$.

Definition 2.3.4 (Deterministic MDP) *We use the term "deterministic MDP" to describe an MDP in which transitions for every state-action pair are deterministic.*

Adversaries. Quantitative information can be retrieved from an MDP model in the form of rewards and probabilities [60]. To reason about probabilities in the same way as for DTMCs, it is necessary to remove the non-determinism so that a fully probabilistic model is left.[1] For this purpose, *adversaries* are responsible for choosing an action (i.e., a distribution) at each of the states of the MDP, based on the execution so far represented as a finite path ρ.

Definition 2.3.5 (Adversary) *An adversary of an MDP $\mathcal{M}$ is a function $\sigma : FPath_{\mathcal{M}} \to Dist(\alpha_{\mathcal{M}})$ where $\sigma(\rho)(a) > 0$ only if $a \in A(last(\rho))$, with $last(\rho)$ denoting the last state on path ρ.*

For the example in Figure 2.2, we can define an adversary that solves the non-determinism after an execution $s0 \xrightarrow{go} s1$, by selecting action *safe* with probability 1,

$$\sigma(s0 \xrightarrow{go} s1) = [safe \mapsto 1]$$

which satisfies $\sigma(s0 \xrightarrow{go} s1)(safe) > 0$ as $safe \in A(s1)$.

An adversary is said to be *memoryless* if $\sigma(\rho)$ depends only on the last state $last(\rho)$ rather than on a path leading to this state, i.e., for any two paths $\rho, \rho' \in FPath_{\mathcal{M}}$ such that $last(\rho) = last(\rho')$, then $\sigma(\rho) = \sigma(\rho')$. Additionally, an adversary σ is said to be *deterministic* if $\sigma(\rho)$ is a point distribution $\forall\, \rho \in FPath_{\mathcal{M}}$, and *randomised* otherwise.

In autonomous systems and MRS applications, probability is used to quantify environmental uncertainty and stochasticity, while non-determinism represents model decisions [62]. Non-determinism allows for the representation of choices made by a controller of a robot, in our case, the choices of idling a robot or travel and perform a task. Solving the non-determinism by generating deterministic (or *pure)* adversaries means that the robot's controller picks a single action (with probability 1) for all finite paths of the MDP. Hence, we use deterministic adversaries to solve when the robot must idle (to synchronise in time with other robots) while completing its allocated tasks.

In the context of general path formulae, optimal strategies are finite-memory and deterministic because they must account for a variety of states and decisions throughout a process. Finite memory allows these strategies to retain information about previous

[1]For any adversary σ, this results in an *induced DTMC* denoted as $\mathcal{M}_{\sigma}$.

states, which is crucial for making informed decisions based on the history of the system's behavior.

However, for specific scenarios, such as calculating probabilities or expected accumulated rewards to *reach* a target (used in this thesis for the synthesis of robot plans), memoryless deterministic adversaries suffice [62]. This is because the decision at any given moment can be made based solely on the current state, without needing to consider past states or actions. In these cases, the outcomes are typically based on well-defined probabilities or rewards, allowing for straightforward, immediate choices that do not require historical context [60].

For these reasons, we focused exclusively on *deterministic memoryless* adversaries, which are sufficient for addressing the MRS problem at hand. A way to formalise specific properties of an MDP for which we wish to synthesise an adversary is using temporal logic, as detailed in the following section.

2.3.3 The Logics PCTL and RPCTL

Probabilistic computational tree logic augmented with rewards (RPCTL) is an extension of probabilistic computational tree logic (PCTL). Hence, in this section, we describe PCTL and then RPCTL.

PCTL. Probabilistic computational tree logic [63] is an extension of computational tree logic (CTL) augmented with probabilities.

Definition 2.3.6 (PCTL Syntax) *The syntax of PCTL is defined by,*

$$\Phi ::= \mathit{True} \;\mid\; ap \;\mid\; \neg\Phi \;\mid\; \Phi_1 \wedge \Phi_2 \;\mid\; \mathbf{P}_{\bowtie p}(\Psi) \tag{2.1}$$

$$\Psi ::= \boldsymbol{X}\Phi \;\mid\; \Phi_1\, U \Phi_2 \;\mid\; \Phi_1\, U^{\leqslant n}\Phi_2 \tag{2.2}$$

where Φ is a state formula and Ψ a path formula; $ap \in AP$ is an atomic proposition and AP the set of atomic propositions; $p \in [0,1]$ with $\bowtie \in \{\geqslant, \leqslant, >, <\}$; and $n \in \mathbb{N}$.

A PCTL formula can be contextualised by a state transition model such as an MDP, which has states labelled with atomic propositions. Hence, atomic propositions $ap \in AP$ match propositions existing in an MDP. A *property* of a model is defined as a state formula; meanwhile, path formulae only occur within the probabilistic operator $\mathbf{P}_{\bowtie p}(\varphi)$ to reason about the possibility (true or false) that any path satisfying φ from an initial given state(s) s has a probability, for example, greater or equal than ($\bowtie = \geqslant$) some predefined value p.

For a path ρ, the *next* path formula $X\Phi$ is true if Φ holds for the next state of ρ; the *until* operator U requires two state formulae ($\Phi_1 U \Phi_2$) to assess if Φ_1 is true in all states along the path until Φ_2 becomes true; similarly, the *bounded until* operator ($\Phi_1 U^{\leqslant n}\Phi_2$) constraints Φ_2 to become true within the next n states, with Φ_1 holding until then.

The semantics of the PCTL logic are described in terms of the states and paths, in our case, of the MDP state transition model and its adversaries (Adv).

Definition 2.3.7 (PCTL Semantics) *Given an MDP $\mathcal{M} = (S, \iota_{init}, \alpha_{\mathcal{M}}, \delta_{\mathcal{M}}, L)$, a class of adversaries Adv of $\mathcal{M}$ with $\sigma \in Adv$, a state $s \in S$, PCTL state formulae $\{\Phi_1, \Phi_2\}$, and path formula Ψ, the satisfaction relation $\models_{Adv}$is defined by:*

$$s \models_{Adv} \mathit{True} \quad \mathit{always\ holds} \tag{2.3}$$

$$s \models_{Adv} ap \quad \mathit{iff} \quad ap \in L(s), \tag{2.4}$$

$$s \models_{Adv} \neg\Phi \quad \mathit{iff\ not} \quad s \models_{Adv} \Phi, \tag{2.5}$$

$$s \models_{Adv} \Phi_1 \wedge \Phi_2 \quad \mathit{iff} \quad s \models_{Adv} \Phi_1 \mathit{\ and\ } s \models_{Adv} \Phi_2, \tag{2.6}$$

$$s \models_{Adv} \mathbf{P}_{\bowtie p}(\Psi) \quad \mathit{iff} \quad Pr^{\sigma}_{(\mathcal{M},s)}(\{\pi \in IPath_{(\mathcal{M},s)} \mid \ \pi \models_{Adv} \Psi\}) \bowtie p \ , \ \forall \sigma \in Adv. \tag{2.7}$$

where Pr is the probability operator and $IPath_{(\mathcal{M},s)}$ is the set of all infinite paths starting from s. Then, $\forall \pi \in IPath_{\mathcal{M}}$:

$$\pi \models_{Adv} X\Phi \quad \mathit{iff} \quad \pi(1) \models_{Adv} \Phi, \tag{2.8}$$

$$\pi \models_{Adv} \Phi_1 U \Phi_2 \quad \mathit{iff} \quad \exists\, n \geqslant 0 \cdot (\pi \models_{Adv} \Phi_1 U^{\leqslant n} \Phi_2), \tag{2.9}$$

$$\pi \models_{Adv} \Phi_1 U^{\leqslant n} \Phi_2 \quad \mathit{iff} \quad \exists\, i \leqslant n \cdot (\pi(i) \models_{Adv} \Phi_2 \wedge (\forall j < i . \pi(j) \models_{Adv} \Phi_1)). \tag{2.10}$$

Min/max probability. The second part of Formula 2.7 is shorten as follows as it is repeatedly used in the next sections,

$$\mathbf{P}^{\sigma}(\Psi) \coloneqq Pr^{\sigma}_{(\mathcal{M},s)}(\{\pi \in IPath_{(\mathcal{M},s)} \mid \ \pi \models_{Adv} \Psi\}) \tag{2.11}$$

Using this notation, we define two types of adversaries $\sigma \in \{\mathtt{min}, \mathtt{max}\}$ corresponding to the minimum and maximum probabilities of a formula Φ holding under the adversaries *Adv* from state s, respectively. For example, in a robotic system, we can reason about "what is the minimum probability of completing all tasks assigned to robots without failure".

$$\mathbf{P}^{\mathtt{min}}(\Psi) \coloneqq \mathtt{inf}_{\sigma \in Adv} Pr^{\sigma}(\Psi) \tag{2.12}$$

$$\mathbf{P}^{\mathtt{max}}(\Psi) \coloneqq \mathtt{sup}_{\sigma \in Adv} Pr^{\sigma}(\Psi) \tag{2.13}$$

where `inf` and `sup` refers to the inferior and superior values, respectively.

Derived operators. The following additional operators are derived from operators in the PCTL syntax:

$$\begin{aligned}
\mathit{False} &:= \neg\, \mathit{True},\\
\Phi_1 \vee \Phi_2 &:= \neg(\neg\Phi_1 \wedge \neg\Phi_2),\\
\Phi_1 \rightarrow \Phi_2 &:= \neg\Phi_1 \vee \Phi_2,\\
\mathtt{F}\Phi &:= \mathit{True}\ \ U\ \ \Phi,\\
\mathtt{F}^{\leqslant n}\Phi &:= \mathit{True}\ \ U^{\leqslant n}\Phi,\\
G\Phi &:= \neg(F\neg\Phi),\\
G^{\leqslant n}\Phi &:= \neg(F^{\leqslant n}\neg\Phi).
\end{aligned}$$

where $\vee$ and $\rightarrow$ are the logic disjoint and implication operators. The *future* or *eventually* operator F (also found in the literature as $\diamond$) intuitively looks for a future state on a path (from a given initial state) that satisfies state formula Φ. Similarly, the *always* or *global* operator G (also written as $\square$) looks for a path where all states satisfy the state formula Φ. F and G operators are borrowed from CTL, and are augmented with bounded variants ($F^{\leqslant n}$, $G^{\leqslant n}$) limiting the number of states ahead for which the formula has to hold to be *True*.

Note that the future operator (F) is implicit within the definition of PCTL semantics as it is derived from the until operator U. However, the global operator (G) requires the negation of a formula that is not part of PCTL. Hence, we define the following equality and direct the reader to [60] for the explanation of equivalence,

$$\mathbf{P}_{\leqslant 1-p}[F\ \neg\Phi] := \mathbf{P}_{\geqslant p}[G\ \Phi] \tag{2.14}$$

PCTL with Rewards. A second extension of the initial PCTL is the reward-related operator R. The complete syntax of **RPCTL** is given by,

$$\Phi ::= \mathit{True}\ \mid\ ap\ \mid\ \neg\Phi\ \mid\ \Phi_1 \wedge \Phi_2\ \mid\ \mathbf{P}_{\bowtie p}(\Psi)\ \mid\ \mathbf{R}^{r}_{\bowtie\, x}[\mathbf{I}^{=n}]\ \mid\ \mathbf{R}^{r}_{\bowtie\, x}[\mathbf{C}^{\leqslant n}]\ \mid\ \mathbf{R}^{r}_{\bowtie\, x}[\mathbf{F}\ \Phi] \tag{2.15}$$

where $r = (r_{state}, r_{action})$ is an MDP reward structure (see Section 2.3.2) that allows the specification of a reward, i.e. a quantity, that changes through changes of transitions and states in the Markov model; and $x \in \mathbb{R}_{\geqslant\ 0}$ is a reward bounding variable. The semantics of RPCTL follows the same semantics as defined for PCTL plus the following formulae.

Three random variables are introduced to specify different types of *expected* rewards, defined overall infinite paths $\pi \in \mathit{IPath}_s$. First, instantaneous reward returns the reward gained immediately *at* the n time step, and is defined as $I^{n}_{r_{state}} := r_{state}(\pi(n))$. The step-bounded cumulative reward computes the reward accumulated *up to* the nth time step, defined as $C^{\leqslant n}_{r}(\pi) = \Sigma_{i=0}^{n-1}(r_{state}(s_i)) + r_{action}(s_i, a_i))$.

The third one is the cumulative reward to reach a *target* where Φ holds, also known as reachability reward. To explain it, we first define a set of target states $T \in S$ which contains all states that satisfy the formula Φ. We now can define F_r as,

$$F_r^T(\pi) = \begin{cases} \infty, & \text{if } s_i \notin T \ \forall\, i \in \mathbb{N} \\ \Sigma_{i=0}^{k_\pi^T - 1}(r_{state}(s_i)) + r_{action}(s_i, a_i)), & \text{otherwise} \end{cases}$$

where $k_\pi^T = \mathtt{min}\{k \mid s_k \in T\}$ is the lower state number that is in the target set T.

Reachability rewards are particularly relevant to MRS missions to reason about energy consumption until the mission is completed, the time elapsed when after some locations are reached, or to track the expected number of failed tasks [64]. Reachability properties are also common in the specification of robotic missions, for example, to specify that a group of robots must visit locations $l1$ and $l2$ (defined in LTL as $F\ (l1 \wedge l2)$) [2,20,64]. To limit the scope of this thesis, we only use as part of this thesis reachability rewards for the synthesis of MDP adversaries, to specify quantitative properties such as: *what is the minimum time that a group of robots need to spend idling (due to waiting for other robots to arrive at a task location) while completing a mission.*

RCPTL adversary semantics. The following RPCTL semantic rules are additional to the PCTL semantics defined in (2.3) to (2.10). These define the semantics of adversaries Adv instead of infinite paths $IPath_s$,

$$s \models_{Adv} \mathbf{R}^r_{\bowtie\ x}[\mathbf{I}^{=n}] \ \text{ iff } \ \mathbb{E}^{\sigma}_{(\mathcal{M},s)}(\mathbf{I}^{=n}_{r_{state}}) \bowtie x \qquad \forall\, \sigma \in Adv, \tag{2.16}$$

$$s \models_{Adv} \mathbf{R}^r_{\bowtie\ x}[\mathbf{C}^{\leqslant n}] \ \text{ iff } \ \mathbb{E}^{\sigma}_{(\mathcal{M},s)}(C_r^{\leqslant n}) \bowtie x \qquad \forall\, \sigma \in Adv, \tag{2.17}$$

$$s \models_{Adv} \mathbf{R}^r_{\bowtie\ x}[\mathbf{F}\Phi] \ \text{ iff } \ \mathbb{E}^{\sigma}_{(\mathcal{M},s)}(F_r^{Sat_{Adv}(\Phi)}) \bowtie x \qquad \forall\, \sigma \in Adv, \tag{2.18}$$

where $\mathbb{E}$ is the expected value under adversary σ.

Intuitively, these formulae answer the question of where a state s has a reward bounded by x under some adversary σ.

Min/max rewards. The value of interest for the rewards formulae is either the minimum or maximum reward. This can be written similar to the quantitative probability formulae, as, $\mathbf{R}^{\mathtt{min},r}[\cdot]$ and $\mathbf{R}^{\mathtt{max},r}[\cdot]$.

2.3.4 Probabilistic Model Checking

The basic theory of probabilistic model checking was developed in the mid-80s [65]. Later on, Alur, Courcoubetis and Dill worked on the timed automata model for real-time and probabilistic systems [66,67]. Continuing, in the mid-90s, [63] and [68] introduced PCTL and PCTL* for probabilistic specifications. Daws [69] provides the following description of probabilistic model checking:

> *Probabilistic model-checking of PCTL over discrete probabilistic systems is based on the computation in every state of the probability measure of the set*

of paths satisfying a (path) formula. These probabilities are computed numerically by solving a system of linear equations in the case of DTMCs [...], and by solving a linear optimisation problem in the case of MDPs [...].

Qualitative properties are concerned with a yes or no answer to the question: "Can this assertion ever be violated?" [70]. On the other hand, **quantitative** properties check for *dependability* and *performance* aspects, such as [70]:

- **Dependability** aspects: safety, reliability, availability, survivability, etc.
- **Performance** aspects: response times, throughput, power consumption, etc.

Quantitative verification requires a *model formalism* that captures real-time aspects, probabilities or continuous dynamics [70]. Examples of these models are DTMCs and MDPs.

As Markov models suffer from the state-explosion problem, i.e., an explosion in the number of states as the complexity of the modelled system grows, probabilistic model checking techniques are usually combined with mechanisms to reduce the number of states to visit. We refer the reader to [29] for detailed descriptions of the algorithms used for probabilistic model checking, as well as strategies for state-space reduction. We also recommend the paper [60] on the formalisation of the techniques implemented for the probabilistic model checker PRISM (see Section 2.3.5), which is used in this thesis.

2.3.5 PRISM Model Checker

PRISM modelling language. KANOA MDPs are defined in the high-level modelling language of the probabilistic model checker PRISM [71]. A model specified in this language comprises several interacting reactive modules. Each such module consists of a set of finite-valued state variables, and several transitions (i.e., commands) modifying these variables. A *transition* has the generic form

[<action>] <guard> → <prob>:<update> +...+ <prob>:<update>;

and comprises an optional *action* label, a guard and state updates with transition probabilities assigned. An *action* is a label that allows synchronisation between modules. A *guard* is a predicate over the variables of all modules. An *update* modifies the module variables if the transition is taken (→) with a probability of *prob*. In an MDP, nondeterministic actions are taken when two transitions in the same or different modules have overlapping guards.

PRISM Model Checker [72] is a probabilistic model checker that supports multiple types of probabilistic models, including DTMCs and MDPs, as well as several types of probabilistic logics, including PCTL and RPCTL. These models and properties must be defined in the PRISM modelling language to be automatically analysed by the PRISM Model Checker. We refer the reader to the PRISM website manual [73] for further detail.

2.4 Evolutionary-guided Synthesis of MDP Adversaries

The MDP model is defined in Section 2.3.2. In this section, we define the multi-objective optimisation of synthesizing MDP adversaries using meta-heuristics. We are interested in optimising a set of parameters related to the quality of the mission. The multi-objective problem (MOP) is formally defined as [74],

Definition 2.4.1 (MOP) *A multi-objective problem (MOP) requires finding a vector* $\overline{x}^{+} = [x_1^{+}, x_2^{+}, \ldots, x_n^{+}]$ *which satisfies: p equality constraints* $h_i(\overline{x}) = 0,\ i = 1, 2, \ldots, p,$ *and m inequality constraints* $g_i(\overline{x}) \geq 0$, *for* $i = 1, 2, \ldots, m$; *and minimizes the vector of functions* $\overline{f}\overline{x} = [f_1(\overline{x}), f_2(\overline{x}), \ldots, f_k(\overline{x})]^T$, *where* $\overline{x} = [x_1, x_2, \ldots, x_n]^T$ *is the vector of decision variables.*

The set of all feasible solutions is referred to as the feasible region Ω, with a feasible solution being $\overline{x} \in \Omega$. Note that while this definition refers to *minimising* functions $\overline{f}_i$, this is equivalent to *maximising* these functions multiplied by -1.

Pareto optimality. Finding a Pareto-optimal solution from the feasible region requires finding a solution such that no other feasible solution exists that would improve some (objective) functions $\overline{f}_i$ without worsening one or more other such functions [74].

Definition 2.4.2 (Pareto optimal point) *A point* $\overline{x}^{+}$ *is Pareto optimal if for every other solution* $\overline{x} \in \Omega$, *either* $\forall\, i \in \{1, ..., k\}.f_i(\overline{x}^{+}) = f_i(\overline{x})$, *or* $\exists\, i \in \{1, ..., k\}.f_i(\overline{x}) > f_i(\overline{x}^{+})$.

Obtaining the Pareto front of a MOP, composed of all non-dominated solutions and their respective vector of function values, is the main aim of the multi-objective optimisation problem [74].

Definition 2.4.3 (Dominated solution) *A vector* $\overline{v}$ *is said to be dominated by* $\overline{u}$ *iff* $\forall\, i \in \{1, ..., k\}.u_i \leq v_i$ *and* $\exists\, i \in \{1, ..., k\}.u_i < v_i$. *We use the notation* $\overline{u} \preccurlyeq \overline{v}$ *to denote this dominance.*

We are interested in finding the set of solutions that are Pareto optimal.

Definition 2.4.4 (Pareto optimal set) *The Pareto optimal set for a MOP* $\overline{f}(\overline{x})$ *is the set as* $\mathcal{P}^{+} = \{\overline{x} \in \Omega \mid \neg \exists\, \overline{x}' \in \Omega, \overline{f}(\overline{x}') \preccurlyeq \overline{f}(\overline{x})\}$.

Definition 2.4.5 (Pareto front) *The Pareto front of a MOP* $\overline{f}(\overline{x})$ *of a Pareto optimal set* $\mathcal{P}^{+}$ *is the set* $\mathcal{PF}^{+} = \{\overline{f}(\overline{x}), \overline{x} \in \mathcal{P}^{+}\}$.

Genetic algorithms. Metaheuristic algorithms are used to solve this optimisation problem when the Pareto front consists of a large number of points. *Genetic algorithms* (GA) is a method for solving optimisation problems. It encodes a large number of solutions (population), where the individual solution (chromosome) is a sequence of values, called *genes*, that are genetically bred through Darwinian principles of survival and reproduction of the fittest solutions, and their recombination (crossover) to create new populations (offspring) that *evolve* towards an optimal solution [75]. GA starts with an initial randomly generated population of feasible solutions (individuals) from a design space $\mathcal{DS}$. This population is evolved into subsequent populations by the GA stages of *selection, crossover* and *mutation*. The selection stage uses a *fitness function* to evaluate each individual and selects the population for the next generation, as well as the *mating pool* (individuals with the highest fitness levels). Crossover selects two individuals from the mating pool and generates new individuals by recombination of their genes. Lastly, mutation produces individuals by randomly modifying some genes from individuals in the pool. This process continues by replacing the initial population with the new population until there is no significant fitness improvement or after a predetermined number of iterations [50]. The most used multi-objective evolutionary algorithm (MOEAs) is the nondominated sorting genetic algorithm II (NSGA-II), used also in this paper. We refer the reader to [76] for further details.

EvoChecker. EvoChecker [50, 77] is a search-based approach and tool for the synthesis of feasible instantiations for parametric Markov models. A parametric Markov model is a Markov model that includes undefined variables, i.e., parameters. EvoChecker allows the use of such parameters to describe transition probabilities between states, system variables, as well as different configurations of the modelled system. EvoChecker uses as input a single model that defines the parameter ranges, as well as these possible system configurations written in an extended version of the PRISM modelling language (see Section 2.3.5). Internally, it creates PRISM models with instances of these variables within the specified ranges and configurations, and uses metaheuristic search techniques to obtain Pareto-optimal combinations of variable values.

The theory behind EvoChecker is used in this thesis for the synthesis of individual robot plans. The main difference to our approach is that we do not create a single initial model. Given that the system configurations are, in our case, the permutation of robot tasks, the number of permutations grows factorially with the number of tasks and linearly with the number of robots, making the definition of a single initial model unfeasible. For example, for 5 tasks and two robots, there are $5!\times 2 = 240$ configurations that had to be modelled as a single EvoChecker model. Hence, the approach proposed in the thesis creates each PRISM model directly from the robot system specifications. Similar to EvoChecker, EvoPoli [78] is another tool available for the evolutionary synthesis of adversaries for MDP models. However, this tool is not considered in this thesis as it is known to have scalability issues with large MDPs.

Context. In the following chapters, we use the Z notation for the formalisation of task allocation and scheduling problem specifications (Chapter 4). The Alloy Analyzer is used to generate task allocations for heterogeneous groups of mobile robots operating under a finite set of spatial constraints. The satisfaction problem is formalised with the help of the Alloy language (Chapter 5). Finally, the verification of probabilistic systems is required for the synthesis of formally verified robot plans, while searching through the solution space of these robot plans using evolutionary guided synthesis. In our formulation, a robot plan is defined by an MDP adversary (see Chapter 6).

Chapter 3
Systematic Review of Multi-robot Task Allocation and Scheduling

Multi-robot systems can carry out useful missions in applications dangerous to humans, such as search and rescue operations during natural disasters, the exploration of other planets, and infrastructure monitoring and repair in nuclear power plants. They are also capable of performing repetitive and big-scale activities, such as the delivery of goods in an urban area, and hospital cleaning and disinfection. Given the benefits of these applications, the decades of significant advances in robotics and control theoretic approaches for automating the physical motion of robots, and the growing body of research on the specification of robot tasks, one may wonder why such uses of multi-robot systems are still far from common.

The systematic review presented in this chapter highlights some of the complexities involved in adopting multi-robot systems. One significant challenge is identifying a suitable language for specifying the multi-robot problem at hand—this language must effectively capture all required elements such as tasks, task constraints, task requirements, world model characteristics, and robot capabilities. Additionally, the lack of consensus on terminology for specific problems like task allocation, scheduling, planning, and motion planning creates further difficulties. Developers must establish and formalise consistent terminology for their solutions to ensure clarity and facilitate adoption by others. Finally, given the extensive body of knowledge available for addressing multi-robot problems, selecting an appropriate solution is not straightforward, and trade-offs between various approaches must be carefully considered. We drew these conclusions following a thorough review of the research literature in which we systematically collected 837 primary research papers related to task specification, allocation and scheduling in robot and multi-robot systems spanning the past five years. After examination, 73 papers were used for the systematic review to answer research questions aimed at understanding: (a) the current terminology used among researchers; (b) how tasks are formally defined and what normative is considered; (c) the types of solutions for the task allocation and scheduling problems, what uncertainties they consider, and which types of formal methods they employ; (d) how these problems are solved and what metrics and simulators or testbeds were used to evaluate their effectiveness.

In addition to answering these research questions, we leverage our assessment of the terms used by the current research on robotic task allocation and scheduling to propose a unified terminology for this important research area.

3.1 Research Questions

Our systematic review follows the guidelines in [79] and [80][2] for systematic reviews in software engineering. As detailed in Section 3.2, we scanned 837 primary studies gathered from multiple leading venues, digital sources and experts in the specified MRS area. From these, 73 relevant studies were retained and used to answer the following exploratory research questions (EQs), which we organised into four problem categories:

EQ1. Terminology for task allocation and scheduling in robot systems.

1.a) Do the terms *mission*, *task*, *allocation* and *scheduling* refer to the same concepts across research papers?

1.b) If the terms from 1.a) refer to different concepts in different papers, what is their most common meaning?

EQ2. Ways of modelling tasks.

2.a) What types of formalisms are used for the specification of tasks?

2.b) What standards/patterns/guidelines are used to define tasks in robot and multi-robot systems?

EQ3. Allocation and scheduling problem solutions and variants.

3.a) What approaches are used to solve the task allocation and scheduling problems in robot systems?

3.b) Are task allocation, scheduling, planning or a combination thereof considered?

3.c) What research papers consider uncertainty, and what types of uncertainty are being considered?

3.d) What solutions use formal methods, and what type of formal techniques do they use? (This question is particularly relevant to our thesis, which focuses on the application of formal methods to MRS task allocation and scheduling.)

EQ4. Future research directions.

4.a) What gaps that need to be addressed by future research are identified in the paper?

3.2 Search Strategy

To identify the most relevant research, the **primary studies** were obtained from: (i) high-impact venues in the areas of robotics, automation and software engineering; (ii) the IEEE Computer Society Digital Library and Google Scholar, using the search engines available for the two platforms; and (iii) authors and publications known to be relevant from personal knowledge. Source (i) ensures the quality of the survey as most (if not

[2]This systematic review is also inspired by [1] and [81].

Table 3.1: Sources used to obtain the primary studies and the number of publications retrieved using specific search queries as described in the text.

i	**Venue**	**Acronym**	**Type**	**Num publ.**
i.1	International Journal of Robotics Research	IJRR	Journal	29
i.2	Transactions on Software Eng.	TSE	Journal	1
i.3	Robotics and Automation Letters	RA-L	Journal	369
i.4	Transactions on Robotics	T-RO (Trob)	Journal	63
i.5	Transactions on Automation Science and Eng.	T-ASE	Journal	0
i.6	Software Eng. for Adaptive and Self-Managing Sys.	SEAMS	Conf. & Symposium	0
i.7	Symposium on Applied Computing	SAC	Conf. & Symposium	2
i.8	Foundations of Software Engineering	ESEC/FSE	Conf. & Symposium	0
i.9	Intelligent Robots and Systems	IROS	Conference	133
i.10	Int. Conference on Robotics and Automation	ICRA	Conference	137
i.11	Int. Conference on Automation Science and Eng.	CASE	Conference	32
i.12	International Conference on Advanced Robotics	ICAR	Conference	15
i.13	International Conference on Software Engineering	ICSE	Conference	9
i.14	Int. Conf. on Model Driven Eng. Languages and Sys.	MODELS	Conference	3
i.15	Simulation, Modeling and Progr. for Aut. Robots	SIMPAR	Conference	2
i.16	Software Engineering and Formal Methods	SEFM	Conference	2
			TOTAL (venues)	797
ii	**Library/Search engine**			
ii.1	IEEE Computer Society Digital Library			22
ii.2	Google Scholar			16
iii	**Personal knowledge**			2
			TOTAL	**837**

all) publications from this category are peer-reviewed and obtained from high-ranked conferences. Source (ii) provides a broader view of the existing body of knowledge on MRS task allocation and scheduling; sorted by relevance, Google Scholar provides an insight into the most-cited/widely-used publications within the research community. Finally, source (iii) adds important research papers known from personal knowledge gained over the last five years.

3.2.1 Inclusion Criteria

To search through the sources, we used a query filtering papers related to *robots* (including robots, multi-robot, MRS terms) and the set of problems on which this study focuses, aka *task allocation, scheduling, mission specification* and *adaptation*. For (i), the publications were retrieved from the *DBLP* [82] repository using the query[3]:

robot|MRS$ task|schedul|allocation|mission|adapt year:2023:|year:2022:| [...] |year:2018:
streamid:journals/ijrr:|streamid:journals/tse:| [...] |streamid:conf/sefm:

For ii.2 and ii.3, the search was limited to finding the top 25 matches (which is a limit imposed by the IEEE library search engine) ranked by relevance. After an elimination

[3]Spaces mean logical *and*'s, | logical *or*'s, $ looks for exactly the word before this sign, *year*:XXXX: filters by publication year, and *streamid*:TYPE/NAME: by venue.

process to remove duplicates, 22 and 16 new papers were found in the IEEE Computer Society Digital Library and Google Scholar, respectively. The search query used to guide both engines was: (robots OR robot OR MRS) AND (task OR tasks OR schedule OR scheduling OR allocation OR mission OR missions OR adapt OR adaptation) .

Finally, another two papers were added from personal knowledge. Hence, a total of 837 **primary studies** were identified, as depicted in Table 3.1, which lists the three sources of information, as well as the full list of venues considered for source (i). The search was limited to the period 2018–2022, portraying the last five years of research. The full list of references can be downloaded from https://github.com/Gricel-lee/MRS-ThesisMaterial.

3.2.2 Exclusion Criteria

From the 837 primary studies, the following exclusion criteria were used to eliminate studies unrelated to the research questions:

1. Studies whose title and abstract are unrelated to task allocation and scheduling for robotic systems.
2. Studies focusing on the hardware of robotic systems rather than their software.
3. Studies described in short or conference papers that the authors also published in an extended journal paper (e.g., [83] and [84]).

The 146 studies retained after applying these exclusion criteria were scanned for relevance based on their conclusions, section titles and figures. After this additional examination, a final list of 73 **relevant studies** was selected to answer our exploratory questions from Section 3.1.

3.2.3 Data Items

To answer the questions formulated in Section 3.1, we used a data extraction form comprising the following entries (whose data type is provided in brackets):

- **Type** of contribution (Categorical): architecture, algorithm, specification language, mission catalogue, etc.
- **Self-adaptation** (Boolean): true if adaptation at runtime is proposed.
- **Formal methods** (Boolean and Text): the specific formal method(s) applied to address the current Multi-Robot System (MRs) problem, if applicable.
- **Summary** of the paper (Textual).
- **Terminology** (Categorical/Textual): the definition of terms that the authors use, such as task, subtask, mission, planning, etc.

- **Task specifications** (Categorical): what language is used for the specification of the tasks, e.g., a domain-specific language (DSL), Planning Domain Definition Language (PDDL), logic language, or other.
- **Task patterns** (Boolean): true if the paper describes a repository of tasks.
- **Solution** (Textual): the type of robot problem(s) studied and the solution proposed.
- **Uncertainties** considered (Text): types of uncertainties considered.
- **Robot type** (Categorical): the type of robot system used, for example, single robot, MRS, human-in-the-loop, or cobot (collaborative robot);
- **Standards** (text): references to standards and norms;
- **Gaps** (text): gaps in the state-of-the-art identified by the authors.

3.3 Results

Demographic Information

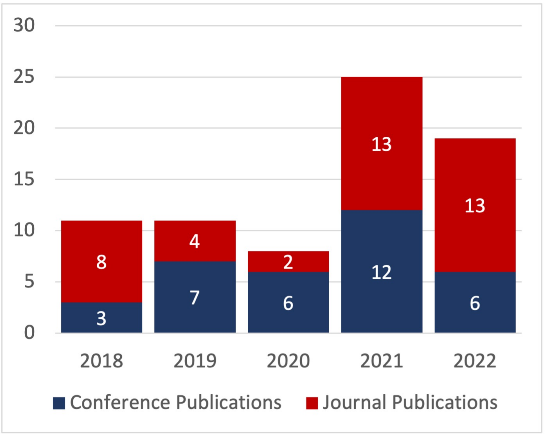


Figure 3.3: Number of publications per year for the final list of selected papers as described in the text. The lower (blue) bars are publications in Conferences and the upper (red) bars in Journals.

As shown in Figure 3.3, the number of selected robot systems, task allocation and scheduling publications from each of the last two full years has almost doubled compared to previous years, with a total of 25 publications in 2021 and 19 in 2022 compared to a maximum of only 11 papers per year between 2018 and 2020. More than half of the publications were found in the Journal Robotics and Automation Letters (RA-L) (26%) and two of its associated conferences, ICRA (16%) and IROS (9%), as depicted in Figure 3.4a. The journal with the next highest number of selected papers is IEEE Transactions on Robotics (T-RO) with 10%, followed by the conferences CASE and IJRR with 7% and 5% of the publications, respectively. From now on, we will refer simply as *papers* to the set of relevant papers.

To assess the content of the papers, we classified the 73 papers depending on the outcome of their study. This is obtained from the "**type**" data item, from the list in Section 3.2.3. We defined the following categories with the number of studies found for each category depicted within parenthesis:

1. a framework (29 papers) set of components or libraries that provide an interconnected structure, often accompanied by a defined workflow.

2. a method/approach (26 papers), a new procedure(s) or algorithm(s) to solve the MRS problem in hand, for example, the scheduling of the robot tasks;

3. an architecture (7 papers), the structure and design of a system, including the system's components and how they interact at a higher level of abstraction compared to a framework, but more structured facilitating in-further analysis (for example, to analyse system qualities such as modifiability, security, and availability);

4. a mission catalogue (4 papers), a repository of robotic missions facilitating their understanding, categorisation, and adoption;

5. a specification language (4 papers), a new language for the specification of robot tasks or robotic mission requirements;

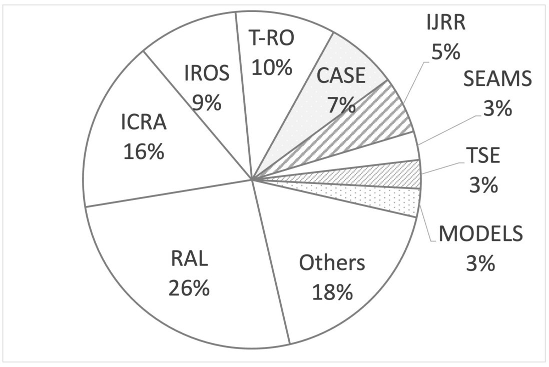
6. a case study (1 paper), the implementation and analysis of an existing solution to the MRS problem in hand within the context of a particular case study (*only the case study*, as most papers include one or more case studies as part of their evaluation);

7. a comparison study (2 papers), comparing two or more methods (*only describing the comparison*, as many papers include a comparison to other studies as part of their evaluation and/or related work sections).

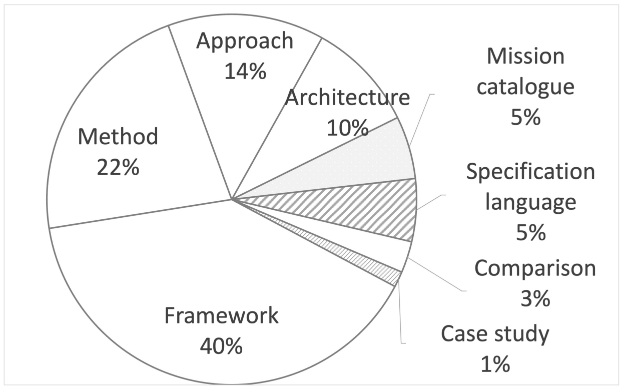
As shown in Figure 3.4b, the majority of the papers propose new frameworks, approaches, or architectures—these types of contributions account for 84% of the papers. Another 12% of the papers propose catalogues and/or new languages for the specification of missions for robotic system. One paper conducts a case study inspired by the assembly line of the Boeing 777, while the remaining two papers compare various methods for solving the task allocation and scheduling problem.

With regard to the type of robot system used by the analysed research papers obtained from the "**robot type**" data item, Figure 3.4c shows that 71% of the papers target some variant of a multi-robot system, while 18% focus on single-robot problems. Unexpectedly, 11% of the studies present research on collaborative robots (**cobots**), with a focus on human-robot interaction (**HRI**), human-robot collaboration (**HRC**), or human-in-the-loop. For additional information, refer to Table 3.2, which presents the paper citations compiled through this review categorized by the type of robotic system, (a) multiple robots (MRS), (b) single robots and (c) collaborative robots (Cobots); the latter is subdivided into three categories depending on the number of agents involved.

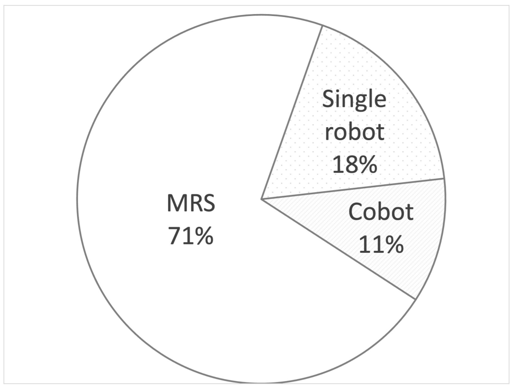
Next, we extracted the types of uncertainties considered by each paper ("**uncertainties**" data item), and assesed whether the robotic system used in the paper is capable of self-adaptation ("**self-adaptation**" data item), and/or if the paper uses formal methods ("**formal methods**" data item). We report the results as a Venn diagram in Figure 3.4d.



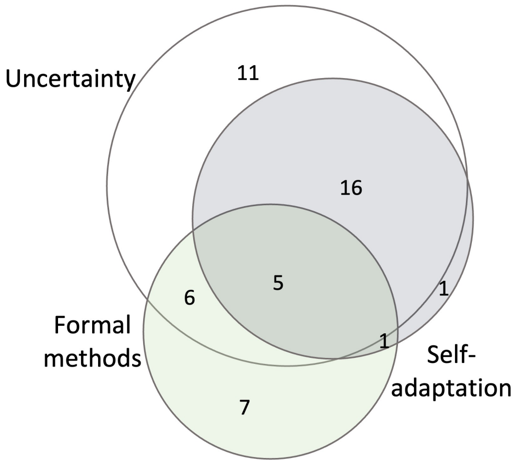
(a) Publications per venue.



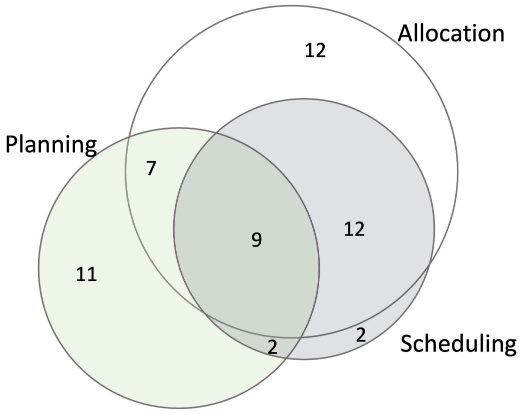
(b) Types of studies found across the 73 papers.



(c) Types of robot systems used in the studies.



(d) Studies that consider uncertainty (N), self-adaptation (SA), formal methods (FM) or a combination thereof.



(e) Studies considering task scheduling (TS), allocation (TA), planning (TP) or a combination of them.

Figure 3.4: Understanding the information contained in the collected papers.

Table 3.2: Types of robot systems and references.

Robot system type	**Count**	**References**
MRS	51	[22, 48, 49, 85–95] [20, 41, 96–108] [24, 84, 109–120] [25, 27, 28, 30, 121–124]
Single robot	14	[1, 2, 23, 36, 125–134]
Cobot (Robot-single human)	4	[47, 135–137]
Cobots (MR-single human)	1	[138]
Cobots (MR-MH)	3	[43, 139, 140]

Similarly, Figure 3.4e shows the number of papers that address the following specific problems relevant to MRS and aligned with the aims of the ongoing systematic review. To establish a clear distinction among the papers—given that some of them simultaneously address multiple issues—we have broadly outlined the following categories,

- **planning**: route/path from one location or configuration of the system to another;
- **allocation**: select who (robot, human, group of robots, etc) does what;
- **scheduling**: the ordering of the tasks, in many cases, under system constraints and objectives.

Although planning was not the target of this review, it had to be considered given its intrinsic connection to allocation and scheduling. For this Venn diagram, we did not consider all of the papers as, for example, some of them study the description of robotic missions rather than solving any of these problems. A threat to the validity of this specific Venn diagram is that, in some cases, it is difficult to assess if more than one problem is solved at the same time (for example, finding paths in the Weighted Transition System [104] may only consider planning if an initial and final point is given, but scheduling if multiple points must be visited). The information for this diagram was obtained from the "**solution**" data item.

To summarise, more than half of the selected research papers focus on frameworks and methods for MRS, while a smaller percentage target single-robot and COBOTs systems. From the 73 papers collected, 47 papers where identified to have used formal methods, self-adaptation and/or model some type of uncertainty. A total of 19 papers consider formal methods in some manner, 23 self-adaptation and 38 model some type of uncertainty. From the three robot problems targeted by the ongoing systematic review, the two mostly considered were planning and allocation accounting. The third problem, scheduling, is studied mostly in combination with the previous two. The insights derived from both Venn diagrams are expanded in response to exploratory research question EQ3.

EQ1. Terminology

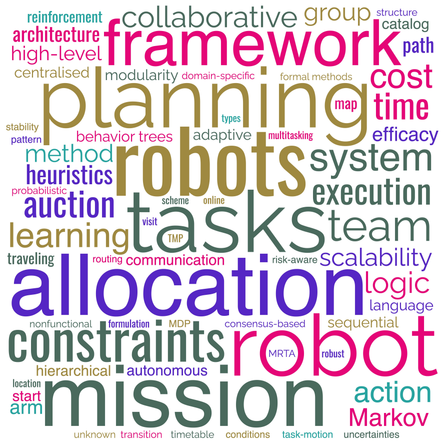


Figure 3.5: Word-cloud of terms found in the *summary* of the papers. Produced on [141]

Figure 3.5 shows a *word cloud* created by joining the text gathered from all papers for the "**summary**" data item. Similar terms were joined (e.g., task & tasks, scalable & scalability), followed by the elimination of pronouns, spaces, signs, author names, and verbs (such as substitutes, assigns, and design). The final list of terms is depicted within this figure. Terms displayed in a smaller font indicate that they occur less frequently compared to terms presented in a larger font size. Some of the most visible terms include *framework, planning, robots, tasks, allocation, constraints* and *mission*; this is expected as they are aligned with the ongoing systematic review. The terms *planning* and *allocation* stand out while the term *scheduling* does not appear in the figure; this observation aligns with the Venn diagram in Figure 3.4e, where the scheduling problem is mostly considered as part of the first two problems. This figure is only for visualisation purposes as it does not provide any unexpected term; hence, we continue in the following paragraphs by defining the terminology of interest such as task allocation, mission, task, subtask and goal.

Comparison of two terminologies for the description of tasks. To understand and classify the terminology used by the studies gathered by this systematic review, in this section, we use the information gathered from the "**terminology**" data item. It is noticeable that some discrepancies exist among studies. As an example, we compare the terminology for two studies, PROMISE [84] and GODA [142], as depicted in the Figure 3.6 summaries.

One could argue that PROMISE [37] uses a terminology focused on the *operators* (tasks are elementary operations and missions complex tasks), while GODA [142] (also used in GoalD [37]) uses the TROPOS methodology, focused on the *agents*. For the latter, goals are agent strategies and tasks are the plan to be performed by robots. Table 3.3 shows five examples of *things* that the robots require *to-do*: a) ordered visit, b) visit, c) parallel, d) fallback, and e) combination of parallel and fallback, and the terminology used by PROMISE versus the GODA. A natural language description of the to-do list is shown in column 2.

Starting with the first row in Table 3.3, *ordered visit* is described as a task in PROMISE but called a goal in GODA. However, *visit* in the second row is referred to as a task in GODA terminology, as it is seen as the composition of multiple single tasks. Continuing, there are three examples of *Missions* in PROMISE. The *parallel mission* is referred to as a goal in GODA, whereas *fallback* and combinations of this, such as the *parallel and*

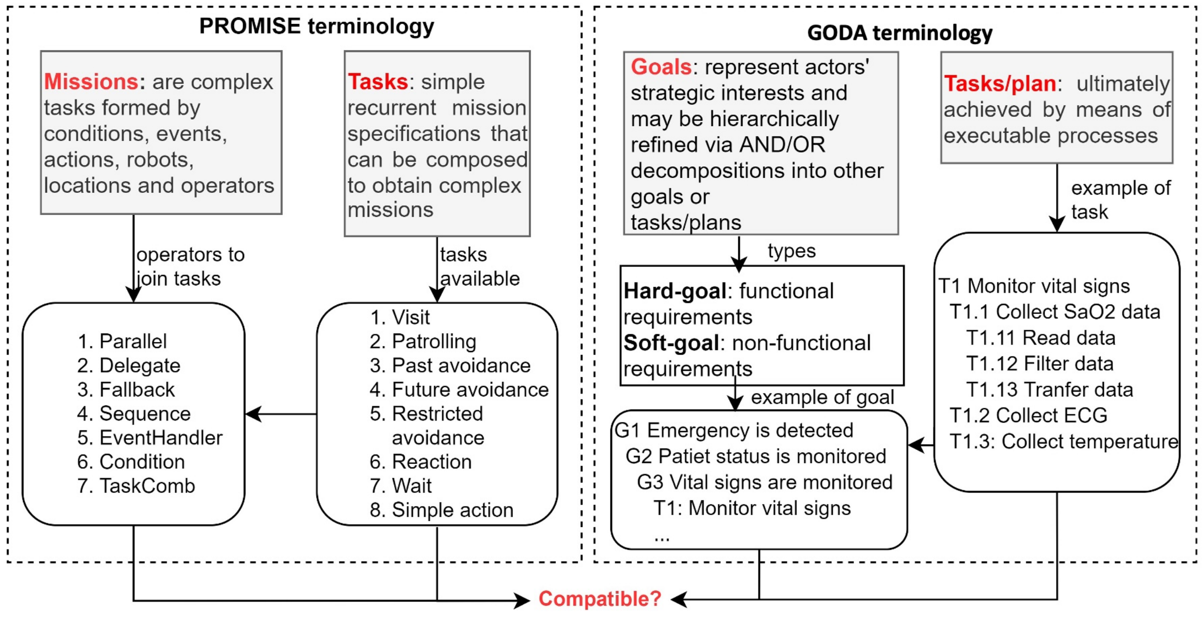

PROMISE terminology
Missions: are complex tasks formed by conditions, events, actions, robots, locations and operators
Tasks: simple recurrent mission specifications that can be composed to obtain complex missions
operators to join tasks
tasks available
1. Parallel
2. Delegate
3. Fallback
4. Sequence
5. EventHandler
6. Condition
7. TaskComb
1. Visit
2. Patrolling
3. Past avoidance
4. Future avoidance
5. Restricted avoidance
6. Reaction
7. Wait
8. Simple action
GODA terminology
Goals: represent actors' strategic interests and may be hierarchically refined via AND/OR decompositions into other goals or tasks/plans
Tasks/plan: ultimately achieved by means of executable processes
types
Hard-goal: functional requirements
Soft-goal: non-functional requirements
example of goal
G1 Emergency is detected
G2 Patiet status is monitored
G3 Vital signs are monitored
T1: Monitor vital signs
...
example of task
T1 Monitor vital signs
T1.1 Collect SaO2 data
T1.11 Read data
T1.12 Filter data
T1.13 Tranfer data
T1.2 Collect ECG
T1.3: Collect temperature
Compatible?

Figure 3.6: On the left, *mission* and *task* definitions defined by PROMISE [84]. On the right, GODA [142] terminology uses the words *goals* and *task/plan* for similar concepts. PROMISE focuses on the operators (ways to combine tasks) and GODA on the agent requirements.

fallback mission, are not supported by GODA.

Beyond PROMISE and GODA, discrepancies in the definition of tasks arise from the type of robotic system. For example, for *robotic arm setups*, a task may involve reaching a position in space or following a trajectory, moving from configuration A to configuration B through a collision-free path [129]. Meanwhile, in *cloud robots*, a task may be defined by a time length and a cost associated with the priority of the task [116]. Cloud robotics is a field that integrates cloud computing with robotics to enhance their capabilities, allowing robots to offload complex tasks to remote servers for processing, analysis and storage.

Common terminology. As previously discussed, there is no universally formal definition for the many of the terms used within the research of MRS, such as *tasks* and *missions*. To ascertain what these terms mean within the broader community, we assembled the following definitions based on the information collected from the "**terminology**" data item and its analysis.

For the definitions of *task* and *mission*, we used the three most cited papers [27, 130, 131][4]. Garrett et al. [131], the authors differentiate between motion tasks (pick, place and move) and higher-level tasks such as cleaning and cooking. Both types of tasks are formalised as action schemas in an extension of SAS+ planner. This planning formulation defines conditions (Boolean restrictions associated to states of the planning problem) and effects (updates to the value of planning variables). Schillinger et al. [27] define a task as inde-

[4]Number of citations obtained on 24th August 2024.

Table 3.3: Examples of terms defined in the PROMISE and the GODA terminology. Three examples of *missions* in PROMISE are shown with their names within brackets. Two of these missions are not supported by GODA.

	Examples of things "to-do"	**PROMISE term**	**GODA term**
a	*Ordered visit patients 1, 2 and 3*	Task	Goal
b	*Visit patient 1*	Task	Task
c	*"Robot 1 visit patient 1" and "Robot 2 check vital signs on patient 2 and 3"*	Mission (Parallel)	Goal
d	*if "Ordered visit to patients 1, 2 and 3", if fails, then "visit doctor 1", if this also fails, then "visit reception"*	Mission (Fallback)	Not supported
e	*"Robot 1 visit patient 1" and "Robot 2 check vital signs on patient 2 and 3", if Robot 2 fails, "go to reception"*	Mission (Parallel and fallback)	Not Supported

pendent activities that can be executed by the robots and that can be formally defined as an LTL formula. They also define a *mission* as the completion of all tasks. Formally, a mission is given as a set of tasks $\mathbb{M} := \{T_1, T_2, ..., T_n\}$, or its respective LTL formula (or its respective non-finite automaton). Additionally, a mission must satisfy the properties of independence (an execution of a task must not violate another task) and completeness (the completion of all tasks implies the completion of a mission).Hence, we informally define tasks and missions as,

A task usually refers to an action that a robot or multiple robots are intended to perform (e.g., cooking or visiting patients). Tasks can be define as individual action, or as a series of them (e.g., cleaning a room or cleaning a series of rooms) and might consider dependencies such as the order or their execution. Meanwhile, a mission generally refers to a similar concept but involves more complex dependencies that may encompass multiple or even all defined tasks.

Terms such as task allocation (*who*), planning (*what*), scheduling (*when*) and motion planning (*how*) are recurrently found in the literature. We guide our following definitions in papers than describe these in more detail by the application of multiple [24, 90, 130] or all [90] of this relevant MRS problems. *Task planning* is generally define as,

Task planning is the process of ordering a discrete sequence of tasks. In ordered task planning, tasks are arranged in a strict sequence. In partially *ordered task planning, tasks are not strictly sequenced, allowing for more flexibility in execution. This ordering is usually guided by the optimisation of resources.*

In [130], Dantam et al. refer to task planning as a well-established field largely evolving from planners such as STRIP into heuristic search and logic programming approaches.

Motion planning can be defined on its own as it usually refers to a lower level where the motion of the robots as it performs its tasks is considered [90, 95]. We define motion planning as follows,

> *Motion planning determines the trajectories needed to move robots from an initial to a goal location (or robot configuration) as they perform tasks under a given allocation and schedule; it might involve finding collision-free paths as robots perform the tasks.*

In some cases, the analysed studies use the term motion planning together with task planning, such as in **task and motion planning** (**TMP** or **TAMP**) [24,90,130], combining the problem of efficiently generate executable and low-cost discrete task plans, while determining the continuous motion decisions for the completion of the generated paths [24, 130]. For multi-agent systems, the term **Multi-Agent Path Finding (MAPF)** is used with a similar meaning [102]. A similar problem, **Multi-agent Pickup and Delivery (MAPD)**, is an industrial problem where a team of robots is tasked with transporting tasks, each from an initial location to a specified target location. This is solved by first allocating the tasks among the robots and subsequently finding collision-free robot paths [102].

Similarly, we consider **kinodynamic** planning as a type of motion planning where the kinodynamic model of the robots and constraints (e.g., limited velocity, acceleration and torque) are considered for motion planning [99]. Other similar terms found in the literature are **robot exploration**, defined in [22, 23] as the exploration of the environment by the robot, deciding “what to do” and “where to go”; and **Orienteering Problem (OP)**, defined as a routing problem where the objective is to determine a set of routes that maximises the summation of collected rewards in the environment while respecting the vehicles’ budget [118].

Additionally, the *task allocation* problem decides “who does what” and it is generally defined as,

> *Task allocation refers to the process of **assigning tasks** or responsibilities to one or more robots within the system. The goal is to ensure that tasks are completed efficiently and effectively by the robots that have the **necessary capabilities** and resources to perform them.*

This problem definition does not specifically address resolving the execution order of tasks, as elaborated later in task scheduling problems. Nonetheless, in numerous proposed solutions, determining the order is resolved as a by-product of assigning tasks to the robots. Special cases of task allocations require distributing a set of tasks between a robot and a human [135]. There is also a recurrent mention of the term multi-robot task allocation (**MRTA**), referring to a set of well-known algorithms to solve task allocation problems [43, 49, 91, 105, 110]. MRTA algorithms and problem formulations are covered in the 2015 survey [9], which broadly covers the Fair Division Problem, Optimal Assignment Problem (OAP), ALLIANCE Efficiency Problem (AEP), integer linear or non-linear programming (ILP or INLP), as well as the multi-Traveling Salesman Problem (mTSP) and its generalised version, the Vehicle Route Planning (VRP) [91].

Another terms found in the literature are **task assignment** [101, 135] and **goal allocation** [95], with the same or a similar meaning. Similarly, **multi-vehicle routing (MVR)** is defined as selecting a set of vehicles and their routes to visit a set of locations [90]; **multi-depot vehicle routing problem (MDVRP)** refers to the problem in which vehicles with limited payloads must pick up or deliver items at different locations [88]; and **Multi-robot Motion Planning and Goal Allocation Problem(MMP-GA)** is defined in [95] as finding feasible paths for multiple robots navigating in shared environments.

Finally, task scheduling deals with ordering and timing of the tasks [90],

> *Task scheduling refers to the process of determining the timing of tasks that need to be performed by one or more robots. The goal of task scheduling is to assigned the start and end times to each task to be performed by the robots. A valid schedule must comply with all relevant temporal constraints. It might consider task priorities and the optimisation of system objectives.*

Similar terms include **task sequencing** [121] and **task-based coordination** [92], defined as ordering the tasks in robot teams. In [90], the authors summarise the MRS problem as answering: what (task planning), how (motion planning), who (task allocation), and when (scheduling).

Finally, other terms such as **goals** [99, 138, 139], **subtasks** [43, 121], **actions** [114, 131], **events** [22, 84, 100, 106], **conditions** [106, 138] and **capabilities** [49] are found in the literature, but cannot be globally defined as they fall into one of two categories: they depend on a well-known language or technique for robot missions, or they are defined by the authors referring to specific types of tasks or task attributes. For example, research projects using the PDDL language [23, 138] adopt its terminology, which includes goals, conditions, effects, actions, etc.

Taxonomy for multi-robot task allocation problems. There exist a common terminology that applies to multi-robot systems and the type of the task allocation problem under research. Proposed by Gerkey and Mataric [39] in 2004, it still used by several of the papers gathered by this review [43,91,96,105,121]. Designed as a three-axes taxonomomy, the first axis distinguishes between single-task (ST) robots, capable of performing only one task at a time, and multi-task (MT) robots, which can handle multiple tasks simultaneously. The second axis, differentiates between single-robot (SR) and multi-robot (MR) tasks, with the former requiring only one robot for execution and the latter cooperative tasks that involve multiple robots. The third axis distinguishes between the instantaneous assignment (IA) of tasks to robots, where the problem only concerns the allocation process; and time-extended assignment (TA), additionally considering the scheduling of these allocated tasks.

EQ2. Specification of Tasks

In this section, we categorised the strategies used for the definition of *tasks* based on information derived from the "**task specifications**" data item. The taxonomy was inspired by the categories in [130]. As depicted in Figure 3.7, this consists of five categories: logic based, hierarchical representation, language based, motion based, and solutions oriented. Logic-based approaches encode robotic tasks using logic languages. The types of logic found to be used are LTL, LTL+CTL, STL, PCTL, RPCTL and CaTL. Hierarchical representations describe the tasks as hierarchical networks or precedence diagrams. Language-based approaches follow well-known languages for the description of tasks, such as PDDL and SAS+, or are based on a new domain-specific language (DSL) or formal grammar. Motion-based approaches describe tasks as dynamic object tasks (for instance, the trajectory of the end effector of a robotic arm), dynamic movement primitives or Jacobian control trajectories.

In the solution-oriented category, research papers represent tasks using lists, buffers, graphs, Markov processes, and various other data types and representations chosen to align with their proposed solutions. For example, [22] uses a method based on the reinforcement learning (RL) algorithm, SARSA, to solve the coordination of a fleet of robots. Its case study involves putting out a wildfire modelled as a dynamic propagation model, where the wildfire spreads following this model. Robots are divided into perception agents and firefighting UAVs, which can search for targets and drop extinguisher fluid to reduce the fire, respectively. When the fire is below a threshold, the fire point is pruned and that area cannot catch fire anymore. The formalisation of tasks as Markov models is specifically designed to be solved by their RL-based approach.

Regardless of the category to which the task representation belongs, it is important to emphasise that the selection of a formalism for task specification is heavily influenced by the specific problem at hand. As highlighted by [130], "each [task] notation provides advantages for certain domains or proprieties, e.g., safety properties are easily specified

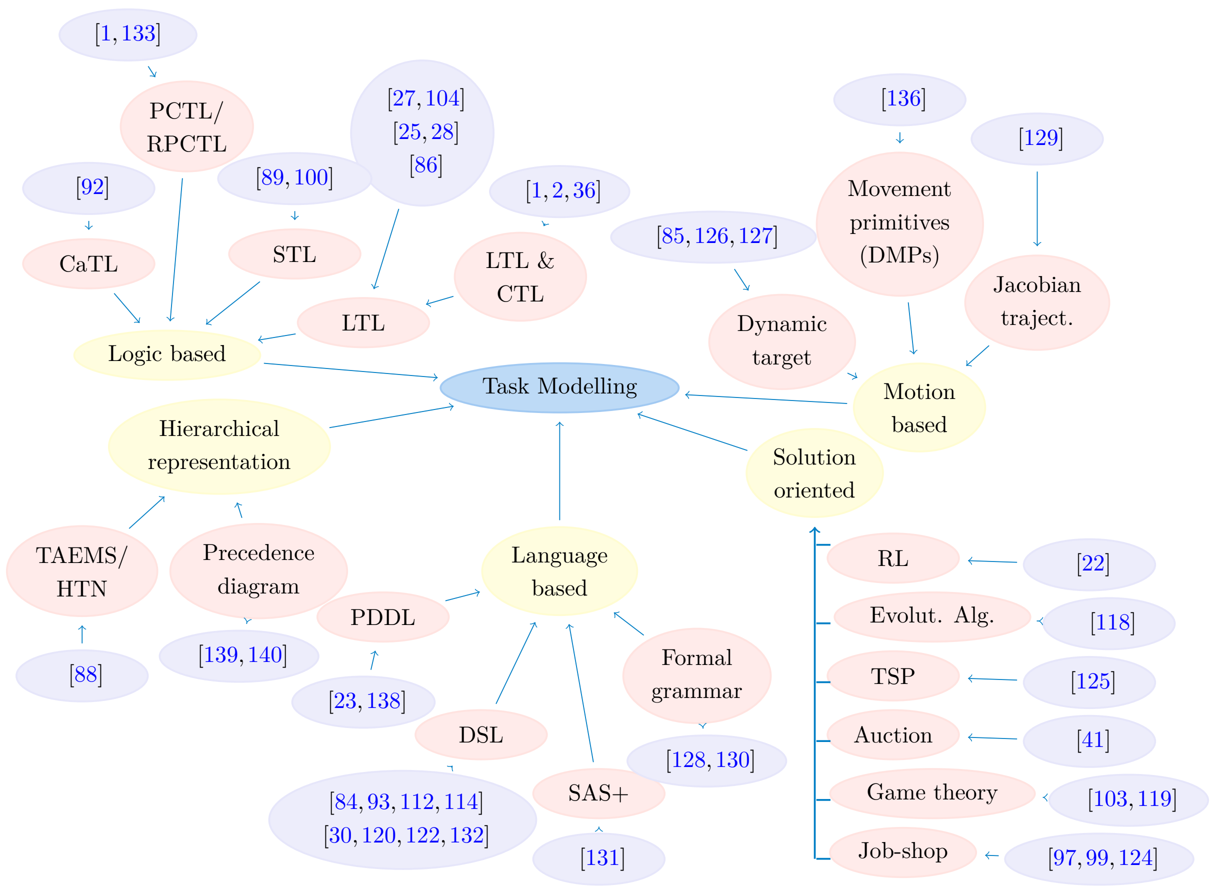


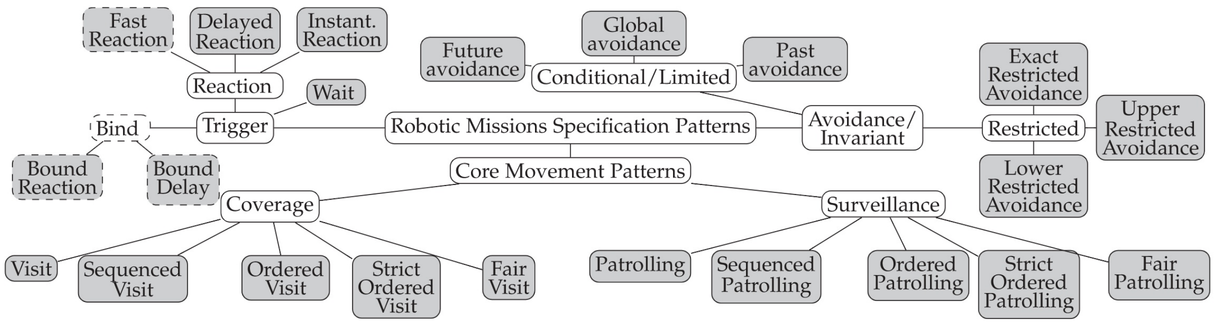
Figure 3.7: Type of strategies for the specification of robotic missions found in the literature in the past five years.

with temporal logics, action effects with PDDL, and hierarchies with grammars".

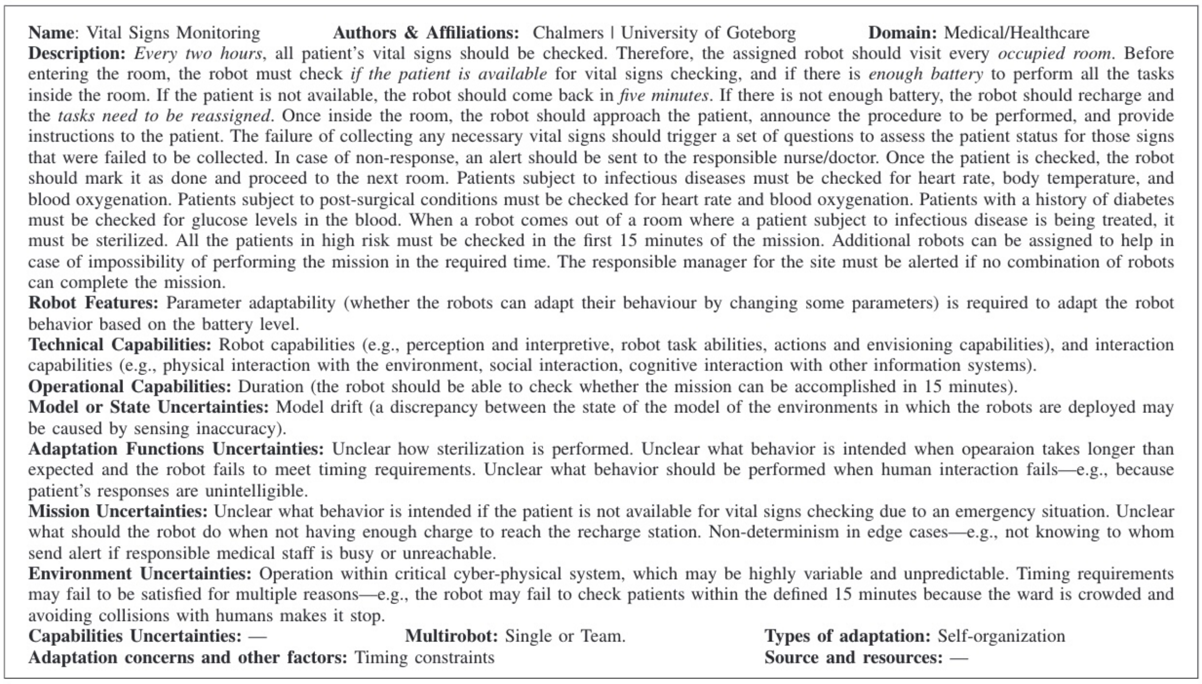
Robot mission (and task) catalogues. Three robot mission catalogues (divided into four papers) were identified in the literature: PsALM patterns [2], QUARTET [1] and ROBOMAX [20]. These were obtained from the "**task patterns**" data item.

- PsALM mission patterns (2019). Menghi et al. (2019) [2] propose 22 qualitative mission specification patterns for mobile robots described in LTL and CTL logic (see Figure 3.8a). Additionally, they created PsALM [36], a tool for the automatic translation of robot missions defined using the patterns from the catalogue, which are written in a higher-level domain-specific language, into various languages for model checkers and planners.

- QUARTET (2022). QUARTET [1], extends the patterns catalogue in [2] with a second set of patterns that handle quantitative reasoning by adding rewards (also known as costs) and probabilities. These are not supported by the original PsALM logics, LTL and CTL. Accordingly, the second catalogue uses a type of probabilistic temporal logic augmented with rewards. QUARTET is further described in Sections 4.1.1 and 4.1.2 with its semantics and tool presented as part of this thesis.



(a) Catalogue of 22 patterns (filled nodes) grouped by patterns (non-filled nodes), from [2].



Name: Vital Signs Monitoring **Authors & Affiliations:** Chalmers | University of Goteborg **Domain:** Medical/Healthcare
Description: *Every two hours*, all patient's vital signs should be checked. Therefore, the assigned robot should visit every *occupied room*. Before entering the room, the robot must check *if the patient is available* for vital signs checking, and if there is *enough battery* to perform all the tasks inside the room. If the patient is not available, the robot should come back in *five minutes*. If there is not enough battery, the robot should recharge and the *tasks need to be reassigned*. Once inside the room, the robot should approach the patient, announce the procedure to be performed, and provide instructions to the patient. The failure of collecting any necessary vital signs should trigger a set of questions to assess the patient status for those signs that were failed to be collected. In case of non-response, an alert should be sent to the responsible nurse/doctor. Once the patient is checked, the robot should mark it as done and proceed to the next room. Patients subject to infectious diseases must be checked for heart rate, body temperature, and blood oxygenation. Patients subject to post-surgical conditions must be checked for heart rate and blood oxygenation. Patients with a history of diabetes must be checked for glucose levels in the blood. When a robot comes out of a room where a patient subject to infectious disease is being treated, it must be sterilized. All the patients in high risk must be checked in the first 15 minutes of the mission. Additional robots can be assigned to help in case of impossibility of performing the mission in the required time. The responsible manager for the site must be alerted if no combination of robots can complete the mission.
Robot Features: Parameter adaptability (whether the robots can adapt their behaviour by changing some parameters) is required to adapt the robot behavior based on the battery level.
Technical Capabilities: Robot capabilities (e.g., perception and interpretive, robot task abilities, actions and envisioning capabilities), and interaction capabilities (e.g., physical interaction with the environment, social interaction, cognitive interaction with other information systems).
Operational Capabilities: Duration (the robot should be able to check whether the mission can be accomplished in 15 minutes).
Model or State Uncertainties: Model drift (a discrepancy between the state of the model of the environments in which the robots are deployed may be caused by sensing inaccuracy).
Adaptation Functions Uncertainties: Unclear how sterilization is performed. Unclear what behavior is intended when opearaion takes longer than expected and the robot fails to meet timing requirements. Unclear what behavior should be performed when human interaction fails—e.g., because patient's responses are unintelligible.
Mission Uncertainties: Unclear what behavior is intended if the patient is not available for vital signs checking due to an emergency situation. Unclear what should the robot do when not having enough charge to reach the recharge station. Non-determinism in edge cases—e.g., not knowing to whom send alert if responsible medical staff is busy or unreachable.
Environment Uncertainties: Operation within critical cyber-physical system, which may be highly variable and unpredictable. Timing requirements may fail to be satisfied for multiple reasons—e.g., the robot may fail to check patients within the defined 15 minutes because the ward is crowded and avoiding collisions with humans makes it stop.
Capabilities Uncertainties: — **Multirobot:** Single or Team. **Types of adaptation:** Self-organization
Adaptation concerns and other factors: Timing constraints **Source and resources:** —

(b) Specification of robot mission from the ROBOMAX repository [20].

Figure 3.8: Two repositories of robotic missions found in the literature, PsALM (a) and ROBOMAX. An example of a ROBOMAX mission is shown in (b).

- RoboMAX (2021). RoboMAX is a living repository of robotic missions intended as a reference to developers in the field of robotics. It contains the description of the missions in natural language, as well as a list of relevant features. These features include their application domain, robot features, robot technical and operational capabilities, types of uncertainties considered, and the types of adaptation required by the mission, if any. An example of a RoboMAX mission is shown in Figure 3.8b [20].

Standards. Regarding standards (identified as part of the "**standards**" data item), we did not encounter any explicitly related to the specification of tasks. However, two standards were found in [140], which refers to the IEC61499 modular function-based standard for the development of distributed control architectures that can be adopted for robot systems; and in [137], which mentions the ISO/TS 15066 standard that provides detailed information on how to assess the risk for a HRC application.

EQ3. Task Allocation, Scheduling *and* Planning

In the implementation of multi-robot systems, significant challenges involve the planning, allocation, and scheduling of tasks, along with motion planning. These were previously defined as part of the terminology discussion in this section. In this section, we extract the information from the "**solution**" data item to understand the types of solutions adopted recently by the research community. To compartmentalise the solutions to these problems, a classification is presented based on [143]. Table 3.4 shows the publications classified based on the problems they solved, and the class of technique/method they used to solve them. Some studies may belong to different categories as the two examples highlighted in yellow and pink in this table. This is due to several reason. For example, in pink, [89] considers all four problems, while in yellow, [41] presents a comparison between multiple task allocation and planning algorithms. We do not classify the motion planning into the same categories as we defined this as a problem involving a lower level of control of the robots (usually involving control theory). We summarise these findings below.

- Auction based. Auction and market-based algorithms are based on economics, where agents use a negotiation protocol to auction tasks between participants. A comparison between multiple auction-based algorithms for the allocation of tasks is described in [41]: **parallel auction, sequential auction, generalisation of Prim allocation, repeated parallel auction, combinatorial auction algorithm**, etc. Their study focused on their performance under failing communication between agents.[5] In the work presented in [117], a market-based optimisation algorithm is used to allocate tasks that remained unallocated after an initial assignment or are

[5]Although the work in [41] creates schedules after the tasks are allocated to robots, this is implicit in the allocation stage, and therefore it is not considered as scheduling in Table 3.4.

Table 3.4: Recent studies classified by the type of robotic problem (task planning, allocation and scheduling, and motion planning) and the methods/techniques used to solve them. Two research studies are highlighted in pink and yellow to understand why they belong to different categories. In pink, [89] considers all four problems solved by different techniques. In yellow, a comparison between multiple task allocation and planning algorithms is presented [41]. The classification of the different techniques/methods is based on [143].

		Problem			Problem
		Task Planning	Task Allocation	Task Scheduling	Motion Planning
Technique/Method class	Auction based	-	[114, 117] [41]	-	[111, 138] [98, 108] [105, 125] [85, 126] [90, 106] [49, 137] [99, 127] [28] [89]
	Optimisation based (deterministic)	[95, 101, 110, 129]	[98, 121, 127, 135] [94, 101, 106, 122] [30]	[98, 101, 121, 124] [30, 116, 122]	
	Optimisation based (heuristics)	[48, 105, 115, 131] [97, 137] [41]	[47, 48, 90, 105, 115] [49] [41]	[47, 48, 105, 115] [49, 137]	
	Optimisation based (metaheuristics)	-	[88, 139]	[88]	
	Learning based	[22, 91, 109]	[22, 91]	[91]	
	Constraint based	-	[93, 108, 111]	[111] [89]	
	Search based	[23, 90, 138]	[23, 90]	[90]	
	Logic based	[25, 27, 28, 100, 104] [89]	[100, 104] [89]	[28]	
	Game theory	-	[87, 103, 119]	[87]	
	Hybrid	[86, 134]	[43, 92, 99, 102]	[43, 92, 93, 102]	
	Other	[96, 107]	[86, 107, 140]	[140]	

left uncompleted due to a robot failure. A domain specific language and a ROS code generator for the allocation of tasks are proposed in [114], solving the allocation problem with an auction-based architecture with one of the robots chosen to be the auctioneer.

- Optimisation based (deterministic). In [127], finding trajectories for a robot arm is formulated as a **local optimisation problem** co-optimising the navigation costs and the manipulation-related costs, mixing the task planning and motion problems. In [95], the authors present the optimisation problem as a **minimum-Cost max-Flow** problem. This is a type of optimisation problem that involves finding the maximum flow in a network while minimising the total cost of the flow. In [124], the scheduling of tasks is modelled as a job shop scheduling problem (JSP) and solved using **constraint programming**. Liu et al. [101] solve the task planning, allocation and scheduling as a **quadratic assignment problem (QAP)**, which can be transformed into a **linear programming (LP)** problem. QAP is a type of optimisation problem that involves assigning a set of facilities to a set of locations in such a way as to minimise a quadratic objective function.

Proposing a method to distribute and allocate a series of tasks among humans and robots, [135] suggests utilising a Convolutional Neural Network (CNN) to estimate human fatigue. The goal is to minimise human fatigue by analysing the correlation between allocating a task to a human and their fatigue levels. In [94], authors use a **distributed constraint optimisation problem (DCOP)**, a type of optimisation problem that involves finding a solution to a problem that is distributed across multiple agents. In [106], a resilient task allocation algorithm is invoked to redistribute robots among tasks while taking into account their degraded capabilities leveraged by a centralised **mixed-integer quadratic program (MIQP)**. Similarly, [121] uses a **mixed-integer linear program (MILP)** as part of the task allocation and the synthesis of robot schedules.

Finally, [98] solves the task allocation problem using the distributed **max-sum algorithm**, in which multiple agents need to make decisions that affect a shared objective function. It also uses Markov Decision Processes (MDP) and Mixed Observed Markov Decision Processes (MOMDP) (depending on whether full states or partial states are observable) and **optimal policy synthesis** to generate robot schedules/plans.

- Optimisation based (heuristics). Optimisation-based (heuristics) refers to a class of algorithms that are designed to find a near-optimal solution to a problem by making a sequence of informed decisions. In [48], the authors present a series of heuristics, such as the **Minimum Transportation Cost (MTC)** and **Predict Earliest Finish Time (PEFT)**, for the planning, allocation and scheduling of tasks among static and mobile robots in a factory. In [137], multi-objective **mixed integer linear programming problem (MILP)** is used for the allocation of tasks between humans and robots. In [97] a heuristics that iterative adapts the initial plans using path rewards is presented. Otte et al. [41] use a mixed integer programming solution (MIPG) and the following heuristics for cost approximating for the comparison of auction algorithms: the **multi-travelling salesman problem (mTSM)**, maximizing profit **(max sum)**, minimising cost **(min-sum)**, and minimising the maximum distance travelled by any particular robot **(min-max)**.

  For task allocation, [90] uses two heuristics: **allocation percentage remaining (APR)**, which guides the search based on the quality of the allocation, and **normalised schedule quality (NSQ)**, which guides the search based on the quality of the schedule makespan. In [105], authors propose a new heuristic estimator for computing interference cost that accounts for the kinodynamics and the current state of the robots in an MRS for the allocation (and planning, indirectly) stage. Also, [49] defines a series of heuristics, including mixed-integer quadratic programming (MIQP) for a framework covering from allocation and scheduling to the execution of tasks in MRS.

For human-robot collaboration, [47] uses a series of heuristics over graph-structured tasks for task allocation and scheduling, identifying a group of tasks to be executed first (by a given pre-defined order and computing their priority value) and assigning them to the agent that is better at completing the task.

- Optimisation based (metaheuristics). Metaheuristic optimisation methods are high-level algorithms to find approximate solutions; they can be based on various mathematical or heuristic concepts, such as randomisation, simulation, and estimation of distribution algorithms. **Pheromone Ant Colony Optimisation** based on the HUMANT algorithm is used in [139] for the allocation of tasks to humans and robots.

  For the task allocation and scheduling problem, [88] employs a multi-objective optimisation method based on **genetic algorithm (GA)**, called Coalition-Based Metaheuristic (CBM).

- Learning based. In terms of learning-based approaches, [22] presents a high-level decision-making problem modelled as a multiagent partially observable semi-Markov decision process (MA-POSMDP) and solved by a variant of the reinforcement learning **SARSA algorithm**, MA-SARTSA. Similarly, [91] presents a graph reinforcement learning (RL) and CNN neural architecture to solve multi-robot task allocation (MRTA) problems that involve tasks with deadlines, workload and robot constraints. In [109], researchers combine dynamic team **Q-learning** based on multi-agent reinforcement learning with heuristics to accelerate space exploration and planning.

- Constraint based. In previous work, [93] utilise a constraint solver to synthesise correct-by-construction feasible allocation of tasks to robots. Meanwhile, in [108] and [89], the authors use the Control Barrier Function (CBF) to allocate each task to the robot that has less possibility of violating its associated constraints.

- Search/Planning based. The solution presented in [23] focuses on the specification of the system in PDDL with subtasks to decide exploration versus pursuing a goal, solved by **PDDL symbolic planners**. Similarly [138] proposed a combination of PDDL and **share control templates** (for example, when moves of a joystick by the user are mapped into lower-level robot motion maps) for the generation of plans under shared control and supervised autonomy.

  On the other hand, [90] uses an **incremental order motion planner** as a search-based algorithm that can simultaneously satisfy planning constraints, comply with task requirements and optimise time schedules. This work also proposes *GRSTAPS* framework, which interleaves all four problems: task planning, allocation, scheduling, and motion planning, while allowing information between modules

- Logic based. In [100], high-level, reactive robot tasks that include timing constraints are modelled in event-based STL logic and solved using **temporal logic synthesis** for the generation of MRS plans. Similarly, in [89], the event-based STL syntax is extended adding conjunction into the tasks description. [104] uses **model checking** for MRS task planning under individual and collaborative temporal logic specifications.

- Game theory. **Consensus-based timetable algorithm (CBTA)** in [87] solves the decentralised simultaneous multi-agent task allocation problem, aiming to minimise the start time of each task, iterating between a timetable construction phase and a consensus phase. The authors of [103] propose a game-theoretic multi-robot task allocation framework for large MRS in dynamically changing environments, where robots select tasks to perform and repeatedly revise their task selections in response to changes in the environment.

- Hybrid. Hybrid techniques use more than one technique to solve one problem. For example, Fang [86] uses **satisfiability theory** over graphs created by joining the robot models and their desired behaviour modelled as Bu̇chi automata (from LTL mission specifications), and a **token-based** approach for the allocation of tasks. In a token-based framework, tasks and resources are abstracted as tokens and passed locally among robots. Each robot decides whether to keep a token or pass it around. Information on how the token has been passed around is saved for each token. In [102], the authors combine **marginal-cost assignment** heuristic and a meta-heuristic strategy based on **Large Neighbourhood Search** for the multi-agent pickup and delivery (MAPD) problem involving allocation and planning. In [92], **mixed integer linear programming (MILP)** and **satisfiability theory** for CaTL logic formulae is used to encode constraints of the mission and robot teams, and for the coordination of the MRS. This work also includes the use of heuristics for the optimisation of availability robustness. In [43] a Boeing 777 case study uses the **flexible job shop problem (FJSP)** formulation and **multiple mixed integer programming** algorithms for planning, allocating and scheduling of tasks between multiple humans and robots. The authors of [128] merged a search-based **Monte Carlo tree search (MCTS)** algorithm and **simulated annealing (SA)** to automatically design agent behaviour trees, where the former expedites the aggregation of the most functional subtrees.

  In [99], the authors mixed a graph-based abstraction to partition the navigable space into polygons, heuristic search for simultaneously determine task assignments and polygons to be traversed by robots; and multi-robot motion planner for the synthesis of robot trajectories. In previous work, RPCTL logic is used in [93] for the scheduling of tasks, in conjunction with genetic algorithms (GA) for the optimisation of multiple objectives.

- Others. For allocation, [86] uses an heuristic, **token-based** conflict resolution task allocation algorithm to generate near-optimal allocations. Token-based conflict resolution is a method used to allocate tasks in multi-agent systems, where a token or a set of tokens is used to represent the right to perform a task. The tokens are passed from one agent to another in a predefined sequence to resolve conflicts and ensure a fair allocation of tasks. Multitasking is handled in [96] inspired in **information invariant theory**. In [107], the task allocation process applies Bayesian inference to compute **confidence levels** of each robot to estimate the agents' specialisation levels, hence, allocating the task in hand to the most specialised robots through heuristic methods. Finally, in [140] a multi-criteria optimisation algorithm based on [144] is used for the task scheduling and online re-scheduling of a human-robot scenario.

Uncertainties. The self-explanatory Table 3.5 depicts the types of uncertainty encountered in the selected studies, based on a classification of uncertainty types adapted from [20]. The information is extracted from the "**uncertainties**" data item. The most commonly studied uncertainties are related to the incompleteness of the environment (4 papers), the addition of new tasks at execution time (5 papers), the modelling of dynamic objects (4 papers) and robots' failure or leaving the mission (6 papers). This Table is particularly useful for understanding the complexity behind robot and MRS systems. There is not a single paper that captures all sources of uncertainty, with [22] covering the most (5) types of uncertainties: unknown environment, model drift, limited action space, variability of task execution and probabilistically modelled capabilities of the robots.

Formal specifications and formal methods for robotic systems. A notable contribution of this thesis involves employing formal languages and methods to address the description of robotic systems and robotic missions, and solve the task allocation and scheduling problems in MRS. In this context, we are interested in understanding the applications of formal methods within this research community. Our findings are summarised in Table 3.6 and elaborated upon in the following discussion. The information presented was extracted from the "**formal methods**" data item.

The theory behind translating Linear Temporal Logic (LTL) formulae into Büchi automata and synthesizing plans that comply with all LTL formulae falls under the broader umbrella of formal methods and model checking. Specifically, this process is often associated with the area of **reactive synthesis**, where the goal is to automatically generate a system that satisfies a given specification, expressed in temporal logic. This concept is studied in [86] for MRS control synthesis from LTL specifications. Non-deterministic

Table 3.5: Uncertainty classes, sources and causes for robot systems. The causes of uncertainty are addressed in the references in the last column. Adapted from [20].

Class	Source	Cause (addressed in the references)	References
Model or state uncertainty	Incompleteness	Unkown environment Unknown cost of paths or tasks	[22–25] [110, 123]
	Model drift	Discrepancy between the robot maintained state and the actual state	[22, 101, 113, 127]
	Complex models	Unknown robot density in a sector	[101]
	Events	Trigger events Adversarial attacks	[84, 100] [106]
	Actuation	Limited action space	[22]
Adaptation functions	Decentralisation & coordination	Changes in communication topology	[110, 112]
Mission uncertainty	Future mission changes	New tasks at runtime Changes in tasks deadlines	[86, 87, 98, 105, 137] [121]
Environment uncertainty	Execution context	Dynamic objects Slope/roughness of the terrain Variability depending on task execution	[89, 103, 126, 127] [126] [22, 131, 134]
	Human in the loop	Human fatigue/performance Robot requires human intervention Human enters the working space	[47, 135] [137] [121]
Capabilities uncertainty	New or defunct capabilities	Sensor failure Probabilistic completion capabilities	[85] [22, 30, 93]
	Changing capabilities	Changing or unknown capabilities Degradation of robots communication	[25, 108] [41]
	Robot failure	Robot dropout	[92, 97, 117, 119], [118, 121]

Büchi automata (NBA) are used in [28]. Further, in [104] and [27], nondeterministic finite automata (NFA) are constructed from LTL formulae. The NFA must fulfill the input LTL formula, and contain possible alternative sequences to fulfill the formula called *strategies* [27].

Menghi et al. [25] propose a new decentralised planning algorithm, MAPmAKER, for the synthesis of robot plans from missions defined in a variation of LTL, under partial knowledge of the system and environment. In [89, 100], the mission specifications are defined in event-based STL, and subsequently translated into LTL. The description of LTL specifications was also used in the robotic missions catalogue in [36]. This catalogue also contains the formalisation in CTL, and is complemented by a domain specific language (DSL) to ease its adoption. Additionally, a formally verifiable DSL through model checking has been proposed in [132].

Tasks and robotic requirements formalised in temporal logic can also be solved by other methods. In [92], the tasks are formally specified in capability temporal logic (CaTL), which is a fragment of signal temporal logic (STL). As the authors mentioned, "[t]he qualitative semantics of STL (and thus, CaTL) can be encoded as mixed integer linear constraints on trajectories of this linear system". Hence, the robot trajectories are solved as a mixed-integer linear program (MILP) problem using off-the-shelf MILP solvers such as *Gurobi*.

Chen et al [98] model each robot planning problem as a Markov Decision Process (MDP) or a partially observable MDP (POMDP) synthesising policies that represent the robot plans. The synthesis is based on a piecewise linear dynamic programming approach. MDPs were also used in [24] along with reinforcement learning to improve previously synthesised robot plans. Moreover, modelling the task planning problem as a **symbolic planning** problem, it uses an **answer-set** solver to generate the initial robot plans. In comparison, in previous research, we used MDPs to model the scheduling of tasks. We used **probabilistic model checking** (PMC) to synthesise robot plans that minimise the total idling time of the robots, needed to synchronise robots in time due to task ordering constraints [93, 122]. System properties were specified in probabilistic temporal logic extended with rewards (RPCTL). This type of logic was also used to define the robotic mission catalogue in [1].

A systematic way to derive new information and conclusions from existing knowledge or axioms is by providing **inference rules**. These are used by [96] to form robot coalitions (robots sharing one or more tasks at a time) modelling tasks such as, maintaining relative position and monitoring a target within a close proximity. These tasks build inference rules to obtain, for instance, the global position of a robot based on two tasks specifying the relative positions between two other robots.

Another approach for the specification of tasks is by building behaviour trees based on a **formal grammar**, as in [128]. **Behaviour trees** are directed rooted trees comprised of

Table 3.6: Studies found to use formal methods or logic specifications. The last two rows belong to repositories of robotic missions written in different types of temporal logic.

Method	Type of model	Propety formalisms	Reference
Reactive synthesis	Büchi automata	LTL	[86]
Reactive synthesis	NBA	LTL	[28]
Reactive synthesis	NFA	LTL	[27, 104]
LTL planner	-	LTL variant	[25]
Reactive synthesis	Büchi automata	event STL	[89, 100]
MILP	-	CaTL	[92]
Model checking	Verifiable DSL	-	[132]
Linear programming	MDP/POMDP	-	[98]
Symbolic planner + RL	MDP	-	[24]
PMC + constraint solver	MDP + constraint allocation problem	RPCTL	[93, 122]*
Information invariant theory	Inference rules	-	[96]
Monte Carlo tree search	Formal grammar + behaviour trees	-	[128]
LTL planner	LTL + behaviour trees	LTL	[84]
SMT	Constraint planning problem	-	[130]
-	-	LTL + CTL	[36]
-	-	RPCTL	[1]

* = previous studies to this thesis

execution nodes and control flow. Examples of control flow nodes are fallback, sequence and parallel. Execution nodes are actions that robots must complete and condition nodes; the latter is a Boolean whose state depends on the robots and environment. Inspired by behaviour trees approaches, PROMISE [84] proposes a new language for the specification of robotic missions build on top of LTL mission specifications.

Finally, in [130], Dantam et al. implement the task planning problem as a **constraint-based planning** problem, using satisfiability module theory (SMT) solvers to generate alternative task plans. Similarly, a constraint solver is used in [93, 122] to solve the allocation of tasks to robots. These two studies are our previous work, extended in this thesis. These findings suggest that there is active research on formal methods applied to robotic systems. They also offer insights into the limited use of formalisms that incorporate probabilistic models and properties in these systems.

EQ4. Future Work

In order to offer insights into potential future research directions that the gathered papers might explore, thereby shaping the forthcoming landscape of research, this section enumerates the gaps and prospective endeavours identified from the "**gaps**" data item.

First, we examined each paper to identify any gaps and areas for further work identified by the authors. Subsequently, we categorised the identified gaps into types, and organised the information based on the nature of the concerns. The following categories emerged:

- Extending the system language for the specification of the robotic system. This involves considering other language candidates or an expansion of the current one to address limitations in the expressiveness of the tasks, tasks constraints and system specification [1,23,47,87,88,104,117,128]. For example, extending the expressiveness of the well-known PDDL language is mentioned in [23].
- Scalability. In studies where there is a limitation in the number of supported robots or tasks due to the state-explosion problem, increasing the scalability is stated as future work [49, 93].
- Online adaptation and its limitations. The surveyed research mentions as areas of further study the adaptation of the robotic system and its limitations such as switching between a centralised and a decentralised architecture for the allocation of tasks [49]; allocating or re-scheduling tasks at runtime [30, 89, 90, 92, 102, 128]; and integrating new robots and new objects reducing or modifying the free-space at runtime [111, 114].
- Computational complexity. Understanding [88, 93, 102] and reducing [87, 104] the computational complexity of the solutions proposed are mentioned as areas requiring further research, especially in papers rely on more than one technique.
- Communication issues. Analysing the failures in the communication between robots and finding ways to make the system more robust and resilient [88].
- Real world applications. Testing the proposed solution in real-world applications or more realistic case studies. Examples in the literature include a greenhouse environment [88], an industrial case scenario [137] and underground mining applications [105].
- Preventive events. For example, using pre-failure warnings to modify a specification at runtime [89]. This is also part of the online adaptation strategies and requires means for the prediction of future failures.
- Dynamic and uncertain environments. To account for dynamic objects where obstacles change positions overtime, and partially-observable environments [90].
- Heterogeneous robots. By modelling robots with different capabilities to perform the given tasks. This research gap was identified in [118]. However, it has been considered in multiple research studies, including this thesis.

- Environmental conditions. Considering multiple environmental conditions in the system specification, for example, related to the type of terrain, the amount of light, the velocity of the wind, local gravity dependent on the altitude and the air density [113]. These may also interfere with the quality of the sensors' reading.

- Specifics to the approach. This category involves future efforts aimed at enhancing the methodologies applied. For instance, fine-tuning specific parameters within meta-heuristic techniques or broadening the semantics derived from a particular language syntax, such as PDDL [23, 138].

3.4 Analysis

This systematic review comprehensively covered multiple challenges related to the description and partition of tasks in robotic systems. These challenges include the allocation and scheduling of tasks for individual robots. Through an extensive examination of both solutions and problem formulations, we identified two other distinct well-defined problems, namely task planning and motion planning. These were defined through the research question **EQ1**. Within this context, we delved into the diverse terminologies employed to denote robot actions, such as *tasks, missions* and *goals*. Specifically, we conducted a detailed comparative analysis of the terminologies used in PROMISE and GODA, elucidating their disparities (refer to Table 3.3). This comparative exploration led to the proposition of a unified definition for the term *task*.

In addressing research question **EQ2**, we identified several repositories for robotic missions developed by the research community with the aim of creating living repositories whose missions can be reused, as is the case of RoboMAX, but also to ease the adoption of intricate logic-based languages, such as in PsALM and QUARTET. The use of logic languages removes ambiguity from the specification of missions and is further studied as part of this thesis in Chapter 4. Figure 3.7 presents a classification of various techniques employed in task definition. These include logic-based, hierarchical representation, language-based, solution-oriented and motion-based.

Despite our comprehensive search, no applicable standards for task description were found, although standards do exist for other facets of robotic systems. Noteworthy among these are the IEC61499 for distributed control architectures, also adopted for distributed multi-robot systems, and the ISO/TS 15066 for risk assessment in Human-Robot Collaboration (HRC) scenarios.

Addressing **EQ3**, we provided a comprehensive classification of the collected papers based on the nature of the problem tackled (task allocation, scheduling, and planning) and the corresponding solution types. This detailed classification is outlined in Table 3.4, accompanied by a thorough description of the various techniques, supported by citations to the respective collected papers. Among these techniques, deterministic and heuristic optimisation approaches emerged as the predominant choices within the collected papers.

Additionally, we systematically extracted and categorised their different types of uncertainties, displayed in Table 3.5. Our analysis revealed 22 different types of uncertainties addressed by the research community. In addressing **EQ3**, it becomes apparent that future endeavours should focus on incorporating multiple of these types of uncertainties in a unified robotic system model, as well as new techniques for system adaptation at runtime to mitigate their effects. We also investigated whether the collected papers incorporate any formal methods as outlined in Table 3.6. A comprehensive description of the diverse formal techniques is provided, substantiated by references to the corresponding collected papers. We also emphasised the limited use of probabilistic models and properties for robotic systems, which could significantly enhance the modelling of uncertainties that impact their adoption in real-world applications.

Concluding with **EQ4**, we explored potential future directions within the dynamic landscape of task allocation and scheduling for robotics. This examination involved presenting a list of research gaps and areas for future work identified within the collected papers. By doing so, we contribute to the ongoing discussion and evolution of this fascinating robotics subfield. The insights derived from this analysis not only deepen our understanding of existing challenges but also provide a roadmap for future research.

3.5 Discussion and Limitations.

Selection of targeted venues. Within the framework of our research questions, our primary focus revolved around unravelling the strategies put forth by the robotics research community to solve robotic systems-related problems such as task allocation, task scheduling, and the adaptation of these systems. Our intent was to gain an understanding of the latest terminology, methodologies, algorithms, and conceptual frameworks that have been advanced within this specialised domain. To ensure the relevance and specificity of our findings, we predominantly targeted venues that are recognised by experts and scholars within the field of robotics.

As part of the research questions, we were interested in understanding what the robotics research community was proposing for the study of task allocation, task scheduling and adaptation of robot systems. Hence, we targeted venues familiar to this research community. Nevertheless, we are aware that variants of the same problems are studied in other fields such as scheduling computing and manufacturing processes [42]. A threat to the validity of our approach is the likelihood of overlooking relevant research publications that might be disseminated in alternative venues not considered in this survey.

Time frame. By confining our search to publications from the last five years, we aim to present a snapshot of the latest advancements and trends in the specified areas covered by this survey. It is crucial to acknowledge that our objective is not to present an exhaustive literature review encompassing the most common or well-established methods, for instance, those relevant to the domains of task scheduling or the adaptation of multi-robot systems. Rather, our emphasis lies in offering insights into the cutting-edge initiatives that have surfaced in the recent research landscape. For readers seeking a comprehensive understanding of the foundational and widely recognised formulations, we refer to authoritative texts, such as *Artificial Intelligence: A Modern Approach*, by Russell and Norvig (2010) [145] and the *Handbook of Scheduling: Algorithms, Models, and Performance Analysis* by Blazewicz et al. (2019) [42].

Division of scheduling, allocation and planning problems. Another challenge in classifying the papers relies on the intricacy of the solutions they proposed. It is not straightforward to differentiate between task allocation, scheduling and planning for multiple reasons. The first reason is that, as mentioned in **EQ1**, there is a wide range of terminology used to refer to variations of these problems. This makes it difficult to identify what problem is going to be solved by only looking at the description of the problem and the problem definition. A second challenge is that multiple problems may be solved in a single paper. Early on, we needed to draw a distinction between what we classify as task allocation, task scheduling and task planning, as described in the **Demographic Information** part of Section 3.3. However, this initial assumption was challenged as we discovered different terminology used for very similar problem definitions. Hence, in **EQ1** we redefine these problems based on the collective information from the *relevant studies*.

Informing the thesis: key literature insights. The literature review provides insights into the terminology and types of solutions adopted by the research community for two problems we are interested in this thesis. Other findings that informed the work carried out in this paper are as follows.

- There is a lack of repositories or catalogues of robotic missions specified in formal languages, specifically that consider quantitative reasoning such as: "what is the expected energy consumed by all robots when all tasks are completed?" or "what is the probability of failing with the mission?".

- The use of formal methods for the synthesis of robot controllers is considered in multiple studies, however, their research considering probabilistic properties is limited.

- There is no consensus on the definitions of terms like task, mission, and similar concepts such as goal. Mathematically defining the adopted terminology is a critical step s.t. readers clearly understand its application and limitations.

- Scalability issues were reported, especially due to the state-explosion problem.

- A wide range of uncertainties can arise in the modelling of MRS, such as the probabilistic completion of tasks, potential sensor failures, variations in terrain slope and roughness affecting robot travel behaviour, and unknown path costs. To advance the state-of-the-art, it is crucial to identify a subset of these uncertainties that are most relevant to the specific problem at hand and develop solutions that enhance the system's resilience.

Finally, after considering the different terminology, we refer to the two main problems addressed in this thesis as *task allocation* and *task scheduling*, i.e., the partition of tasks into robots, and the generation of individual robot plans.

3.6 Summary

Our systematic review spans the last five years of research within the robotics community. It specifically delves into the allocation and scheduling challenges encountered in robotic systems. Additionally, the review addresses the formalisms employed for describing robot tasks, explores the associated terminology, examines the various uncertainties modelled in these systems, evaluates the types of formal methods applied, and identifies gaps for future research.

As the momentum grows in the research area of task allocation and scheduling for robot systems, surveys like this, roadmaps and (maybe in the near future) standards must update the community on what has been accomplished, where the field is heading, and what common ground, algorithms, etc., can be followed to ease reusability, accessibility and avoid repetition of work. Moreover, such surveys contribute to the independent maturation of the field, setting it apart from other applications of allocation and scheduling beyond the scope of robotics, such as manufacturing and computing. For a more in-depth analysis and summary, readers are directed to Section 3.4.

This systematic review further motivates the need for formalisms that can capture a wide range of mission constraints, multiple conflicting optimisation objectives, as well as different types of uncertainties inherent in the use of multi-robot systems. In the following chapters, we propose solutions for the specification of a combination of these, with a particular focus on the task allocation and scheduling problem.

Chapter 4

Mission specification

This chapter presents a methodology for specifying the missions to be carried out by a robotic team. The chapter has two parts. In the first part, we describe the use of a domain-specific language to specify quantitative requirements for robotic missions (Section 4.1). In the second part, we present a systematic approach to specifying all the information required for the task allocation and scheduling associated with an MRS mission (Section 4.2).

This domain-specific language (DSL) based method for specifying quantitative requirements of robotic missions has been developed as part of a larger project called QUantitAtive RoboTic mission spEcificaTion patterns (QUARTET) [1], and comprises a set of 22 patterns (see Figure 4.3) for the specification of quantitative properties for robot missions. QUARTET provides a method for mapping these patterns to RPCTL logic formulae in order to be used in conjunction with probabilistic model checkers, such as PRISM [71]. As the patterns were identified jointly by the QUARTET research team, they are presented briefly in the chapter preliminaries (Section 4.1.1). The mapping method, which establishes the semantics of these patterns–together with a tool that automates this mapping–have been developed as part of this PhD project, and therefore they are contributions of this thesis and are presented in detail in Section 4.1.2. QUARTET covers a wide spectrum of robotic mission pattern types identified from the research literature and domain experts. These patterns support the specification of requirements corresponding to the quantitative aspects of a mission, such as the *minimum* and *maximum* value of a measure (energy used, time taken, number of tasks completed, etc.) when a mission is completed.

However, only specifying the requirements of MRS missions using patterns such as those provided by QUARTET is insufficient for MRS task allocation and scheduling. For these to be possible, additional information is required. This includes, for instance, the specification of robot capabilities, the tasks that the robots can perform with these capabilities, and the locations and robots. To allow the systematic specification of these key aspects of MRS missions, we introduce a DSL for tasK specificAtion for heterogeNeous rObot teAms (KANOA, Section 4.2). KANOA specifications define the tasks (with their constraints, subtasks, etc.), the world model (e.g., physical locations and boundaries), and the mission (with its requirements expressed using formalisms such as those provided by QUARTET) of an MRS separately, in a separation-of-concerns fashion.

Finally, related work and the summary of the chapter are presented in Sections 4.4 and 4.5, respectively.

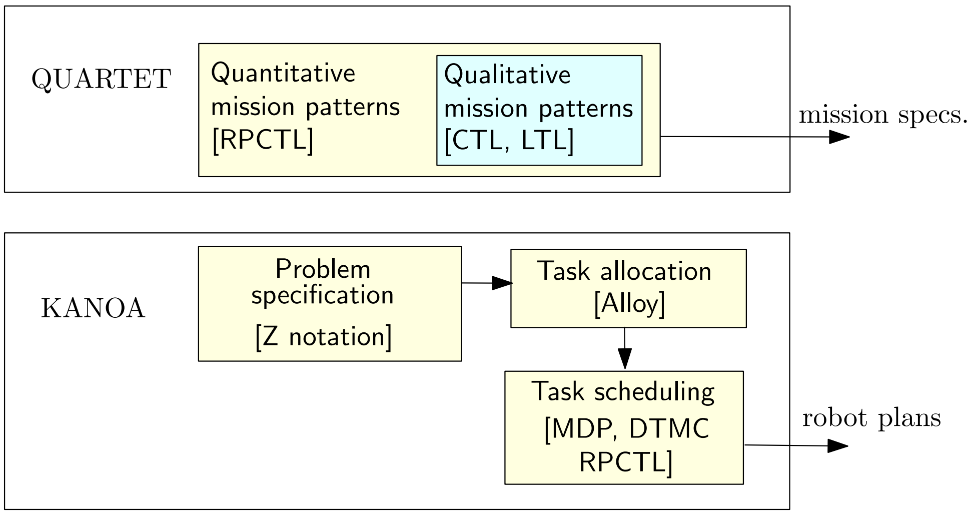

QUARTET
Quantitative mission patterns [RPCTL]
Qualitative mission patterns [CTL, LTL]
mission specs.
KANOA
Problem specification [Z notation]
Task allocation [Alloy]
Task scheduling [MDP, DTMC RPCTL]
robot plans

Figure 4.1: High-level description of the two contributions of the chapter, QUARTET and KANOA. KANOA's task allocation and scheduling stages are covered in the following chapters of this thesis.

Context for QUARTET and KANOA frameworks. In this part, we provide context to the reader for the current and subsequent chapters by briefly introducing the frameworks developed as part of this thesis. This chapter presents two orthogonal contributions to the description of mission specifications for robotic systems. First, QUARTET comprises a catalogue of quantitative mission specifications automatically translated from a proposed pattern-based DSL to RPCTL logic. An example of these requirements is: *minimise the energy to complete a mission.* Second, KANOA is an end-to-end tool for the description of task allocation and scheduling problems for MRS formally defined within this chapter. We intend these tools to be standalone s.t. they can be used separately by stakeholders.

KANOA provides the means to explicitly define other information relevant to MRS scheduling problems, such as locations, paths, probabilities of failing while travelling and possible task retry. As KANOA deals with the specification of task scheduling problems, we allow the generation of task schedules to be guided by multiple optimisation requirements. Currently, the KANOA DSL allows three optimisation requirements: "minimise the travelling cost of the mission", "maximise the probability of success by the end of the mission" and "minimise the robots' idling time until completing the mission". It is worth noticing that these requirements can be defined in QUARTET and automatically formalised into RPCTL. However, the semantics of KANOA and QUARTET target different problems.

The quantitative specifications from QUARTET assume the existence of a model on which these RPCTL specifications can be verified. In contrast, KANOA generates automatically models relevant to task allocation and scheduling problems. The models are encoded using various formalisms–including various Markov models. As KANOA synthesises robot plans using probabilistic model checking, this tool also formalises some of the required properties in RPCTL. This slight overlapping between QUARTET and KANOA can be further explored in future work.

The KANOA end-to-end framework is detailed in this thesis, comprising the following components: (a) a formal description of the parts and constraints of the type of problems solved by KANOA in Z-notation; (b) a user-friendly DSL for the description of the problem complying with the Z specifications; (c) a formal description of the task allocation problem as a constraint solver in Alloy (based on the Z specifications); (d) a formal description of the multi-objective task scheduling problem using the PRISM language; and (e) the tool-supported automatic generation of robot plans for problems defined in the KANOA DSL.

4.1 QUARTET Mission Patterns

In this section, we present the project's contributions to QUARTET, a comprehensive catalogue for quantitative robotic mission specifications. This catalogue is accompanied by a domain-specific language (DSL), which is detailed in the preliminaries. Then, we present the semantics of QUARTET as a key contribution to this thesis. We show the automatic translation of the DSL written requirements into a probabilistic logic language supporting quantitative reasoning, such as "what is the level of the robot's battery at the end of the day" or "what is the probability of failing to retrieve a series of objects." Finally, we conclude with the evaluation of the catalogue's effectiveness and applicability.

4.1.1 Preliminaries

This section introduces the syntax for the specification of quantitative robotic mission patterns. These patterns, extracted from literature and expert input, follow a hybrid approach detailed in Figure 4.2. The bottom-up method extracts recurrent patterns from the literature, while the top-down approach proposes patterns based on domain experts' experiences. The extraction of the mission patterns and their syntax was conducted as part of a larger collaborative project [1] that this PhD research contributed to. Hence, these patterns are described as part of the preliminaries section. To enable programmatic reasoning about the quantitative robotic mission patterns, these are translated into temporal logic; the semantics and tool related to this translation are an original contribution of this thesis and elaborated upon in Section 4.1.2.

QUARTET DSL syntax

QUARTET [1] is a pattern-based Domain-Specific Language (DSL) for the specification of quantitative and qualitative robot missions. One of its key features is that the DSL

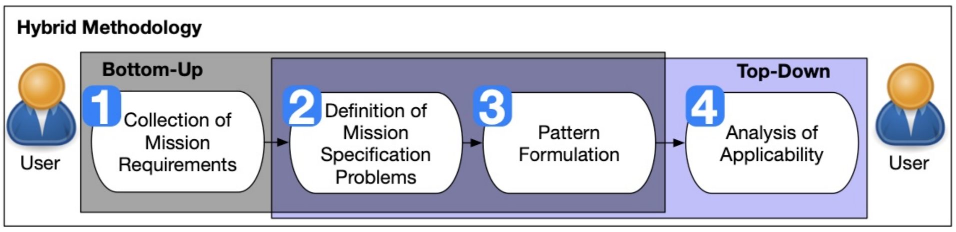

Hybrid Methodology
Bottom-Up
Top-Down
User
1 Collection of Mission Requirements
2 Definition of Mission Specification Problems
3 Pattern Formulation
4 Analysis of Applicability
User

Figure 4.2: Methodology used for the collection and pattern formulation of robot missions. Figure obtained from [1].

is constructed based on patterns extracted from both literature and research experts. The methodology for the collection and identification of the mission patterns follows a top-down and bottom-up hybrid methodology, as depicted in Figure 4.2. The bottom-up methodology collects and defines the robot missions and mission requirements from the literature, whilst the top-down methodology is used to collect users' data and case studies from domain experts to test the applicability of these patterns. QUARTET identifies 22 quantitative patterns shown in Figure 4.3, which are subdivided into two categories, the *elementary problems* and the *composite problems*.

Elementary problems are subdivided into *objectives*, *intervals* and *bounds*; the first addresses the optimisation of mission requirements, specifically aiming to either maximize or minimize certain quantities. The second defines a range within which a variable should be sustained throughout the mission, whereas the third limits a variable to either above or below a specified value.

Composite problems are subdivided into four types: performance and dependability, space, and time and resource. *Performance and dependability* refers to confidence levels of performing a mission (confidently), the reliability of completing a mission (reliability) and the maximisation of a metric that measures performance (accrue). *Time* incorporate deadlines (timeout, end), repetition of missions at every pre-specified interval of time (repeat), time to execute a mission relative to a second mission (proportionality), time to pause a mission (pause), and the execution of multiple actions simultaneously (simultaneously). Finally, *resources* refers to maintaining a measurement within a specified interval during the mission (preservation) or minimising its value (conservation). A detailed description of all problems is provided in Table 4.1. The first column shows the name of the pattern, the second column the description in natural language, and the third column the syntax in QUARTET DSL.

These *quantitative* robotic missions patterns (as shown in Figure 4.3) build upon a prior collection of 22 *qualitative* specification patterns compiled by [2], as illustrated previously in Figure 3.8a. A tool called PSALM [36] was developed for the translation of the patterns of this earlier repository into LTL and CTL formulae for their later verification through model checkers. The limitation of the initial mission repository and PSALM is

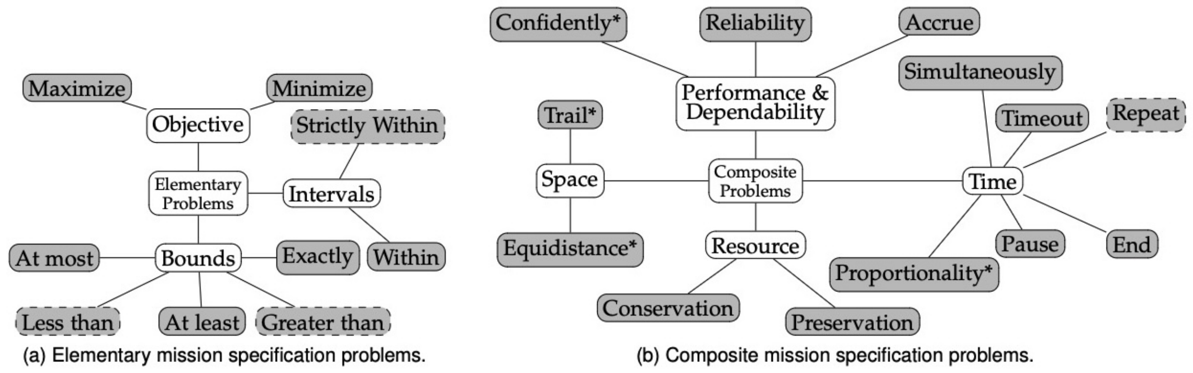


Figure 4.3: Elementary and composite QUARTET patterns for robotic mission specification. Shadowed nodes represent robotic mission specification problems and clear nodes denote categories of requirements. Nodes with solid and dashed borders represent the mission specification problems identified by following bottom-up and top-down procedures, respectively, as depicted in Figure 4.2. Patterns marked with a star '*' were not formalised within QUARTET.

that it does not support quantitative verification, meaning that the result only provides True or False feedback. To address this limitation, the QUARTET repository supplies an additional set of mission patterns that enable qualitative feedback in terms of probabilities and rewards. From this point onwards, we exclude the patterns of trail, equidistance, proportionality and confidence as they were not able to be formalised within QUARTET. For more details on these patterns, we refer the reader to [1].

Syntax. The QUARTET syntax is depicted in Figure 4.4. The grammar is divided into five categories (mission, patterns, elementary patterns, composite patterns, and conditions and locations). Alternatives are separated by the symbol |; optional items are enclosed in round brackets followed by a question mark (?); recursive items in round brackets followed by an asterisk ($*$); while keywords are shown in coloured font.

There are six terminals of the language: (1) `loc` represents a location, e.g., a room of the building or a physical position x, y, z; (2) `rob` indicates a robot identifier; (3) `condition` indicates a Boolean condition that can be either true or false; (4) $\texttt{act}_i$ represents the ith action, which is a Boolean variable representing an action achievable by the robot; (5) `m` represents the name of a quantitative measurement (i.e., *Energy*, referring to the energy consumption of a robot), and (6) $\texttt{v}_i$ represents a numerical value, $\texttt{v}_i \in \mathbb{R}$.

A robotic mission `miss` consists of the composition of missions through logic connectors **`and`** and **`or`**, the negation of a defined mission (**`not`** `miss`), a robot in charge of executing a pattern (`rob` **`shall`** `pat`), an elementary pattern (`e_qpat`), or a composite quantitative pattern (`c_qpat`).

Patterns (`pat`) consist of non-quantitative robotic specifications. Patterns are assigned to a robot by specifying a mission of the form "`rob` **`shall`** `pat`", where this mission can

Mission	miss	::=	miss and miss | miss or miss | not miss | rob shall pat | e_qpat | c_qpat
Patterns	pat	::=	visit (in sequence | in order | in strict order | fairly)? locs |
			patrol (in sequence | in order | in strict order | fairly)? locs |
			visit (more than | less than | exactly) n times loc |
			avoid (loc until cond | loc | loc after cond) |
			react (instantly | with a delay | promptly) to cond by (exec act | pat | reach loc) |
			counteract (instantly | with a delay) when reach loc by cond |
			wait in location loc until cond
Elementary Patterns	e_qpat	::=	(reward)? maximize m miss | minimize m miss | m at most v miss | m less than v miss |
			m at least v miss | m greater than v miss | m exactly v miss | m within v_1 and v_2 miss |
			m strictly within v_1 and v_2 miss
Composite Patterns	c_qpat	::=	conserve m while miss | preserve m within [v_1,v_2] while miss | pause v miss |
			timeout v miss | repeat miss every v | end miss exactly at v |
			execute rob actions act_1,act_2,... act_n | rob accrue m while miss |
			achieve miss with reliability m (greater | less) than v |
Condition	cond	::=	condition is true | act is ended | rob in loc
Locations	locs	::=	{loc (, loc)*}

* miss, $miss_1$, $miss_2$ are missions; v, v_1, v_2 are values; rob is a robot, m is the name of the quantitative measure.

Figure 4.4: The syntax of the DSL for the quantitative specification patterns for robotic missions. From [1].

be later used as part of the elementary patterns; for example (based on [1]):

> "*...after the opening, the robot* **r1** *shall visit the different parts of the hospital to record the number of employees that are present in each of the rooms. The robots have to minimize the energy required to perform this mission.*"

In QUARTET, elementary patterns are differentiated based on whether they refer to *probabilities* or *rewards* (aka costs). The previous example, "...minimize the energy...," requires a reward elementary pattern to reason about the energy cost. Thus, we can write this example in the QUARTET DSL by defining a mission identifier, `m1`[1]; the measure of `Energy`; an event specifying the opening of the hospital, `open`; the locations to visit, `room1, room` and `room3`; and the condition that must be satisfied when reaching these locations while recording the number of employees, `record`. Thus, the mission is defined in QUARTET language as:

```
m1: reward minimize Energy (
(r1 shall react instantly to open by visit room1, room2, room3)
and
(r1 shall counteract instantly when reach room1 by record) and
(r1 shall counteract instantly when reach room2 by record) and
(r1 shall counteract instantly when reach room3 by record)
)
```

Elementary (`e_qpat`) and composite (`c_qpat`) patterns are defined in the first and second half of Table 4.1, respectively. This table contains the intermediate step between the

[1]The identifier is added for readability purposes but not explicitly defined in the DSL syntax.

Table 4.1: Quantitative mission specification problem descriptions and their translation into the QUARTET DSL. From [1].

Problem	**Description**	**DSL**
Maximize	Maximize `m` while performing the mission `miss`.	`reward maximize m miss`
Minimize	Minimize `m` while performing the mission `miss`.	`reward minimize m miss`
At most	Keep `m` lower than or equal to `v` while performing `miss`.	`reward m at most v miss`
Less than	Keep `m` strictly lower than `v` while performing `miss`.	`reward m less than v miss`
At least	Keep `m` greater than or equal to `v` while performing `miss`.	`reward m at least v miss`
Greater than	Keep `m` strictly greater than `v` while performing `miss`.	`reward m greater than v miss`
Exactly	Keep `m` exactly `v` while performing `miss`.	`reward m exactly v miss`
Within	Keep `m` within the (closed) interval $[v_1, v_2]$ while performing `miss`.	`reward m within` v_1 `and` v_2 `miss`
Strictly Within	Keep `m` within the (open) interval (v_1, v_2) while performing `miss`.	`reward m strictly within` v_1 `and` v_2 `miss`
Conservation	Minimize the value of `m` performing `miss`.	`conserve m while miss`
Preservation	Keep the value of `m` within interval $[b_l, b_u]$ while performing `miss`.	`preserve m within` $[v_1, v_2]$ `while miss`
Pause	Pause the mission `miss` for `v` time instants. Then, resume it.	`pause v miss`
Timeout-deadline	Execute `miss`. Stop the the execution when the timeout `v` is reached.	`timeout v miss`
Repeat	Repeat the mission `miss` every `v` time units.	`repeat miss every v`
End	Terminate mission `miss` exactly at time `v`.	`end miss exactly_at v`
Simultaneously	Execute the actions $act_1, \dots, act_n$ simultaneously.	`execute rob actions` $act_1, \dots, act_n$
Accrue	Maximize the performance `m` while performing `miss`.	`rob accrue m while miss`
Reliably	Ensure that the measure `m` is higher/lower than the value `v`.	`achieve miss with reliability m` [...]

* `miss`, $miss_1$, $miss_2$ are missions; `v`, v_1, v_2 are values; `rob` is a robot, `m` is the name of the quantitative measure; [...] represents portions of the DSL of Figure 4.4 omitted for graphical reasons. Elementary patterns (*maximize* to *strictly within*) descriptions can refer to probabilistic requirements (instead of rewards) by removing the `reward` keyword from their DSL counterpart.

ontological representation of mission problems in Figure 4.3, and their DSL counterparts presented in Figure 4.4.

Finally, we define the two last categories, conditions (`cond`) and locations (`locs`). The latter refers to a set of locations, {`loc` (,`loc`)*}; while the former corresponds to Boolean events specifying that: a condition is happening (`condition is true`), an action has ended (`act is ended`), or a robot reached a certain location (`rob in loc`). A measurement `m` represents a relevant quantity within the mission specification. These measurements are included in the mission description to provide context for the quantities under consideration, but they are not part of QUARTET's formal semantics. In the previous example where the goal was to minimize energy consumption, `m` was defined as `Energy`.

4.1.2 QUARTET Semantics and Implementation

In this section, we introduce a key contribution of this thesis, namely the semantics of QUARTET for the translation of quantitative specification patterns to *Reward-augmented Probabilistic Computational Tree Logic* (RPCTL) formulae. An introduction to RPCTL is provided in the background Section 2.3.3. To conclude, we assess the applicability and exploitability of the QUARTET catalogue.

Table 4.2: Semantics of QUARTET for the translation of quantitative missions into RPCTL logic formulae.

Mission		$\tau(\texttt{miss1 and miss2}) = \tau(\texttt{miss1}) \land \tau(\texttt{miss2})$	$\tau(\texttt{miss1 or miss2}) = \tau(\texttt{miss1}) \lor \tau(\texttt{miss2})$
		$\tau(\texttt{not miss}) = \neg\tau(\texttt{miss})$	$\texttt{rob shall pat} = \tau(\texttt{pat}[r \leftarrow \texttt{rob}])$
Elementary Patterns	**Prob.**	$\tau(\texttt{maximize m miss}) = \mathcal{P}_{max=?}(\tau(\texttt{miss}))$	$\tau(\texttt{minimize m miss}) = \mathcal{P}_{min=?}(\tau(\texttt{miss}))$
		$\tau(\texttt{m at most v miss}) = \mathcal{P}_{\leq \texttt{v}}(\tau(\texttt{miss}))$	$\tau(\texttt{m less than v miss}) = \mathcal{P}_{< \texttt{v}}(\tau(\texttt{miss}))$
		$\tau(\texttt{m at least v miss}) = \mathcal{P}_{\geq \texttt{v}}(\tau(\texttt{miss}))$	$\tau(\texttt{m greater than v miss}) = \mathcal{P}_{> \texttt{v}}(\tau(\texttt{miss}))$
		$\tau(\texttt{m exactly v miss}) = \mathcal{P}_{\geq \texttt{v}}(\tau(\texttt{miss})) \land \mathcal{P}_{\leq \texttt{v}}(\tau(\texttt{miss}))$	
		$\tau(\texttt{m within v}_1 \texttt{ and v}_2 \texttt{ miss}) = \mathcal{P}_{\geq \texttt{v}_1}(\tau(\texttt{miss})) \land \mathcal{P}_{\leq \texttt{v}_2}(\tau(\texttt{miss}))$	
		$\tau(\texttt{m strictly within v}_1 \texttt{ and v}_2 \texttt{ miss}) = \mathcal{P}_{> \texttt{v}_1}(\tau(\texttt{miss})) \land \mathcal{P}_{< \texttt{v}_2}(\tau(\texttt{miss}))$	
	Rewards	$\tau(\texttt{reward maximize m miss}) = \mathcal{E}_{max=?}(\tau(\texttt{miss}))$	$\tau(\texttt{reward minimize m miss}) = \mathcal{E}_{min=?}(\tau(\texttt{miss}))$
		$\tau(\texttt{m reward at most v miss}) = \mathcal{E}_{[0,\texttt{v}]}(\tau(\texttt{miss}))$	$\tau(\texttt{m reward less than v miss}) = \mathcal{E}_{[0,\texttt{v})}(\tau(\texttt{miss}))$
		$\tau(\texttt{m reward at least v miss}) = \mathcal{E}_{[\texttt{v},\infty)}(\tau(\texttt{miss}))$	$\tau(\texttt{m reward greater than v miss}) = \mathcal{E}_{(\texttt{v},\infty)}(\tau(\texttt{miss}))$
		$\tau(\texttt{reward m exactly v miss}) = \mathcal{E}_{\geq \texttt{v}}(\tau(\texttt{miss})) \land \mathcal{E}_{\leq \texttt{v}}(\tau(\texttt{miss}))$	
		$\tau(\texttt{reward m within v}_1 \texttt{ and v}_2 \texttt{ miss}) = \mathcal{E}_{[\texttt{v}_1,\infty)}(\tau(\texttt{miss})) \land \mathcal{E}_{[0,\texttt{v}_2]}(\tau(\texttt{miss}))$	
		$\tau(\texttt{reward m strictly within v}_1 \texttt{ and v}_2 \texttt{ miss}) = \mathcal{E}_{(\texttt{v}_1,\infty)}(\tau(\texttt{miss})) \land \mathcal{E}_{[0,\texttt{v}_2)}(\tau(\texttt{miss}))$	
Composite Patterns		$\tau(\texttt{conserve m while miss}) = \mathcal{E}_{min=?}(\tau(\texttt{miss}))$	
		$\tau(\texttt{preserve m within } [\texttt{v}_1,\texttt{v}_2] \texttt{ while miss}) = \mathcal{E}_{[\texttt{v}_1,\texttt{v}_2]}(\tau(\texttt{miss}))$	
		$\tau(\texttt{pause v miss}) = \mathcal{G}^{[0,\texttt{v}]}\tau(\neg\texttt{miss}) \land (\mathcal{F}^{[\texttt{v}+1,\texttt{v}+1]}(\tau(\texttt{miss})))$	
		$\tau(\texttt{timeout v miss}) = \mathcal{G}^{[\texttt{v},\infty]}(\neg\tau(\texttt{miss}))$	
		$\tau(\texttt{repeat miss every v}) = \tau(\texttt{miss}) \land \mathcal{G}^{[0,\infty]}(\tau(\texttt{miss}) \rightarrow (\mathcal{G}^{[1,\texttt{v}-1]}(\neg\tau(\texttt{miss})) \land (\mathcal{F}^{[\texttt{v},\texttt{v}]}(\tau(\texttt{miss})))))$	
		$\tau(\texttt{end miss exactly at v}) = \mathcal{G}^{[0,\texttt{v})}(\tau(\texttt{miss})) \land \mathcal{G}^{[\texttt{v},\infty]}(\neg\tau(\texttt{miss}))$	
		$\tau(\texttt{time of miss}_1 \texttt{ proportional to miss}_2 \texttt{ by factor v})$=NA (Not Available in PRCTL)	
		$\tau(\texttt{execute rob actions act}_1\texttt{,act}_2\texttt{,}\ldots\texttt{, act}_n) = \mathcal{F}\ (\bigwedge_{i=1}^{n} \texttt{act}_i)$	
		$\tau(\texttt{r accrue m while miss}) = \mathcal{E}_{max=?}(\tau(\texttt{miss}))$	
		$\tau(\texttt{achieve miss with reliability m (greater} \mid \texttt{less) than v}) = \mathcal{E}_{[\texttt{v},\infty)}(\tau(\texttt{miss}))/\mathcal{E}_{[0,\texttt{v})}(\tau(\texttt{miss}))$	

Semantics

The semantics of QUARTET are depicted in Table 4.2. The table presents the semantics divided into *missions, elementary patterns* and *composite patterns.* Notice that the patterns (`pat`) from the QUARTET syntax in Table 4.4 are not defined. This is because `pat` are qualitative formulae with their translation into LTL and CTL logic presented in previous work [2]. We include the translation of `pat` in Appendix A for reference.

We define a function τ for the translation of QUARTET missions into RPCTL formulae. The first defined mapping function takes a mission `miss and miss`, and maps it to the conjunction of the translation of each part, $\tau(\texttt{miss1}) \land \tau(\texttt{miss2})$; similarly for the disjunction (`or`) and negation (`not`) variants. For a robot performing a non-quantitative pattern $\tau(\texttt{rob shall pat})$ the translation is defined as $(\tau(\texttt{pat}[r \leftarrow \texttt{rob}]))$, in which `pat` is the translation of some QUARTET pattern containing the terminal r, which in this case takes the value of `rob`. The further translation of each QUARTET pattern, defined as CTL and LTL formulae, is described in Appendix A.

Elementary patterns enable the quantitative verification of robot mission characteristics. Their quantitative values can be in the form of a reward/cost or a probability that is returned through the verification of RPCTL formulae. Hence, the elementary patterns are divided into two categories: *prob*abilities and *rewards.* The operands $\mathcal{P}_{max=?}$ and $\mathcal{P}_{min=?}$ are RPCTL structures which denote the maximum and minimum probabilities of a RPCTL property, respectively. Similarly, the operators $\mathcal{E}_{max=?}$ and $\mathcal{E}_{min=?}$ denote the maximum and minimum expected reward. In the case of the rewards formulae, they rely heavily on the model as the reward structure must describe the changes in the measure `m` or value `v` indicated as part of the formula. For example, for the elementary pattern maximising the measure `m`, where `m` is a probability value of completing mission `miss`, the mapping function $\tau($**`maximize`** `m miss`$)$ returns the formulae $\mathcal{P}_{max=?}(\tau(\texttt{miss}))$. On the other hand, if a pattern refers to some other measure, e.g. energy consumption, the mapping function corresponds to $\mathcal{E}_{min=?}(\tau(\texttt{miss}))$.

In the elementary patterns mapped to reward formulae, the structure $\mathcal{E}_J(\phi)$ corresponds to the RPCTL reward operator. If J is an interval, the formula corresponds to the expected reward on the interval J, where J is $[0, \texttt{v}]$, $[0, \texttt{v})$, $[\texttt{v}, \infty)$ or $(\texttt{v}, \infty)$, depending on the pattern to be translated. If J is max =? or min =?, the resulting formula corresponds to the maximum or minimum expected rewards. Finally, if J is $\bowtie v$, where $\bowtie = \{\leqslant, \geqslant, <, >\}$ the resulting formula corresponds to the bounding RPCTL.

The **composite patterns** also consider reward and probabilities formulae. However, these quantities are treated as metrics that assess the resource, performance and dependability of the mission. For example, the *conservation* pattern translation $\tau($**`conserve`** `m` **`while`** `miss`$)$ calculates the minimum reward $\mathcal{E}_{min=?}$ that the user can evaluate assessing if a measure (e.g., the battery energy of a robot) is conserved (or not) above a threshold throughout the mission. The *preservation* pattern $\tau($**`preserve`** `m` **`within`** [v1, v2] **`while`** `miss`$)$ translation maintains the rewards within the interval `[v1,v2]` through the operator $\mathcal{E}_J$.

The *Pause* pattern translation specifies an interval $[0, \texttt{v}]$ when a mission `miss` is not executed (i.e., $\mathcal{G}^{[0,\texttt{v}]}\tau(\neg\texttt{miss})$ holds), with the execution of the mission guaranteed at time instant $\texttt{v} + 1$ (i.e., $\mathcal{F}^{[\texttt{v}+1,\texttt{v}+1]}(\tau(\texttt{miss})))$ holds). The *Timeout* pattern translation specifies an interval $[\texttt{v}, \infty]$ in which the mission is not executed (i.e., $\mathcal{G}^{[\texttt{v},\infty]}(\neg\tau(\texttt{miss}))$ holds). The pattern translation of *repeat* specifies that the formula must hold initially ($\tau(\texttt{miss})$), and requires its repetition such that whenever the mission happens at a time step, then it must not happen in the following `v-1` time steps, and must repeat only after `v` time steps.

The *End* pattern translation specifies that a mission `miss` is in execution from the beginning and up to a time instant `v` (i.e., $\mathcal{G}^{[0,\texttt{v}]}(\tau(\texttt{miss}))$ holds), and its execution stops at time `v` (i.e.,$\mathcal{G}^{[\texttt{v},\infty]}(\neg\tau(\texttt{miss}))$ holds). Finally, the translation for the *Simultaneously* pattern specifies that eventually all the actions are performed at the same time instant.

Notice that the translation proposed for the patterns belonging to the "Time" category (*simultaneously, timeout, repeat, end* and *pause*) do not follow the RPCTL syntax (i.e., the temporal formula is not preceded by the $\mathcal{P}$ or $\mathcal{E}$ operator). Therefore, to ensure that our translation generates well-formed RPCTL formulae, we **constrain the patterns belonging to the "Time" category to be used only within elementary patterns**.

The translation for the *Accrue* pattern relies on the operator $\mathcal{E}_{max=?}$ that enables maximising a reward measure while performing a mission `miss`. Similarly, the translation for the *Reliability* pattern relies on the operator $\mathcal{E}_J$ where the interval J is set to $(\mathtt{v}, \infty)$ or $[0, \mathtt{v})$ depending on whether the **`less than`** or **`greater than`** option is used.

Well-formedness conditions

To have a well-defined meaning in the RPCTL semantics accepted in the PRISM language, a robotic mission specification written in our QUARTET language must satisfy multiple well-formedness conditions. Mission requirements must be well-typed and well-scoped. Locations and conditions must be defined before they are used within elementary and composite patterns. The generated requirements must comply with the RPCTL syntax accepted by the PRISM model checker (see Sections 2.3.3 and 2.3.5). The following well-formedness conditions apply to missions in QUARTET.

F1. *The elementary patterns* ***maximise*** *and* ***minimise*** *must not be preceded nor followed by any of the boolean operators accepted (**`and`**, **`or`** and **`not`**) as these patterns result in a numerical value rather than a Boolean.*

F2. *Elementary patterns cannot be composed by missions `miss` that refer to other elementary pattern. This is due to their semantics translating them as probabilistic $\mathcal{P}$ and reward $\mathcal{E}$ formulae—RPCTL does not allow to have two consecutive $\mathcal{P}$'s or $\mathcal{E}$'s, nor their combination, $\mathcal{P}_J(\mathcal{E}_J(\mathtt{miss}))$ or $\mathcal{E}_{\mathcal{J}}(\mathcal{P}_{\mathcal{J}}(\mathtt{miss}))$.* This also apply to composite patterns **conservation**, **preservation**, **accrue**, **reliability** and **confidently**.

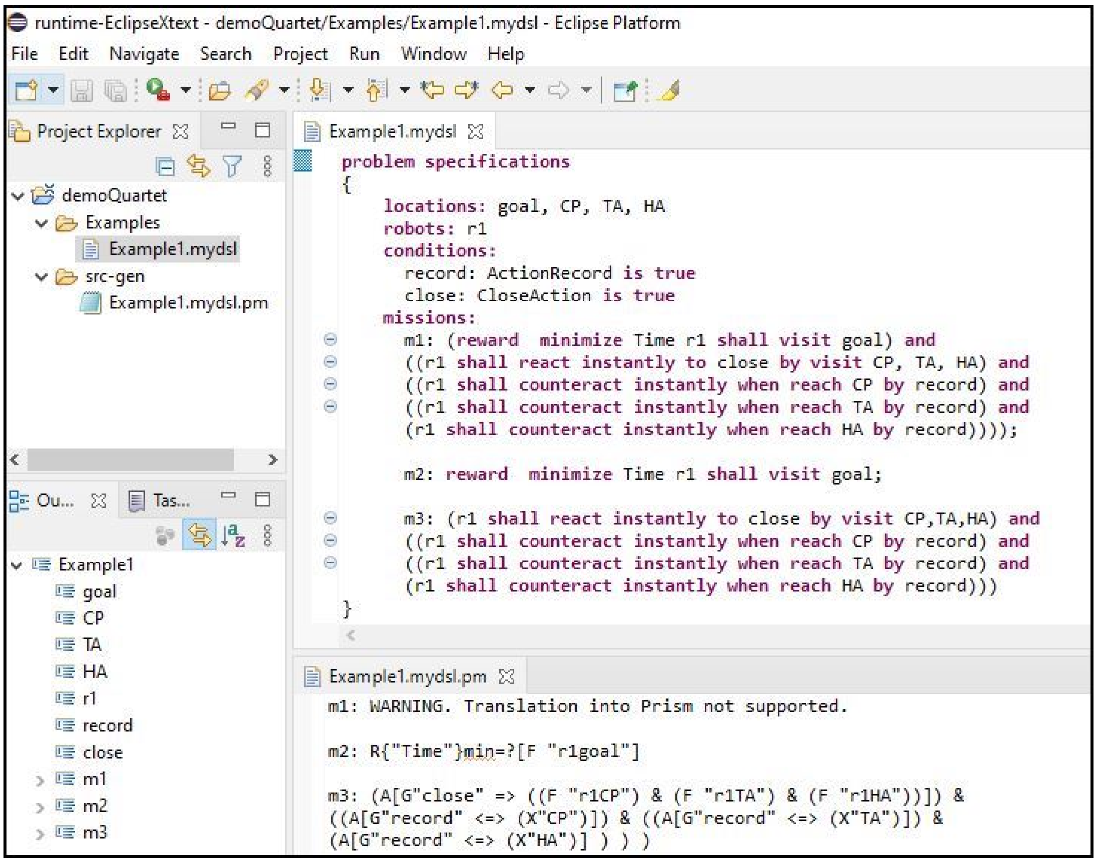
F3. *Nested probabilities are not allowed. Nesting can occur as explained in* ***F2*** *or by having elementary patterns joint by logic operators inside other elementary patterns, as in $\mathcal{P}_J(\mathcal{P}_J(\mathtt{miss}) \wedge \mathcal{P}_J(\mathtt{miss}))$. These types of formulae do not comply with the RPCTL syntax.*

F4. The **repeat** composite pattern must comply with **F2** and **F3**; e.g., the mission included as part of this pattern must not result in a requirement with nested $\mathcal{P}$'s.

If these constraints are not satisfied, QUARTET generates a warning indicating that the mission specification cannot be generated in the PRISM language. The following section shows an example of the automated process.

QUARTET Tool

Another contribution of the thesis is the implementation of a proof-of-concept tool and user interface (GUI) for the automatic translation of missions specified in QUARTET DSL to RPCTL formulae. The generated formulae are written in the PRISM model checker's property specification language [71] and can be used for the verification of PRISM-encoded Markov models of an MRS behaviour. To ensure that our tool generates mission specifications expressed in the PRISM property specification language, we constrained the



```
problem specifications
{
    locations: goal, CP, TA, HA
    robots: r1
    conditions:
      record: ActionRecord is true
      close: CloseAction is true
    missions:
      m1: (reward  minimize Time r1 shall visit goal) and
      ((r1 shall react instantly to close by visit CP, TA, HA) and
      ((r1 shall counteract instantly when reach CP by record) and
      ((r1 shall counteract instantly when reach TA by record) and
      (r1 shall counteract instantly when reach HA by record))));

      m2: reward  minimize Time r1 shall visit goal;

      m3: (r1 shall react instantly to close by visit CP,TA,HA) and
      ((r1 shall counteract instantly when reach CP by record) and
      ((r1 shall counteract instantly when reach TA by record) and
      (r1 shall counteract instantly when reach HA by record)))
}
```

```
m1: WARNING. Translation into Prism not supported.

m2: R{"Time"}min=?[F "r1goal"]

m3: (A[G"close" => ((F "r1CP") & (F "r1TA") & (F "r1HA"))]) &
((A[G"record" <=> (X"CP")]) & ((A[G"record" <=> (X"TA")]) &
(A[G"record" <=> (X"HA")] ) ) )
```

Figure 4.5: Screenshot of the QUARTET tool showing a portion of the mission requirement of mission `m1` introduced in Section 4.1.1. The problem specifications define the locations (goal, CP, TA and HA), robots (r1) and conditions (record and close). Missions `m2` and `m3` are derived from m1 as quantitative and qualitative formulae, respectively, translated automatically into PRISM (bottom part). Mission m1 cannot be translated directly into PRISM as it joins (by a logical "and") a number (from `m2`) and a Boolean (from `m3`), a type of property conjunction which PRISM does not support.

DSL semantics to the well-formedness conditions presented before. These prohibit nested probabilities, accept only LTL properties for the reward and probability operators, and prohibit the definition of specifications that lead to the conjunction of quantitative and non-quantitative PRISM formulae since such formulae cannot be processed by PRISM. The tool is publicly available at [146] as an Eclipse plugin.

Figure 4.5 shows the GUI interface as well as *Example1.mydsl* file with the description of three missions, `m1,m2` and `m3`. Mission `m1` is already described in Section 4.1.1. Mission `m2` is a quantitative formula derived from `m1`. Mission `m3` is a qualitative formula also derived from `m1`. The translation of the missions depends on whether or not the PRISM properties language supports their translation. For missions (`m1,m2,m3`), the tool output is depicted at the bottom of this Figure 4.5 and described in the next paragraphs.

The QUARTET tool automatically translates QUARTET mission requirements into

RPCTL formulae according to the semantics specified in Table 4.2. The translation uses the open-source software framework Xtext [147] for the development of domain-specific languages. Xtext supports the Java-inspired high-level language Xtend as a general-purpose language that accelerates the process of coding the semantics of a language given its flexibility and expressiveness.

We selected the property specification language PRISM [71] for the specification of QUARTET-generated formulae as it supports the entire RPCTL logic, it has been widely and successfully used within the formal methods community and it supports the LTL and CTL formulae necessary for the translation of QUARTET non-quantitative missions. Also, it provides an open-source tool available online for the specification and verification of these logic formulae.

To ensure that the QUARTET tool generates well-defined PRISM properties in its supported RPCTL, LTL and CTL logics, we constrained the generated output to (a) prohibit nested probabilities, (b) accept only LTL properties as arguments for the reward and probability operators, and (c) prohibit the definition of specifications that lead to the conjunction/disjunction of quantitative and non-quantitative PRISM formulae, since such formulae cannot be processed by the PRISM model checker.

Given constraint (a), a formula of the form $\mathcal{P}_{max=?}(\mathcal{P}_{min=?}\sigma)$ is not allowed. Constraint (b) forces the formulae used within the reward and probability operators to be LTL formulae, such as $\phi_1\mathcal{U}\phi_2$, i.e., it does not enable the exploitation of the values assumed by J and N within formulae of the form $\phi_1\mathcal{U}_J^N\phi_2$. Finally, an example of a mission that violates constraint (c) is mission `m1` from Figure 4.5, in which a quantitative formula and a non-quantitative formula are joined by the conjunction operator. If one of the constraints is violated, the QUARTET tool will return a *warning* as in the case of mission `m1` result depicted at the bottom of Figure 4.5.

4.1.3 Evaluation

To evaluate the coverage and exploitability of the QUARTET catalogue of robot mission specifications, we considered the following research questions.

- **RQ1 (Applicability of the translation).** *In how many cases does new identified requirements were able to be formalised using the intended semantics?* We assess how many requirements which are agnostic to the generation of QUARTET can be translated into RPCTL using our developed tool.

- **RQ2 (Exploitability)**. *How can the translation of robotic mission requirements using the QUARTET semantics be used in practice?* We assess the usefulness of the QUARTET mission formalisation, for example, for the verification of models.

RQ1 experimental tasks (Applicability). In order to evaluate the applicability of the QUARTET patterns, we use 21 mission requirements previously collected as part

Table 4.3: Number of times each of the QUARTET patterns was used to express a mission requirement or part of this from our dataset.

Pattern	#N	Pattern	#N	Pattern	#N
Maximize	5	*Within*	2	*Timeout*	5
Minimize	6	*Strictly Within*	1	*Reliability*	4
At most	3	*Conservation*	5	*Simultaneously*	1
Less than	-	*Preservation*	4	*Accrue*	3
At least	3	*Pause*	-	*Exactly*	2
Greater than	-	*Repeat*	1	*End*	-

of the hybrid methodology described in Figure 4.2. These patterns were not considered in the formulation of any of the QUARTET patterns.[2] We attempted to express these collected papers in the proposed DSL and evaluate if their translation is possible.

RQ1 results and discussion. In order to evaluate the applicability of our approach, we first assess the number of missions collected from the literature that can be defined in QUARTET's domain-specific language. Then, we assess how many of these requirements can be translated into RPCTL formulae written in the PRISM language. From the 21 requirements collected agnostic to the QUARTET catalogue, we were able to completely express 20 of them ($\tilde{9}5\%$), and to partially express the last one ($\tilde{5}\%$). This coverage is acceptable for practical applications since the patterns are (by definition) not exhaustive. The requirement that was not able to be formalised requires "adapt[ing] the velocity profile of the robot, according to the wireless channel measurements" [148]. This requirement requires modifying the velocity as a function of the wireless communication measurements. Functions describing the dynamics between measurements are not currently supported in QUARTET.

Recall that to express one mission requirement, the DSL allows more than one pattern to be used. Table 4.3 reports the number of times each of the QUARTET patterns was used to express a (part of) a mission requirement from our dataset. Continuing, we assess how many of these requirements were able to be formalised using the QUARTET tool expressing them in the PRISM language. Our results show that our translation provides reasonable applicability by translating more than 50% of the requirements. We intend this to grow as the dynamic repository is increased as (a) more expressive logic is defined by the research community, and (b) efficient tools that support and scale to more complex logic formulae are proposed.

RQ2 experimental tasks (Exploitability). In order to assess the exploitability of the QUARTET requirements in different robotic missions, one would need to have access to the missions themselves specified in a model compatible with the RPCTL formalism. For example, PRISM supports the verification of RPCTL formulae for Markov models

[2]The collection of these papers is not part of the contributions of these papers. We refer the reader to [1] for more details.

such as DTMCs and MPDs [73]. However, manually deriving these models would introduce significant threads to the validity to our results. For this reason, we opted to collect mission requirements from the literature that were accompanied by a PRISM specification already proposed by the respective authors. Then, we check if the mission requirements in each of the collected could be express using QUARTET's DSL and if the translation was possible using the accompanying tool. We verified whether QUARTET generated the PRISM mission specification defined by the authors. If this was the case, we considered the results reported in the publication and discussed how the mission specifications were employed by the authors.

The dataset for RQ2 consists of 16 requirements collected from the PRISM Case studies webpage [149] (2 requirements) and from the literature (14 requirements). All studies from which the requirements were obtained contain both the mission specification and its formalisation into the PRISM language. Requirements from research question RQ1 were not reused as they do not contain the PRISM counterpart.

RQ2 results and discussion. The 16 mission requirements considered with PRISM formalisms were able to be defined through our QUARTET DSL, demonstrating the flexibility and expressiveness of the DSL in describing robotic missions. Moreover, the original collected requirements defined in the PRISM language matched the ones synthesised by our QUARTET tool, further validating the applicability of our catalogue. This data is publicly available in [146].

Upon analysis, we identified two key applications within the research studies. In 75% of the cases, the specifications were used for the synthesis problem (e.g. [150, 151]), highlighting the predominant use of the specifications in generating decision-making strategies for robotic systems. Meanwhile, for 25% of the cases these specifications were considered for model checking (e.g. [152, 153]), emphasising the importance of verifying and validating robotic missions.

These findings indicate that our robotic mission specification catalogue can serve dual purposes in future applications. It can be employed both for policy synthesis, enabling the development of adaptive and responsive robotic behaviours, and for model checking ensuring, for example, the reliability and safety of the designed missions.

Discussion and limitations

QUARTET provides a wide range of quantitative mission patterns extracted from extensive research on robotic missions found in the research literature. It presents a comprehensive language for the specification of quantitative robotic patterns and missions and the translation of these into logic formulae. However, there exist some limitations in using QUARTET. First, the translation into logic formulae imposes constraints on the expressiveness of missions. For example, it is not possible to express a qualitative and a quantitative formula joined by the *and* logic operator.

As briefly described at the beginning of this chapter, another important limitation is that whilst QUARTET is focused on the description of robotic problems, i.e., elementary and composite problems as depicted in Figure 4.3, QUARTET does not support the description of locations, robots, tasks, etc. associated with robotic missions in greater detail. Consider for instance the mission **`minimize`** `Time` `r1` **`shall visit`** `goal`, in which the robot identifier $r1$ is obtained from a set of robot identifiers given in advance. QUARTET does not allow the description of robot capabilities for the robots with these identifiers, nor of the tasks that they can perform with those capabilities and the probability that task X can be completed successfully by robot Y. Another example of key information that cannot be captured in QUARTET is associated with the locations where the mission tasks are to be performed, which can be specified as location identifiers but cannot be mapped to physical coordinates from the real world. A mechanism to specify such information that cannot be encoded in QUARTET is presented in the following section.

4.2 KANOA Mission Specifications

This section presents the KANOA approach for the description of robotic problem specifications. KANOA uses a DSL defined in a *separation-of-concerns* fashion where different models capture different aspects of the robotic problem: (a) the world model, (b) the task specifications, (c) the robots' specifications, and (d) the mission and its properties, constraints, and objectives. The KANOA DSL is intended to be a user-friendly language that captures more information than QUARTET. However, QUARTET mission patterns can be added into KANOA's specifications. As QUARTET provides an extensive catalogue of patterns, from now on we only continue with the formulae ***minimise*** and ***maximise*** as a running example on how to include QUARTET patterns into KANOA. These are used in the definition of **objectives** including *minimising the idling time* i.e., the time during which the robots are in stand-by, *minimising the robots' travelling costs* and *maximising the probability of succeeding* with the mission, and as such constitute a general example of core QUARTET functionality which KANOA must support.

We also present the formalisation of the KANOA DSL using Z notation (see Section 2.1 for a background on Z notation). We use the DSL for the modelling of the system and Z notation for the formal definition of the different parts of the problem (world model, tasks, robots and the mission), as well as to ensure the completeness and compatibility between these parts. In particular, using Z notation enables the specification of key constraints that need to be satisfied by a valid problem specification defined in the KANOA DSL. As an example, a robot is characterised by a (non-empty) set of capabilities; each capability, in turn, corresponds to an atomic task identifier, which is associated with a location identifier, and so forth. Furthermore, the KANOA task allocation (detailed in the next chapter) is carried out using the Alloy Analyzer, whose input can is obtained from this Z specification.

4.2.1 KANOA Syntax and Problem Definition

In this section, we define the KANOA DSL syntax. We also describe the relations between the elements of an MRS mission, ant their scopes and mathematical constraints by supporting the description of the DSL with the Z notation. Hence, after introducing the KANOA language, each section of the grammar is also described in terms of its Z-notation counterpart.

KANOA's language syntax for the specification of MRS problems is depicted in Table 4.4, and is divided into five parts (world model, tasks model, robots model, mission and overall problem specification). In this table, alternatives are separated by the symbol |; possible recursive items are encircled in round brackets and followed by an asterisk *; items that can appear in any order are listed between curly brackets { }; while keywords are depicted in **bold** font. There are two types of terminals in the language: (a) identi-

Table 4.4: The syntax of KANOA's DSL for the specification of MRS missions consists of four parts (world model, tasks model, robots model and mission specification) that are combined into a problem specification.

Name	Term	Syntax
World Model	world	::= loc (,loc)*(,path)*
Location	loc	::= locID :(x v$_1$, y v$_2$)
Path	path	::= distance locID$_1$ to locID$_2$ is v (has success rate:v$_{[0,100]}$ %)?
Task Model	task	::= {(a_task | c_task)* }
Atomic Task	a_task	::= atID : v$_1$ robots needed at location locID (, v$_2$ retries allowed)?
Compound Task	c_task	::= ctID : subtasks [{ (ctID | atID) ((,ctID | ,atID))* }] (constraint: ordered | constraint:consecutive)?
Robots Model	robot	::= (rob)*
Robot	rob	::= robID: at initial position locID has velocity v with capabilities (capab (, capab)*)
Capability	capab	::= atID - required time: v, success rate:v$_{[0,100]}$ %
Mission	miss	::= {m_task(, m_task)*} objectives:{obj(, obj)*} (constraints:{con(, con)*})? (parameters: time: v)?
Mission task	m_task	::= mtID: (compound task ctID | atomic task atID) (at locID)?
Objectives	obj	::= (minimiseIdle | minimiseTravel | maximiseSuccess)
Constraints	con	::= rateSucc | spaceXY | taskTime | allocateT | closest | maxTasks
Success rate	rateSucc	::= rate of success greater than v$_{[0,100]}$
Set coordinate	spaceXY	::= (robot robID | all robots) work in (x | y) (lower | greater) than v
Task time	taskTime	::= (atomic task atID | compound task ctID | mission task mtID) has to (start after | end before) time: v
Allocate task	allocateT	::= allocate (atomic task atID | compound task ctID | mission task mtID) to (robot robID | single robot)
Closest task	closest	::= allocate (all tasks | atomic task atID | compound task ctID | mission task mtID) to closest robot
Limit n tasks	maxTasks	::= limit max number of tasks (in robot robID | per robot) to v
Allocations	nAlloc	::= v allocations
Time Available	time	::= time available: v
Problem Spec.	system	::= ProblemSpecification{ WorldModel: world TaskModel:task RobotsModel:robot Mission:miss }

* v's are values; v$_{[0,100]}$ is a value between 0 and 100; v$_{bool}$ is a Boolean value; and locID,robID,atID,ctID,mtID are identifiers of locations, robots, atomic-, compound- and mission-tasks, respectively.

fiers for the locations (locID), the atomic (atID) and composite (ctID) tasks, the robots (robID) and the mission tasks (mtID); and (b) values, which can be integers $v, v_n \in \mathbb{N}$, probabilities $v_{[0,1]} \in [0,1]$ and Booleans $v_{bool} \in \{true, false\}$.

4.2.2 World Model

Syntax. Multi-robot systems (MRS) involve the coordinated motion of a group of robots within a physical space, which is specified by a *world model* that define the area where these robots must perform various tasks. The world model is composed of locations loc and paths path, where:

- a location (locID:(x v$_1$, y v$_2$)) is defined by an identifier locID and the (x,y) coordinates of that location in a 2D discrete space, $(\mathtt{v}_1,\mathtt{v}_2) \in \mathbb{N} \times \mathbb{N}$.

- a path (**distance** `locID`$_1$ **to** `locID`$_2$ **is** `v`) is defined as the distance $\mathtt{v} \in \mathbb{N}$ between two locations with identifiers `locID`$_1$,`locID`$_2$ $\in$ `LocationID`, where `LocationID` is the set of all location identifiers.

Every path has an optional parameter specifying the success rate of travelling between `locID`$_1$ and `locID`$_2$ (**has success rate:** $\mathtt{v}_{0,100}$**%**), where $\mathtt{v}_{0,100} \in [0, 100]$. Some examples that characterise a path with a low rate of transitioning are a path that is narrow—thus increasing the collision of the robots, very long, or that has an uneven terrain. Notice that not all paths may be specified; only those paths that robots can traverse are declared.

Formal Specification. In the Z notation, we define the set of discrete location identifiers as,

$$[LocationID]$$

Each location is then formally defined by its (x,y) coordinates and its location identifier *id*

Location

$$id : LocationID$$
$$x, y : \mathbb{N}$$

where

$$\forall\, l1, l2 : Location \bullet l1.id = l2.id \Leftrightarrow l1 = l2$$

The world model is defined as a set of these locations, and two (partial) functions, *dist* and *succRate*, to describe the feasible paths between locations and the rate of succeess if attempting to navigate these paths, respectively.

World

$$locations : \mathbb{P}\, Location$$
$$dist : LocationID \times LocationID \rightarrow\!\!\!\!\!+ \mathbb{N}$$
$$succRate : LocationID \times LocationID \rightarrow\!\!\!\!\!+ \mathbb{N}$$

$$\forall\, l1, l2 : Location \bullet$$
$$\quad (l1.id, l2.id) \in \operatorname{dom} dist \Rightarrow l1 \in locations \land l2 \in locations$$

$$\forall\, l1id, l2id : LocationID \bullet$$
$$\quad ((l1id, l2id) \in \operatorname{dom} dist \land$$
$$\quad dist(l1id, l2id) = 0) \Leftrightarrow l1id = l2id$$

$$\operatorname{dom} succRate \subseteq \operatorname{dom} dist$$

$$\forall (l1id, l2id) \in \operatorname{dom} succRate \bullet 0 < succRate(l1id, l2id) \leq 100$$

The first two predicates of this Z schema indicate that all location identifiers appearing in the paths are specified beforehand as locations, and that every path has non-zero distance except if its start and end are the same location. The last two predicates state the existence of a rate of success (*succRate*) for some of the paths, and its possible range of values, respectively.

4.2.3 Task Model

Syntax. Robotic missions involve the execution of tasks that have unique identifiers. These tasks can be *atomic* (indivisible) tasks, or *compound* tasks formed of multiple subtasks.

The task model is a set composed of multiple atomic and compound tasks. Atomic tasks (`atID: v`$_1$ **`robots needed at location`** `locID`) are defined by an identifier `atID`, the number of robots needed `v`$_1 \in \mathbb{N}$, and a location identifier `locID` where the task is to be done. Each atomic task comes with an optional parameter specifying the permissible number of retries for the robot(s) in the event that the task is not successfully completed on the initial attempt(s) (**`,`** `v`$_2$ **`retries allowed`**), where `v`$_2 \in \mathbb{N}$.

Compound tasks (`ctID` **`: subtasks [`** { (`ctID | atID`) ((**`,`**`ctID |` **`,`**`atID`))* } (**`constraint: ordered`** | **`constraint: consecutive`**)?) are defined by an identifier `ctID` and a list of subtasks (compound or atomic tasks). The optional parameter **`constraint:ordered`** means that the subtasks must be completed in the specific order (but not necessarily without the robots also doing other tasks or pausing between them). The optional parameter **`constraint:consecutive`** means that the subtasks must be completed immediately one after the other, i.e., in a consecutive task with N subtasks, the time on which its subtask $i-1$ is completed is the beginning of its i-th subtask, for all $i = 2, 3, \ldots, N$.

Formal Specification. In the Z notation, we formally define the set of all possible task identifiers as

$$[\mathit{TaskID}]$$

We distinguish between the sets of identifiers for atomic tasks(*ATaskID*) and compound tasks (*CTaskID*):

$$\begin{array}{|l}
\mathit{ATaskID} : \mathbb{P}\, \mathit{TaskID} \\
\mathit{CTaskID} : \mathbb{P}\, \mathit{TaskID} \\
\hline
\mathit{ATaskID} \cup \mathit{CTaskID} = \mathit{TaskID} \\
\mathit{ATaskID} \cap \mathit{CTaskID} = \emptyset
\end{array}$$

An atomic task comprises an identifier *id*, specifies the number of robots *nRobots* required to perform the task, the allowed number of retries *nRetry* in case of failure, and a location *locationID*:

$$\begin{array}{|l}
\textit{AtomicTask} \\ \hline
id : ATaskID \\
nRobots : \mathbb{N}_1 \\
nRetry : \mathbb{N}_1 \\
locationID : LocationID \\ \hline
\end{array}$$

Atomic tasks whose execution requires several robots are termed *joint tasks*.

Compound tasks are formed by a sequence of subtasks that can be performed in no specific order, in a specific order (without forcing them to be consecutive), or consecutively and ordered (start each subtask that is not the first one exactly when the previous subtask finishes):

$$CON ::= none \mid ordered \mid consecutive$$

The subtasks of a compound task can be atomic or other compound tasks. Hence, a compound task is defined by an identifier *id*, its subtasks, and an ordering constraint value, *constraint*.

$$\begin{array}{|l}
\textit{CompoundTask} \\ \hline
id : CTaskID \\
subtasks : \mathrm{seq}\ TaskID \\
constraint : CON \\ \hline
\end{array}$$

Compound tasks are hierarchical tree-like structures. Therefore, we prevent loops by recursively forbidding specifying a composite task within its subtasks. We define the *descendants* as a well-defined relation between a compound task identifier and a set of *reachable* task identifiers, i.e., its subtasks and the descendants of its (compound) subtasks,

$$\begin{array}{|l}
descendants : CTaskID \rightarrow \mathbb{P}\ TaskID \\ \hline
\forall\, c : CompoundTask \bullet descendants\,(c.id) = \mathrm{ran}\ c.subtasks\ \cup \\
\qquad \bigcup\{subCT : CompoundTask \mid \\
\qquad subCT.id \in \mathrm{ran}\ c.subtasks \bullet descendants(subCT.id)\}
\end{array}$$

and impose the constraint

$$\begin{array}{|l}
\forall\, c : CompoundTask \bullet c.id \notin descendants\,(c.id)
\end{array}$$

Hence, an organisation that uses robotic teams to carry out missions will define their MRS missions based on a *TaskSet* comprising finite sets of atomic and compound tasks:

TaskSet

$atomicTasks : \mathbb{P}\, AtomicTask$

$compoundTasks : \mathbb{P}\, CompoundTask$

$\forall\, a1, a2 : atomicTasks \bullet a1.id = a2.id \Leftrightarrow a1 = a2$

$\forall\, c1, c2 : compoundTasks \bullet c1.id = c2.id \Leftrightarrow c1 = c2$

$\forall\, c : compoundTasks \bullet$
$\quad descendents\,(c.id) \subseteq \{at : atomicTasks \bullet at.id\} \cup$
$\qquad \{ct : compoundTasks \bullet ct.id\}$

4.2.4 Robot Model

Syntax. The robot model comprises the set of all robots available for the mission. Each robot is defined by an identifier (`robID`), the identifier `locID` for the initial location of the robot at the start of the mission, the average velocity of the robot `v`$\in \mathbb{N}$, and a list of capabilities describing the tasks that the robot can perform (`capab` (, `capab`)*).

A robot capability (`capab`) associates a robot with an atomic task identified by `atID`. It also contains information specifying how the robot performs for this task in terms of two metrics: the completion time (*time*) and the rate of succeeding with the mission (*succRate*). The probability that the robot completes the task successfully is obtained through dividing the success rate by 100.

The probability of completing of tasks can differ from robot to robot (even for robots of the same type) due to multiple factors. For example, if a robot $r1$ is capable of cleaning the floor, aging can impact the actuator used to clean the floor. Hence, robot $r1$ has a lower probability of completing successfully this task than a newer robot of the same type. Another example is that of a robot that is bigger, may get stuck from time to time while trying to clean a room with multiple beds or objects, which also impacts the probability of completing this task.

Formal Specification. A robot is referenced by a unique identifier of type

$[RobotID]$

Each robot has a set of *capabilities* consisting of the application-related atomic tasks that the robot can carry out. Each capability states that the robot can accomplish an atomic *task* in a mean *time*, and with probability of success *succRate*/100,

Capability

$task : ATaskID$

$time : \mathbb{N}$

$succRate : \mathbb{N}$

$0 < succRate \leq 100$

Finally, a robot is defined by its identifier, an initial location *initLocation*, its nominal velocity when in motion, and a set of capabilities,

Robot

$id : RobotID$

$initLocation : LocationID$

$velocity : \mathbb{N}$

$capabilities : \mathbb{P}_1\ Capability$

As explained earlier, times and rates of success can vary between homogeneous robots due to ageing, manufacturing defects, etc.

4.2.5 Mission Specification

Syntax. Once the atomic and compound tasks are specified, the user must select which tasks are to be done by the robots and the associated constraint and objective requirements. The tasks that the robots must complete (atomic or compound) are named *mission tasks* and we informally define a **mission** as the set of **mission tasks**, a list of mission-related **parameters**, as well as the sets of **constraints** that must hold and **optimisation objectives** that must be pursued by the MRS mission (see Table 4.4).

Each **mission task** (`m_task`) consists on the task identifier (`mtID`), an atomic (`atID`) or compound (`ctID`) task identifier, and an optional location where the task is to be done (`locID`). The optional location is added for reusability of the tasks; for example, consider an atomic task $at1$ defined in the task model in a location $loc1$, if the same task is to be done in three different locations ($loc1, loc2, loc3$), the user can avoid defining the same task three times in the task model and instead, reuse $at1$ within the mission overriding the location to $loc2$ and $loc3$. Four scenarios are possible to assign the location of tasks depending on how the mission task is defined:

- *atomic task + location:* the atomic task must be completed in the location defined by the mission task;
- *atomic task + no location:* the atomic task must be completed in the location defined by the atomic task (`a_task`);
- *compound task + location:* all atomic tasks reachable from the subtasks in the compound task must be completed in the location defined by the mission task;
- *compound task + no location:* each atomic task reachable from the subtasks in the compound task must be completed in the location defined by each atomic task definition (`a_task`).

KANOA supports three types of mission **optimisation objectives**: `minimiseIdle`, `minimiseTravel` and `maximiseSuccess` (see Table 4.4). These are measures that are to be optimised when creating a plan for each of the robots that will contribute to accomplishing the mission. First, minimising the idle time `minimiseIdle` aims to reduce the total time during which each robot is not performing any activity due to waiting for other robots to complete a task. Second, `minimiseTravel`ling cost reduces the time a robot spends travelling between locations. Third, `maximiseSuccess` probability aims to increase the overall probability of completing the mission. We note that the three optimisation objectives supported by KANOA may be conflicting.

There are six types of mission constraints (denoted as `con` in Table 4.4):

- `rateSucc` $\texttt{v}_{[0,100]}$, which requires that the mission is achieved with a success rate of at least $\texttt{v}_{[0,100]}$.

- `spaceXY` constrains the 2D space within which a robot can perform tasks. It requires the robot identifier (`robID`), the coordinate to constrain (`x` or `y`), the type of constraint (`lower than` or `greater than`) and the limit in space units $\texttt{v} \in \mathbb{N}$.

- `taskTime` constrains the time $\texttt{v} \in \mathbb{N}$ when a task (with identifier `atID`, `ctID` or `mtID`) is to be started after or completed by (`start after` or `end before`).

- `allocateT` enforces the allocation of a task (`atID`, `ctID` or `mtID`) to a specific robot (`robID`). It allows the task to be an atomic, compound or mission task.
  If multiple constraints exist, the allocation algorithm reads each constraint one by one. Hence, order matters when writing these requiments. For example, there are three robots $r1, r2, r3$, a compound task $ct1$ with two atomic subtasks $(at1, at2)$ and a mission task $m1$ to accomplish $ct1$; the user specifies three constraints (in this order): (a) allocate $m1$ to $r1$, (b) allocate $ct1$ to $r2$, (c) allocate $at1$ to $r3$.
  In this case, $m1$ is first allocated to $r1$. Then, the second constraint overwrites the allocation and assigns $m1$ to $r2$ (as $m1$ only consists of $ct1$). Finally, the subtask $at1$ of $m1$ is allocated to $r3$ so mission $m1$ is completed by $r2$ and $r3$.

  If no robot is provided (`single robot`), the mission, atomic or compound task is assigned to a robot or (for joint tasks) set of robots that possess the capability to complete it.

- `closest` allocates a task or all tasks (`all tasks`, `atID`, `ctID` or `mtID`) to the robot with the shortest path to the task location. Similarly to `allocateT`, the order in which multiple constraints of this type are defined matters.

- `maxTasks` limits the number of tasks $\texttt{v} \in \mathbb{N}$ to allocate to a robot or for all robots (`in robot robID`, `per robot`).

Finally, the mission **parameter** (`time: v`) limits the time `v` $\in \mathbb{N}$ within which the mission must be completed.

Formal Specification. The tasks that the robots must complete (atomic or compound) are named *mission tasks*; each consists of an identifier of tyype

$$[MissionTaskID]$$

and the identifier of the task *taskID*, atomic or compound, that the robots must complete at a given location with identifier *locationID*,

```
MissionTask ______________________________
| id : MissionTaskID
| taskID : TaskID
| locationID : LocationID
|_________________________________________
```

There are three types of optimisation objectives related to the mission. The user can select between minimising the time that the robots stay idle (*minimiseIdle*), minimising the travelling cost of travelling between locations to complete the mission (*minimiseTravel*) and maximising the probability of succeeding with the mission (*maximiseSuccess*),

$$OBJ ::= minimiseIdle \mid minimiseTravel \mid maximiseSuccess$$

The six types of constraints are defined as,
(a) *rateSucc*: the rate of successfully completing a mission,
(b) *spaceXY*: the strict coordinates on which a robot can have allocated tasks. In the Z notation, this is divided into four combinations constraining the coordinates x and y (XY), and the restriction of being *greater* or *lower* than (GLT). We define the coordinates and restrictions as,

$$XY ::= x \mid y$$

$$GLT ::= greater \mid lower$$

(c) *taskTime*: define the start or completion time of an atomic or compound task, and *mtaskTime* for mission tasks. We define the *start* or *end* restriction as,

$$STARTEND ::= start \mid end$$

(d) *allocateT*: assign an atomic or compound task to a designated robot; *allocatemT* assigns a mission task to a specified robot. Similarly, for assigning all tasks to a single (undefined) robot, utilise *allocTtogether* and *allocmTtogether*;
(e) *closest*: allocate an atomic or compound task to the closest robot; similarly, *closestmT* allocates the mission task (and all recursive subtasks from its compound tasks) to the

closest robot.
(f) *maxTasks*: limit the maximum number of tasks to assign to a robot.
These constraints are shown in the following schema,

Constraints

$$
\begin{array}{l}
\mathit{rateSucc} : \mathbb{N} \\
\mathit{spaceXY} : \mathbb{P}((\mathit{RobotID} \times \mathit{XY} \times \mathit{GLT}) \times \mathbb{N}) \\
\mathit{taskTime} : \mathbb{P}((\mathit{TaskID} \times \mathit{STARTEND}) \times \mathbb{N}) \\
\mathit{mtaskTime} : \mathbb{P}((\mathit{MissionTaskID} \times \mathit{STARTEND}) \times \mathbb{N}) \\
\mathit{allocateT} : \mathbb{P}(\mathit{TaskID} \times \mathit{RobotID}) \\
\mathit{allocatemT} : \mathbb{P}(\mathit{MissionTaskID} \times \mathit{RobotID}) \\
\mathit{allocTtogether} : \mathbb{P}\ \mathit{TaskID} \\
\mathit{allocmTtogether} : \mathbb{P}\ \mathit{MissionTaskID} \\
\mathit{closest} : \mathbb{P}\ \mathit{TaskID} \\
\mathit{closestmT} : \mathbb{P}\ \mathit{MissionTaskID} \\
\mathit{maxTasks} : \mathbb{P}(\mathit{RobotID} \times \mathbb{N}) \\
\hline
0 < \mathit{rateSucc} \leq 100
\end{array}
$$

Finally, a mission is defined as the set of mission tasks *tasks*, the set of objectives and constraints and the mission parameter, *time*, defining a deadline to complete the mission.

Mission

$$
\begin{array}{l}
\mathit{tasks} : \mathbb{P}\ \mathit{MissionTask} \\
\mathit{constraints} : \mathit{Constraints} \\
\mathit{objectives} : \mathbb{P}\ \mathit{OBJ} \\
\mathit{time} : \mathbb{N}
\end{array}
$$

As detailed in the following chapters, a time deadline is essential to the mission because of the approach used to solve the task scheduling problem within the KANOA framework. The KANOA task scheduler generates a state-transition model that tracks the robots' progress on tasks, and the time limit defines the maximum allowable idle time for the robots. Without this bound, the idling state variable overflows leading to a state explosion.

4.2.6 Problem Specifications

Syntax. We can now define the *problem specification* as shown in Table 4.4: (a) the world model; (b) the tasks model; (c) the robots available; and (d) the mission to be carried out by the robots,

```
ProblemSpecification{ WorldModel: world
                      TaskModel:task
                      RobotsModel:robot
                      Missions:miss  }
```

Formal Specification. Finally, we formally define the *system specification* as,

$$\begin{array}{l} \textit{ProblemSpecification} \\ \hline \textit{worldModel} : \textit{World} \\ \textit{tasksModel} : \textit{TaskSet} \\ \textit{robotsModel} : \mathbb{P}\,\textit{Robot} \\ \textit{mission} : \textit{Mission} \\ \hline \end{array}$$

KANOA DSL is domain-specific in the sense that it focuses on multi-robot system missions; however, it is generic in its support of different applications and scenarios. For example, it can be used to define a hospital MRS mission where robots must clean and rearrange medical equipment (as presented in Section 4.3) or in the description of the deployment of an MRS to map a natural disaster area in a search-and-rescue scenario.

To ensure that the specifications written in Z notation are well-formed, we verified it using the fuzz typechecker for Z [54]. Fuzz is a collection of tools that check Z specifications for correctness in their formulae and compliance with the Z scope and type rules. It is built to check the grammar and correctness in Z specifications written in a dedicated Z LaTeX package.

Complexity of the problem specification and discussion. To discuss the complexity arising from the specification of the problem, we discussed each of its parts separately. The world model depends on the number of locations *Location* and the number of paths *dist*. For n_l number of locations, the number of paths n_p can be calculated as the number of edges in a graph of length $n_p = \frac{n_l(n_l-1)}{2}$. This represents an upper bound since users can disregard some paths due to real-world constraints. The KANOA DSL is agnostic to the motion planning used to travel between two locations. KANOA DSL does not support the direct encoding of properties such as the elimination of subtours affecting MRS travelling salesman problems (in which trajectories are not continuous as disconnected paths, called subtours, exist)[3]. However, as described in the following Chapters,

[3]Subtour elimination inequalities is a crucial concept in solving the Traveling Salesman Problem (TSP),

these properties are not necessary for the types of solutions implemented in the KANOA tool (Chapter 7.1).

The size of the robot model depends on the number of robots and the number of capabilities n^r_{cap} for each robot r, i.e. $\Sigma_r n^r_{cap}$. The task model allows us to specify (atomic and compound) tasks only once and reuse them as part of the subtasks of compound tasks. This reduces the size of the task model to the sum of the number of atomic tasks and compound tasks $n_{at} + n_{ct}$.

The mission specification complexity is linear in the number of mission tasks n_{mt}, the number of constraints n_c. Hence, the complexity of the mission specification is linear as $n_l, n_p, n_{at}, n_{ct}, n_{cap}, n_c \in \mathbb{N}$ are finite.

Finally, we can conclude that the problem specification is linear in the size of robots, task and mission models, but exponential in the size of the world model. As the number of paths in the world model increases combinatorially with the number of locations, the KANOA tool (introduced in Chapter 7.1) simplifies the writing on the KANOA DSL by providing an option to automatically represent all unspecified paths as the Euclidean distance between any two locations.

4.3 Hospital Case Study

In this section, we illustrate the use of the KANOA DSL to specify a hospital MRS mission in which a set of robots must complete a series of tasks inspired by the robot missions from the RoboMAX catalogue [20]. In the hospital, patient rooms must be kept clean, and medical equipment has to be moved around to perform daily operations in the surgical room. We describe a concrete scenario where the physical space consists of six rooms, Room1 to Room6 as depicted in Figure 4.6. Rooms 1 and 6 are surgical rooms, where robots must move the surgical equipment into place to make it ready for the next operation. Rooms 2 to 5 are patient rooms that the robot must clean in a certain manner. Hence, the **mission** consists of six mission tasks (M1 to M6), where M1 and M2 require rearranging rooms 1 and 6, and M3–M6 require robots to clean patient rooms 2 to 5.

Moving medical equipment (AT1) is a task requiring two robots to meet and coordinate in the room to complete the task. Cleaning a patient's room (CT2) is a task consisting of two subtasks. First, the patient must be notified that the robot is about to start cleaning (AT4). Second, the robot must proceed to clean the room (CT1), which comprises cleaning the floor (AT2) and sanitising the room (AT3), in no specific order. Notice that the subtasks of CT2 are required to be done in that specific order and each of the subtasks

particularly in integer programming formulations. The main purpose of these inequalities is to ensure that any feasible solution to the TSP visits every vertex exactly once, thereby eliminating the possibility of isolated loops [154]. The TSP involves finding the shortest possible route that visits a set of cities exactly once and returns to the origin city.

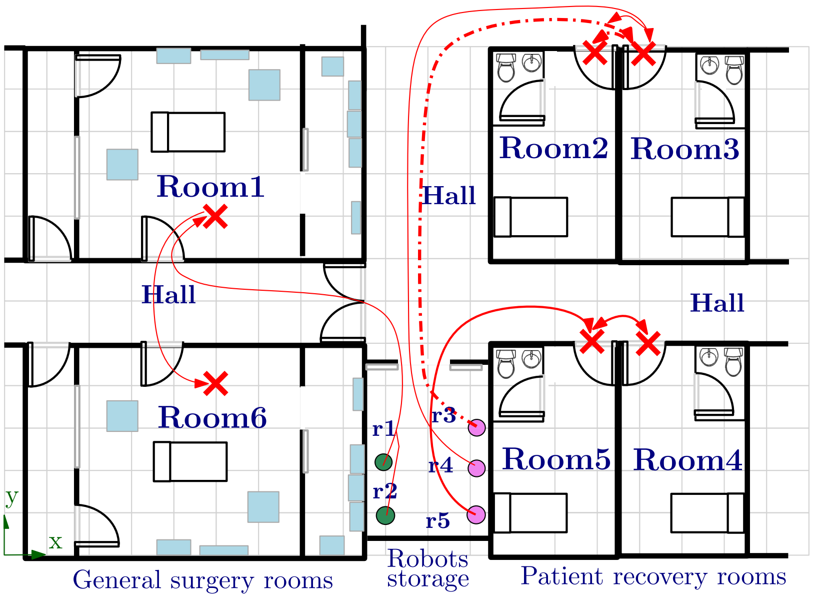


Figure 4.6: Floor plan of hospital area with six rooms serviced by robots r1-r5. Crosses mark the locations that robots must reach to start a task in each room, and blue rectangles denote medical equipment (inspired by [155]). The red lines show a possible robots' schedule to clean rooms 2 to 4, and to move equipment in rooms 1 and 6.

requires only one robot to complete them.

There are five robots available within the hospital (r1 to r5). Robots r1–r3 are cleaner robots able to sanitise rooms, clean the floor and communicate with patients when they start cleaning (i.e., they are capable of performing tasks at2--at4). Robots r4 and r5 are pick-and-place robots able to rearrange equipment and cooperate between them to pick up larger objects (i.e., capable of performing task at1). Each robot has a given speed and a given probability and time of completing a task, as shown in Table 4.5. At the beginning, the robots are located in the robot storage room.

The layout in Figure 4.6 has its origin on the lower left side and each square measures 1×1 units. Hence, we know the exact X and Y initial position of the robots and the location at each room (marked with a cross) where robots must report to start activities in these rooms. The length of paths between any two locations is also known beforehand.[4]

Besides task constraints imposed within the tasks definitions, such as the number of robots required to complete a task and the ordering in which some of the tasks must be completed, there are other **constraints** that the robots must consider when completing the mission: (a) the mission must be completed within 120 time-units; (b) the mission must be completed with a rate of success greater than 90%; (c) robot r3 must be close to patient room room5 because the patient is in critical condition, hence a constraint is placed to move r3 only in positions where x>9 and y<7, and the mission in room5

[4]The coordinates of each of the rooms' locations, the initial positions of the robots and the travel distances between locations are shown in Z schema boxes in Section 4.3.2.

Table 4.5: Robot parameter values for the hospital case study: velocity (meters/minute), probability of completing a task and time to complete a task (minutes). NA means not apply as the robot does not possess the capability to perform the task.

				Time to complete task AT$_i$				**Probability of completing task AT$_i$**			
Robot	**Type**	**Initial Loc.**	**Velocity**	**AT1 (move)**	**AT2 (floor)**	**AT3 (sanit)**	**AT4 (notify)**	**AT1**	**AT2**	**AT3**	**AT4**
r1	cleaner	l1	1	Na	5	5	1	Na	0.95	0.95	0.99
r2	cleaner	l2	1	Na	5	4	1	Na	0.85	0.85	0.90
r3	cleaner	l3	2	Na	6	6	1	Na	0.99	0.99	0.99
r4	p & p	l4	2	4	Na	Na	Na	0.99	Na	Na	Na
r5	p & p	l5	2	4	Na	Na	Na	0.99	Na	Na	Na

is explicitly assigned to r3; and (d) probability of a robot moving successfully between rooms 2 and 3 is set to 90% as the two rooms are linked by a narrow hall where the robots may collide with the walls.

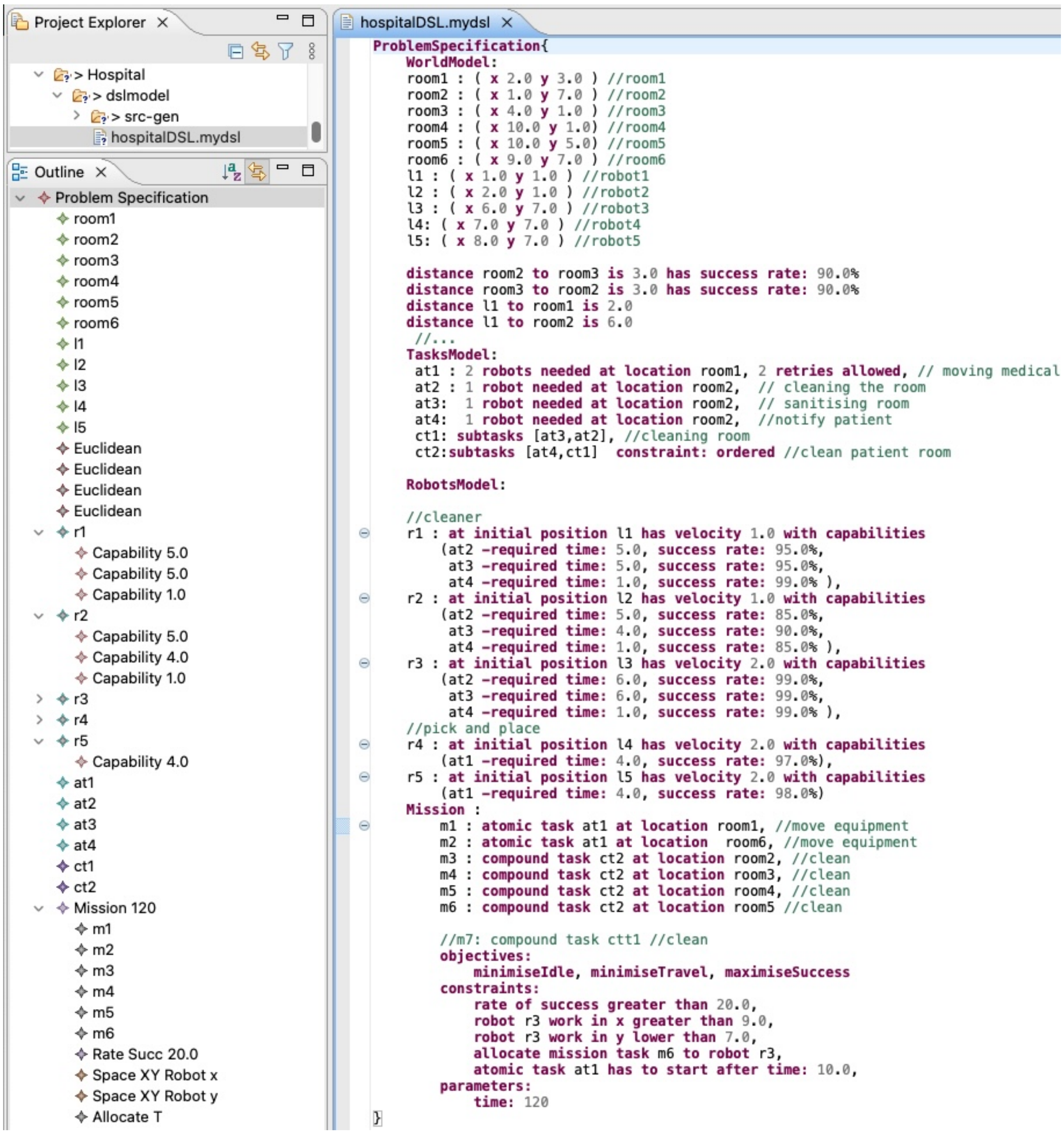
Finally, the mission must be completed optimising three objectives: (i) minimisation of idle time, (ii) maximisation of the probability of success, and (iii) minimisation of the travelling cost. These objectives are not independent, and optimising one may affect the optimisation of the others. For example, minimising the travelling cost may mean that a robot spends more time at the location where the tasks are to be completed, which in turn means that the idling waiting for other robots to complete some of their tasks may increase. Also, robots that have a greater probability of success may be successful because they travel slower and are at less risk of collision; hence, maximising the probability of succeeding with the mission may increase the travelling cost.

So far, we have described the different parts of the hospital, the mission to be accomplished and the set of constraints and optimisation objectives. For the mission to be accomplished, a plan must be created such that the robots must know what tasks to do, how to reach the locations of these tasks, and at what time to do so. The red trajectories in Figure 4.6 show an example of how the robots can be deployed following their synthesised plans. This problem is further explored in Chapters 5 and 6.

4.3.1 Hospital Case Study in KANOA DSL

We encoded the hospital case study using the KANOA DSL as introduced in Section 4.2.1. Figure 4.7 shows the problem specification containing the hospital world model, the tasks model, the robots available to be deployed and the hospital mission. The `WorldModel` shows the locations of the six rooms, and the locations `l1-l5` of the robots, followed by (a subset of) the possible paths between two locations.

The `TaskModel` depicts the four atomic tasks `at1-at4`, the number of robots required for each of these tasks, and the locations to complete these tasks. For example "`at1 : 2 robots needed at location l1,`" means that there is an atomic task with identifier



```
ProblemSpecification{
    WorldModel:
    room1 : ( x 2.0 y 3.0 ) //room1
    room2 : ( x 1.0 y 7.0 ) //room2
    room3 : ( x 4.0 y 1.0 ) //room3
    room4 : ( x 10.0 y 1.0) //room4
    room5 : ( x 10.0 y 5.0) //room5
    room6 : ( x 9.0 y 7.0 ) //room6
    l1 : ( x 1.0 y 1.0 ) //robot1
    l2 : ( x 2.0 y 1.0 ) //robot2
    l3 : ( x 6.0 y 7.0 ) //robot3
    l4: ( x 7.0 y 7.0 ) //robot4
    l5: ( x 8.0 y 7.0 ) //robot5

    distance room2 to room3 is 3.0 has success rate: 90.0%
    distance room3 to room2 is 3.0 has success rate: 90.0%
    distance l1 to room1 is 2.0
    distance l1 to room2 is 6.0
     //...
    TasksModel:
     at1 : 2 robots needed at location room1, 2 retries allowed, // moving medical
     at2 : 1 robot needed at location room2,  // cleaning the room
     at3:  1 robot needed at location room2,  // sanitising room
     at4:  1 robot needed at location room2,  //notify patient
     ct1: subtasks [at3,at2], //cleaning room
     ct2:subtasks [at4,ct1]  constraint: ordered //clean patient room

    RobotsModel:

    //cleaner
    r1 : at initial position l1 has velocity 1.0 with capabilities
        (at2 -required time: 5.0, success rate: 95.0%,
         at3 -required time: 5.0, success rate: 95.0%,
         at4 -required time: 1.0, success rate: 99.0% ),
    r2 : at initial position l2 has velocity 1.0 with capabilities
        (at2 -required time: 5.0, success rate: 85.0%,
         at3 -required time: 4.0, success rate: 90.0%,
         at4 -required time: 1.0, success rate: 85.0% ),
    r3 : at initial position l3 has velocity 2.0 with capabilities
        (at2 -required time: 6.0, success rate: 99.0%,
         at3 -required time: 6.0, success rate: 99.0%,
         at4 -required time: 1.0, success rate: 99.0% ),
    //pick and place
    r4 : at initial position l4 has velocity 2.0 with capabilities
        (at1 -required time: 4.0, success rate: 97.0%),
    r5 : at initial position l5 has velocity 2.0 with capabilities
        (at1 -required time: 4.0, success rate: 98.0%)
    Mission :
        m1 : atomic task at1 at location room1, //move equipment
        m2 : atomic task at1 at location  room6, //move equipment
        m3 : compound task ct2 at location room2, //clean
        m4 : compound task ct2 at location room3, //clean
        m5 : compound task ct2 at location room4, //clean
        m6 : compound task ct2 at location room5 //clean

        //m7: compound task ctt1 //clean
        objectives:
            minimiseIdle, minimiseTravel, maximiseSuccess
        constraints:
            rate of success greater than 20.0,
            robot r3 work in x greater than 9.0,
            robot r3 work in y lower than 7.0,
            allocate mission task m6 to robot r3,
            atomic task at1 has to start after time: 10.0,
        parameters:
            time: 120
}
```

Figure 4.7: KANOA tool screenshot showing DSL encoding of the hospital MRS mission from Section 4.3.

`at1` requiring 2 robots at location `l1` to be accomplished. This location can be changed when defining the mission tasks, as we proceed to do later on.

The `RobotModel` shows the five robots available at the hospital. For example, the first robot is defined as "`r1:at initial position` l1 `has velocity` 2.0 `with capabilities` ..." followed by its set of capabilities. Its first capability is written as "`at2-required time:` 5.0, `success rate:` 99.0%," meaning that robot `r1` requires 5 time units to complete task `at2` and succeeds with this task 99.0% of the times. The robot's performance information is previously provided in Table 4.5.

Next, the `Mission` is formed by the list of mission tasks to be done by robots and the associated mission objective, constraints and parameters. The mission task "`m1: atomic task at1 at location room1`" has identifier `m1` and requires the robots to perform atomic task `at1` at `room1`. The mission optimisation objectives require to minimise the idling time (`minimiseIdle`), minimise the travelling cost (`minimiseTravel`) and maximise the probability of succeeding with the set of mission tasks (`maximiseSuccess`). There are three mission constraints: the rate of succeeding with the mission must be `greater than` 90%, robot $r3$ must work in a section of the hospital where x is `greater than` 9 units and y is `lower than` 7 units. Robot $r3$ is also assigned to mission `m6`. Finally, the mission parameter specifies a time limit of 120 time units for completing the mission (`time:`120).

4.3.2 Hospital Case Study Formalisation

In the hospital case study, there are 6 rooms. We define the location identifiers of rooms 1 to 6 as $room1 - room6$, and their coordinates; for example, the coordinates of room1 and room2 are (2,3) and (1,7), respectively.

$$\begin{array}{|l}
room1, room2, room3, room4, room5, room6 : LocationID \\
Room1, Room2, Room3, Room4, Room5, Room6 : Location \\
\hline
Room1.id = room1 \land Room1.x = 2 \land Room1.y = 3 \\
Room2.id = room2 \land Room2.x = 1 \land Room2.y = 7 \\
Room3.id = room3 \land Room3.x = 4 \land Room3.y = 1 \\
Room4.id = room4 \land Room4.x = 10 \land Room4.y = 1 \\
Room5.id = room5 \land Room5.x = 10 \land Room5.y = 5 \\
Room6.id = room6 \land Room6.x = 9 \land Room6.y = 7
\end{array}$$

We also describe locations related to the initial positions of the robots

$$loc1, loc2, loc3, loc4, loc5 : LocationID$$
$$Loc1, Loc2, Loc3, Loc4, Loc5 : Location$$
$$Loc1.id = loc1 \land Loc1.x = 1 \land Loc1.y = 1$$
$$Loc2.id = loc2 \land Loc2.x = 2 \land Loc2.y = 1$$
$$Loc3.id = loc3 \land Loc3.x = 6 \land Loc3.y = 7$$
$$Loc4.id = loc4 \land Loc4.x = 7 \land Loc4.y = 7$$
$$Loc5.id = loc5 \land Loc5.x = 8 \land Loc5.y = 7$$

We now define the world model with the rooms and robot locations, as well as the paths between them. In the following schema, we only show the paths between the robots and the rooms to reduce space,

$$hospitalWorld : World$$
$$hospitalWorld.locations = \{Room1, Room2, Room3, Room4, Room5, Room6, Loc1, Loc2, Loc3, Loc4, Loc5\}$$
$$hospitalWorld.dist(l1, room1) = 2 \land hospitalWorld.dist(l1, room2) = 6$$
$$hospitalWorld.dist(l1, room3) = 3 \land hospitalWorld.dist(l1, room4) = 9$$
$$hospitalWorld.dist(l1, room5) = 12 \land hospitalWorld.dist(l1, room6) = 14$$
$$hospitalWorld.dist(l2, room1) = 2 \land hospitalWorld.dist(l2, room2) = 6$$
$$hospitalWorld.dist(l2, room3) = 2 \land hospitalWorld.dist(l2, room4) = 8$$
$$hospitalWorld.dist(l2, room5) = 11 \land hospitalWorld.dist(l2, room6) = 13$$
$$hospitalWorld.dist(l3, room1) = 7 \land hospitalWorld.dist(l3, room2) = 5$$
$$hospitalWorld.dist(l3, room3) = 7 \land hospitalWorld.dist(l3, room4) = 8$$
$$hospitalWorld.dist(l3, room5) = 5 \land hospitalWorld.dist(l3, room6) = 3$$

$$hospitalWorld.dist(l4, room1) = 8 \land hospitalWorld.dist(l4, room2) = 6$$
$$hospitalWorld.dist(l4, room3) = 8 \land hospitalWorld.dist(l4, room4) = 7$$
$$hospitalWorld.dist(l4, room5) = 4 \land hospitalWorld.dist(l4, room6) = 2$$
$$hospitalWorld.dist(l5, room1) = 9 \land hospitalWorld.dist(l5, room2) = 6$$
$$hospitalWorld.dist(l5, room3) = 9 \land hospitalWorld.dist(l5, room4) = 6$$
$$hospitalWorld.dist(l5, room5) = 3 \land hospitalWorld.dist(l5, room6) = 1$$
$$hospitalWorld.succRate(room2, room3) = 90$$
$$hospitalWorld.succRate(room3, room2) = 90$$

The tasks to be done in the hospital consist of: (a) cleaning an empty room $ct1\_clean$, (b) cleaning a patient room $ct2\_patient$, and (c) moving medical equipment $at1\_move$. Cleaning an empty room is a compound task with two atomic subtasks: floor cleaning $at2\_floor$ and room sanitising $at3\_sanit$. Cleaning a patient's room ($ct2\_patient$) consists of notifying the patient in the room that the cleaning is about to start $at4\_notify$, followed by compound task $ct1\_clean$. Lastly, moving medical equipment is an atomic task requiring two robots. The tasks are defined as:

$$ct1\_clean, ct2\_patient : \mathit{CTaskID}$$
$$at1\_move, at2\_floor, at3\_sanit, at4\_notify : \mathit{ATaskID}$$

$$AT1, AT2, AT3, AT4 : \mathit{AtomicTask}$$

$$AT1.id = at1\_move \land AT2.id = at2\_floor \land AT3.id = at3\_sanit \land AT4.id = at4\_notify$$
$$AT1.nRobots = 2 \land AT2.nRobots = 1 \land AT3.nRobots = 1 \land AT4.nRobots = 1$$

Notice that we did not set the locations of the atomic tasks (e.g., AT2.locationID = room1), as we define them later when the mission is specified. Continuing,

$$CT1, CT2 : \mathit{CompoundTask}$$

$$CT1.id = ct1\_clean \land CT1.subtasks = \langle at2\_floor, at3\_sanit\rangle \land CT1.constraint = none$$
$$CT2.id = ct2\_patient \land CT2.subtasks = \langle at4\_notify, ct1\_clean\rangle \land CT2.constraint = ordered$$

$$hospitalTasksSet : \mathit{TaskSet}$$

$$hospitalTasksSet.atomicTasks = \{AT1, AT2, AT3\}$$
$$hospitalTasksSet.compoundTasks = \{CT1, CT2\}$$

Continuing with the robot's model, there are five robots available at the hospital: three cleaners, $r1$-$r3$; and two pick-and-place, $r4$ and $r5$,

$$r1, r2, r3, r4, r5 : \mathit{RobotID}$$

Robots with identifiers $r1$-$r3$ have the capabilities to clean rooms (atomic tasks *at*2*_floor* and *at*3*_sanit*), as well as notify a patient that a robot is about to clean the room (perform task *at*4*_notify*). We use the form "CapXrN" to define the capability of accomplishing X of robot N. We now define robots with identifiers $r1$-$r3$ (their tasks, completion times *time* and success rates *succRate* for each of its capabilities) as,

$$
\begin{array}{|l}
Cap1r1, Cap2r1, Cap3r1 : Capability \\
Cap1r2, Cap2r2, Cap3r2 : Capability \\
Cap1r3, Cap2r3, Cap3r3 : Capability \\
\hline
Cap1r1.task = at2\_floor \land Cap1r1.time = 5 \land Cap1r1.succRate = 95 \\
Cap2r1.task = at3\_sanit \land Cap2r1.time = 5 \land Cap2r1.succRate = 95 \\
Cap3r1.task = at4\_notify \land Cap3r1.time = 1 \land Cap3r1.succRate = 99 \\
\\
Cap1r2.task = at2\_floor \land Cap1r2.time = 5 \land Cap1r2.succRate = 85 \\
Cap2r2.task = at3\_sanit \land Cap2r2.time = 4 \land Cap2r2.succRate = 90 \\
Cap3r2.task = at4\_notify \land Cap3r2.time = 1 \land Cap3r2.succRate = 85 \\
\\
Cap1r3.task = at2\_floor \land Cap1r3.time = 6 \land Cap1r3.succRate = 99 \\
Cap2r3.task = at3\_sanit \land Cap2r3.time = 6 \land Cap2r3.succRate = 99 \\
Cap3r3.task = at4\_notify \land Cap3r3.time = 1 \land Cap3r3.succRate = 99
\end{array}
$$

Similarly, robots with identifiers $r4$ and $r5$ can move medical equipment. Hence, we define the capability of moving equipment for robot $r4$ (Cap1r4) and robot $r5$ (Cap1r5) as,

$$
\begin{array}{|l}
Cap1r4, Cap1r5 : Capability \\
\hline
Cap1r4.task = at1\_move \land Cap1r4.time = 4 \land Cap1r1.succRate = 97 \\
Cap1r5.task = at1\_move \land Cap1r5.time = 4 \land Cap1r2.succRate = 98
\end{array}
$$

The robots have respective initial locations given by l1,...l5 identifiers, as specified in the world model. Hence, we define each robot by its identifier, velocity and set of capabilities,

$$
\begin{array}{|l}
R1, R2, R3, R4, R5 : Robot \\
\hline
R1.id = r1 \land R1.initLocation = l1 \land R1.velocity = 1 \land R1.capabilities \\
\qquad = \{Cap1r1, Cap2r1, Cap3r1\} \\
R2.id = r2 \land R2.initLocation = l2 \land R2.velocity = 1 \land R2.capabilities \\
\qquad = \{Cap1r2, Cap2r2, Cap3r2\} \\
R3.id = r3 \land R3.initLocation = l3 \land R3.velocity = 2 \land R3.capabilities \\
\qquad = \{Cap1r3, Cap2r3, Cap3r3\} \\
R4.id = r4 \land R4.initLocation = l4 \land R4.velocity = 2 \land R4.capabilities \\
\qquad = \{Cap1r4\} \\
R5.id = r5 \land R5.initLocation = l5 \land R5.velocity = 2 \land R5.capabilities \\
\qquad = \{Cap1r5\}
\end{array}
$$

We define the hospital mission as the list of tasks to be done by the robots: rearrange medical equipment in rooms 1 and 6 ($m1, m2$); and clean patient rooms room2 to room5 ($m3 - m6$),

$$m1, m2, m3, m4, m5, m6 : MissionTaskID$$

$$M1, M2, M3, M4, M5, M6 : MissionTask$$

$$\begin{aligned}
&M1.id = m1 \land M1.taskID = at1\_move \land M1.locationID = room1\\
&M2.id = m1 \land M2.taskID = at1\_move \land M2.locationID = room6\\
&M3.id = m1 \land M3.taskID = ct2\_patient \land M3.locationID = room2\\
&M4.id = m1 \land M4.taskID = ct2\_patient \land M4.locationID = room3\\
&M5.id = m1 \land M5.taskID = ct2\_patient \land M5.locationID = room4\\
&M6.id = m1 \land M6.taskID = ct2\_patient \land M6.locationID = room5
\end{aligned}$$

We can define the constraints of the mission as follows. First, the rate of success must be at least 90%; then, robot $r3$ is constrained to accept tasks only within the coordinates $x > 9$ and $y < 7$,

$$hospitalConstraints : Constraints$$

$$\begin{aligned}
&hospitalConstraints.rateSucc = 90\\
&hospitalConstraints.spaceXY = \{(r3, x, greater) \mapsto 9, (r3, y, lower) \mapsto 7\}
\end{aligned}$$

Now we define the hospital mission as,

$$hospitalMission : Mission$$

$$\begin{aligned}
&hospitalMission.tasks = \{M1, M2, M3, M4, M5, M6\}\\
&hospitalMission.constraints = hospitalConstraints\\
&hospitalMission.objectives = \{minimiseIdle, minimiseTravel, maximiseSuccess\}\\
&hospitalMission.time = 120
\end{aligned}$$

Lastly, we can define the hospital problem specifications as,

$$HospitalScenario : ProblemSpecification$$

$$\begin{aligned}
&HospitalScenario.worldModel = hospitalWorld\\
&HospitalScenario.tasksModel = hospitalTasksSet\\
&HospitalScenario.robotsModel = \{R1, R2, R3, R4, R5\}\\
&HospitalScenario.mission = hospitalMission
\end{aligned}$$

Similarly to Section 4.2.1, we used the Z notation checker fuzz [54] to ensure that the instantiation of the hospital case study is well-formed.

4.3.3 Discussion

KANOA's DSL is designed in a separation-of-concerns fashion. As it draws a separation between the world model, tasks model, robots model and the mission itself, it makes it easier to identify the relation between these parts of an MRS problem and any existing constraints to be modelled. For example, the probability of succeeding when travelling between locations concerns the world model. Hence, this type of constraint is encoded into the paths as part of the world model. Another advantage to exploit in future work is that if a new MRS problem requires the extension of the problem definition, KANOA's problem specification design makes it easier to add these extensions. For example, consider a group of mobile robots to be deployed in a search-and-rescue scenario where some robots have newer wheels better adapted to the terrain. In this case, the designer may modify the probability of succeeding with the paths to be dependent on the robots, from:

```
path ::= distance locID_1 to locID_2 is v
                    (and success rate:v_[0,100] % )?
```

by adding the robot's dependency as:

```
path ::= distance locID_1 to locID_2 is v
                    (and success rate:v_[0,100] % for robot robID)?
```

Similarly, as robotic task decomposition[5] has been the centre of attention in multiple studies such as in [156], an extension of the current syntax can add also add these alternatives to subtasks by modifying the compound task specification as:

```
c_task ::= ctID : subtasks [
                    { (ctID | atID)
                    ((,ctID | ,atID))* }
                    ] (constraint: ordered | constraint:consecutive)?
```

to:

```
c_task ::= ctID : subtasks [
                    { (ctID | atID) ((or ctID | or atID))*
                    ( , (ctID | atID) ((or ctID | or atID))* )* }
                    ] (constraint: ordered | constraint:consecutive)?
```

In the following chapters, we map the semantics of KANOA problem specification to models for the task allocation and scheduling, and the generation of robot plans. If the current problem specification is modified, the semantics must capture these changes.[6]

[5]By task decomposition we mean the possibility of having logical *or*s in the subtasks of compound tasks rather than only a sequence of these (implicitly defining logical *and*s). This flexibility allows for diverse configurations of subtasks to define the same compound task.

[6]We also propose methods that ease the adoption of new constraints for the task allocations and the task scheduling problems, described in Sections 5.7 and 6.8, respectively.

4.4 Related Work

Early applications of robotic systems deployed in the industry rely on tailored controllers following pre-defined tasks [20]. However, as these systems are increasingly used in more complex critical domains such as search-and-rescue and healthcare, so is the need to define robotic missions unambiguously. Beyond natural language, multiple approaches have been considered for the specification of robotic missions throughout the years, including hierarchical representations such as HTN [88] and precedence diagrams [139, 140]; language-based such as PDDL [23, 138] and formal grammars [128, 130]; motion-based such as movement primitives [136] and Jacobian trajectories [129]; and logic-based approaches [89, 104]. Among this spectrum of solutions, the last one offers an unambiguous common language for the specification of robotic missions, and has been successfully used for the specification of such missions. These findings are summarised in Table 3.7, part of the literature review carried out in Chapter 3.

Different types of temporal logic have been widely explored for mission specifications. These logics provide formal languages avoiding ambiguity in the specifications of missions and mission requirements. The most common found in the literature is LTL [25, 27, 28, 86, 104]. Researchers choose this formalism for several reasons. Firstly, it is widely recognized and has been effectively utilised in specifying robotic missions. Secondly, it provides simpler semantics compared to other logic. Lastly, it is integrated into formal techniques such as model checking for verifying LTL properties and reactive synthesis for generating models from LTL formulae such as Büchi automata, NBA and NFA. These findings are illustrated in Table 3.6, Chapter 3, showcasing recent studies that have employed LTL in the context of robotic systems.

One of the limitations of LTL is that it can only be used for qualitative verification of single traces (i.e. linear time). Properties such as "perform task A before task B" can be expressed in LTL. However, LTL falls short when it comes to representing temporal constraints reasoning about time-bound scenarios like "perform task B within 10 minutes of task A". To overcome this limitation, in [89] and [100], Gundana D. et al. employ event-based Signal Temporal Logic (STL) which introduces the time-bounded *eventually*, *always* and *until* operators. A logic based on STL called Capability Temporal Logic (CaTL) is successfully employed by Leahy K. et al. [92]. Our QUARTET catalogue allows the specification of properties using these bounded operators with patterns such as *pause* and *timeout*. Moreover, the use of RPCTL logic in QUARTET allows the specification of quantitative properties such as *maximising energy while performing a mission*. To the best of our knowledge, the only other study that uses RPCTL for the specification of robotic mission properties is [133] to reason about the minimum time to reach the goal ($\mathcal{E}_{min=?}(\mathcal{F} goal)$), the maximum remaining energy ($\mathcal{E}_{max=?}(\mathcal{F} goal)$) and the maximum probability of collision ($\mathcal{P}_{max=?}[\mathcal{F}(goal \wedge \neg collided)]$. The QUARTET repository supports all of these properties. RPCTL properties allow the verification of some probabilistic state-transition models (see Section 2.3.1). Rewards in properties identified by the symbol $\mathcal{E}$ refer to a real positive value that is increased at specific states or transitions described within the probabilistic model under consideration. In PRISM, these are defined within *reward structures*. We refer the reader to [73] for further information..

Due to the intricacy of logic languages, multiple domain-specific languages have been created for the description of robotic systems [84, 112, 114]. For example, PsALM [36] provides a structured English grammar for the specification of missions, which is translated into LTL and CTL [2]. NASA's Formal Requirements Elicitation Tool (FRET) provides support to write their system's requirements using a restricted natural language. This general-purpose tool provides the automatic elicitation of properties in metric temporal logic and has been explored for robotic scenarios [157, 158]. In contrast, KANOA is designed as a user-friendly language for the specification of missions for mobile robots designed in a separation-of-concerns fashion. The semantics of KANOA is formalised in Z notation and its semantics can be tailored for the application in hand. In Chapters 5 and 6, we use KANOA as the starting point for the allocation and scheduling of robotic tasks.

Multiple catalogues for logic specifications have emerged easing the adoption of temporal logic requirements. Two of the most known come from software engineering, [159] and [160]. In contrast, repositories of patterns for the specification of mobile robot missions (e.g., PsALM [36] and QUARTET) employ different types of patterns focusing on agents (robots), locations and the quality of completing such missions (e.g., verifying that the battery's energy is more than 20% when the mission is completed). ROBOMAX [20] is a living repository of robotic missions described in natural language whose missions are worth exploring in future work.

4.5 Summary

In this chapter, we described the QUARTET catalogue, a new repository of quantitative robotic mission specifications. As part of this thesis, we presented its semantics that map QUARTET quantitative specification patterns to a type of temporal logic that supports the quantitative reasoning of probabilities and costs, RPCTL. We also developed a tool for the automatic formalisation of these mission specifications to ease its adoption.

Additionally, we proposed the KANOA DSL, a user-friendly domain-specific language for the specification of multi-robot system missions underpinned by a separation of concerns approach. We presented the KANOA DSL syntax and formalised its components using the Z notation. Also, we introduced a hospital case study, from which we instantiated the KANOA DLS and its associated Z notation, to illustrate the use of KANOA for a concrete MRS mission. Within the definition of a mission, we described the tasks that the robots must perform, a set of objectives, constraints and parameters associated with the mission. We note that the KANOA DSL can be expanded to other types of constraints; for example, by adding a constraint specifying that there cannot be more than N robots in a room at the same time, or by specifying the initial level of energy within the robots' description and the required energy while travelling or performing tasks within the world and the task models.

Chapter 5

Task Allocation

The multi-robot task allocation problem involves the partitioning of the tasks among the available robots. Assigning tasks to robots is a challenging endeavour due to the existence of compound tasks (tasks with subtasks), temporal and spatial task constraints, robots' heterogeneity, and the unreliability of sensors and actuators, along with the search for an optimal allocation [9]. In the most general scenario, the solution space for task allocation increases exponentially with the number of robots and tasks. For T number of tasks to be allocated into R robots, there are R^T possible allocations. However, relevant practical constraints can significantly narrow this solution space. These constraints include spatial limitations, the maximum number of allocated tasks per robot, and the specific capabilities of the robots dictating the types of tasks they can undertake. For example, suppose only one robot is capable of performing pick-and-place operations. In that case, all such tasks will be assigned to that single robot, narrowing the solution space to just one possibility. We presented the KANOA DSL for specifying constraints relevant to the task allocation problem. This chapter examines how they can be utilised to generate effective multi-robot task allocations.

The diversity of variants in the definition of the task allocation problem has opened the door to a vast variety of solutions, some of which are described in Section 3.3 and Table 3.4. That table mentions, for example, a study that compares multiple auction-based and optimisation-based techniques for solving different variants of the task allocation problem [41]. Most solutions proposed for the task allocation problem are optimisation-based techniques, such as the *min-max* algorithm [95], a centralised optimisation algorithm that aims to allocate a set of limited resources among a group of robots in a way that minimises the maximum cost of a variable (e.g., the total travelling cost) over a set of possible allocations. This is a centralised approach that requires a central authority to allocate resources based on a predetermined set of rules and constraints. In contrast, *auction-based* allocation is a decentralized approach that involves entities bidding for resources in an auction. In this auction, the entities compete with each other by offering prices or bids for the resources they need. The resources are then allocated to the highest bidder, and the process is repeated until all resources are allocated. Another technique is the *multi-travelling salesman problem*, where there are multiple robots, each of which must visit a set of task locations exactly once and return to their starting point such that the distance travelled by all robots combined is minimised.

As detailed in the literature review from Chapter 3, existing algorithms cannot provide formal guarantees that the resulting allocations comply with the types of mission requirements considered in this thesis. This is a problem especially when there is a large number of constraints or the mission constraints are too complex for a human to determine whether the allocation fulfils all of them. A solution to this *satisfiability problem* is the use of SAT solvers, model checkers and constraint solvers to generate allocations complying with such set of requirements.

In this chapter, we introduce the KANOA task allocator, a new approach to solving the task allocation problem for a group of heterogeneous robots under a set of constraints, while assuring that the solutions comply with the mission requirements. We use a model generator for the generation of potential allocations, and a SAT solver for the analysis of the allocations. These engines are part of the Alloy Analyzer, previously introduced in Section 2.2. Therefore, we use the Alloy language for the description of the allocation problem and the Alloy analyzer for the generation and analysis of solutions.

This chapter is divided as follows. Section 5.1 introduces the KANOA task allocation problem. Section 5.2 describes a pre-allocation stage for the pre-processing of the problem specification. Sections 5.2 and 5.3 present pre-allocation and allocation components of the KANOA task allocator, respectively. Section 5.4 illustrates the application of the KANOA allocation to the hospital case study. The advantages and limitations of the KANOA task allocation are then discussed in Section 5.5. Finally, Sections 5.6 and 5.7 review related work and summarise the chapter, respectively.

5.1 Task Allocation Problem

In Chapter 4, we used the Z notation to define the problem specification. This section formalises the task allocation problem using the Alloy language [55], a declarative language inspired by the Z notation and previously introduced in the background part of the thesis (Section 2.2). The Alloy language is a formal modelling language for software design that is based on set theory and relational theory, and shares similarities with the Z notation. In [161], Bolton shows how to use the Alloy Analyzer to verify data specified in the Z notation. We follow this approach to generate a model in the Alloy language that can be used as an input for the task allocator. In other words, given a problem defined in the KANOA DSL, we can transform it into an Alloy language model that can be fed into the task allocator.

In the previous chapter, we captured the MRS mission allocation problem by means of a formal specification divided into: the *world model*, the *tasks model*, the *robots model* and the *mission*, each of which was formalised in the Z notation. To solve the task allocation problem, we extract from this specification the information relevant to this problem and convert it to an Alloy model similar to the Z specification. We are interested in finding allocations of tasks to robots that comply with the following types of constraints (introduced in Section 4.2):

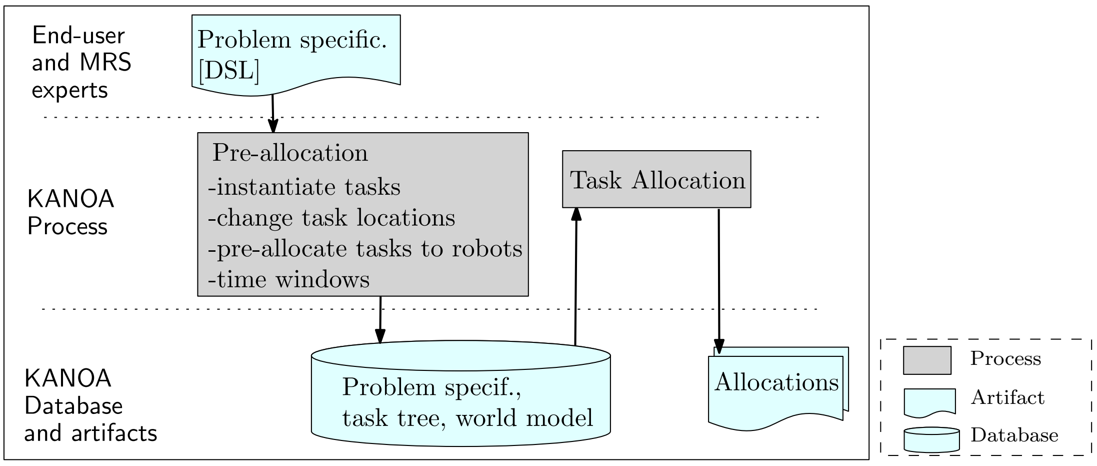


Figure 5.1: Overview of the KANOA workflow from the specification of the problem to the generation of task allocations.

- (C1) **spaceXY**. Restricts the physical working space of one or more robots.
- (C2) **allocateT**. Enforces the allocation of a task to a specific robot.
- (C3) **closest**. Enforces the allocation of a task to the closest robot.
- (C4) **maxTasks**. Bounds the maximum number of tasks allocated to a robot.

We note that the allocation must assign a task to more than one robot if this task is defined as a *joint task*.

The steps followed by KANOA for the generation of allocations from the problem specifications defined by end-users and MRS domain experts are shown in Figure 5.1. First, a pre-allocation step creates the *tree*-like structure containing *task instances* and saves the rest of the information from the problem specification into KANOA's database. This database is then accessed by the task allocator in a second step, to generate feasible allocations of tasks to robots as explained in the next section.

We define two new entities necessary for the allocation of tasks to robots: atomic and compound task *instances*. A robotic mission may require performing a task more than once, e.g., at different locations. The mission example in Section 4.3.2 requires the robots to complete two mission tasks $M1, M2 : \mathit{MissionTask}$.[1] Both mission tasks require the compound task with identifier $\mathit{ct2\_patient} : \mathit{CTaskID}$ to be performed ($M1.\mathit{taskID}$, $M2.\mathit{taskID} = \mathit{ct2\_patient}$) at different locations. To support this, we define *instantiation* of compound or atomic tasks for a given problem specification p,

$$\mid \; p : \mathit{ProblemSpecification}$$

Definition 5.1.1 (Atomic task instance) *An instance of an atomic task from the problem specification p is defined by the data type AtomicTaskInst, which comprises a type that*

[1] *CompoundTask, AtomicTask, Mission*, etc. are defined in the *problem specification* in Section 4.2.

maps the instance of the atomic task to its uninstantiated counterpart, a unique id of type ATInstanceID, the number of robots nRobots required to execute the task (which has the value type.nRobots), and locationID, the identifier of the location where the instantiated task has to be performed.

$[\mathit{ATInstanceID}]$

$\mathit{AtomicTaskInst}$

$\mathit{type} : \mathit{AtomicTask}$
$\mathit{id} : \mathit{ATInstanceID}$
$\mathit{nRobots} : \mathbb{N}$
$\mathit{locationID} : \mathit{LocationID}$

$\mathit{type} \in p.\mathit{taskModel}.\mathit{atomicTasks}$
$\mathit{nRobots} = \mathit{type}.\mathit{nRobots}$

We note that an atomic task instance $\mathit{at} : \mathit{AtomicTaskInst}$ may have a different location than the uninstantiated task, i.e., $\mathit{at}.\mathit{locationID} \neq \mathit{at}.\mathit{type}.\mathit{locationID}$ may hold. Compound task instances are defined similarly.

Definition 5.1.2 (Compound task instance) *An instance of a compound task from the problem specification p is defined by the data type CompoundTaskInst, which comprises the non-instantiated task (type), a unique identifier id, and the sequence subtasks of identifiers for its instantiated subtasks. As an example, the subtasks for the instantiated version of an uninstantiated compound task with subtask IDs [at*1*, ct*2*] may be given by [at1_1, ct2_*1*].*

$[\mathit{CTInstanceID}]$

$\mathit{TaskInstID} = \mathit{ATInstanceID} \cup \mathit{CTInstanceID}$

$\mathit{CompoundTaskInst}$

$\mathit{type} : \mathit{CompoundTask}$
$\mathit{id} : \mathit{CTInstanceID}$
$\mathit{subtasks} : \mathrm{seq}\ \mathit{TaskInstID}$

$\mathit{type} \in p.\mathit{taskModel}.\mathit{compoundTasks}$

Instances of (atomic and compound) tasks are obtained from mission tasks. Let $\mathit{mission} \in \mathit{Mission}$ be the mission specification, where $\mathit{mission}.\mathit{tasks}$ is the set of tasks to be performed by the robots; we define a task instance for every $t \in \mathit{mission}.\mathit{tasks}$. If t is a compound task, its subtasks are also instantiated recursively until all descendants are instantiated[2]. We refer to the set of all atomic task instances as *AtomicTaskInst*. This is the set of tasks to be allocated to robots..

Continuing, we define a *feasible problem specification.* This definition is required for the task allocation problem to have a solution (i.e., a non-empty set of allocations.

Definition 5.1.3 (Feasible problem specification) *A problem specification PS is called feasible if there exist (a) all robots in the required number (for joint tasks) (c) that can travel to the respective task locations and (b) with the necessary capabilities to complete all tasks defined as part of the mission.*

We now define an allocation as follows.

Definition 5.1.4 (Allocation) *Given a feasible problem specification PS s.t. the robot capabilities are available to complete all tasks in the problem specifications, the number of robots... s.t. an allocation can be synthesised An allocation is a partial function that maps robots (from the problem specification p) to the sets of tasks they are assigned to complete.*

$$
\begin{array}{|l}
\mathit{alloc} : \mathit{Robot} \nrightarrow \mathbb{P}\, \mathit{ATInstanceID} \\
\hline
\operatorname{dom} \mathit{alloc} \subseteq p.\mathit{robotsModel} \\
\bigcup\{r : \operatorname{dom} \mathit{alloc} \bullet \mathit{alloc}\; r\} = \mathit{AtomicTaskInst} \\
\forall (r, t) \in \mathit{alloc} \bullet \exists\, a : \mathit{AtomicTaskInst} \bullet (a.\mathit{id} = t \land \\
\qquad\qquad\qquad\qquad a.\mathit{type}.\mathit{id} \in \bigcup\{c : r.\mathit{capability} \bullet c.\mathit{task}\})
\end{array}
$$

Note that *alloc* is a partial function as some robots may not be deployed, even though they are declared in the problem specification *p*.[3] Tasks can only be assigned to robots that possess the capability to perform them, as expressed by the last predicate from the definition.

Finally, we define the allocation problem as follows.

Definition 5.1.5 (Allocation Problem) *The allocation problem consists of finding allocations alloc that satisfy the space constraints (spaceXY), the task pre-allocation constraints (allocateT), the constraints to assign tasks to the nearest robots (closest), and the limits on the number of tasks assigned to a robot (maxTasks) declared in the problem specification.*

The closest distance is considered from the initial state of the MRS as defined in the world model of the KANOA problem specification[4]. In the following paragraphs, we show

[2]This is explained later with an example in Figure 5.2.

[3]See problem specification in Section 4.2.6.

[4]Modifying the closest distance from the last visited location is left for further work.

the Z schemas defined so far on the left and their counterparts in the Alloy language in a *box* on the right. For an introduction to the Alloy language, please refer to the background Section 2.2.

Problem specification

Robots

A robot from KANOA's robot model is defined in the Z notation by an identifier (*id*), an initial location (*initLocation*), its average velocity (*velocity*) and its set of capabilities (*capabilities*). In Alloy (we refer to the Alloy language simply as Alloy when it is clear we are referring to the language and not the Alloy Analyzer tool), we declare an **abstract sig**nature for the robot with the relation *hascapability* mapping a robot to the set of capabilities that the robot possesses. An abstract signature (**abstract sig**) is analogous to a class in object-oriented programming, where instances of these classes are declared as signatures (**sig**) instantiating (**extends**) abstract signatures.

We define each of the robots as **sig**natures with identifiers matching their Z notation counterpart. In the following box on the right, we show robot $r1$ as an example. The multiplicity operator **lone** states that each of the robot's instances must appear at most once. This allows the robot to appear in an allocation, and in others to be removed if there are enough robots that can complete the mission without its help.

Robot

$id : RobotID$

$initLocation : LocationID$

$velocity : \mathbb{N}$

$capability : \mathbb{P}_1\ Capability$

abstract sig *Robot*{
  hascapability : **set** *Capability*
}...
//*example* :
lone sig $r1$ **extends** *Robot*{}
  {*disj*[*hascapability*, *Capability*−
  $r1at2\_floor - r1at3\_sanit-$
  $r1at4\_notify$]}
...

The *disj*oint fact defined within the signature,

{*disj[hascapability, Capability- r1at2_floor - r1at3_sanit- r1at4_notify]*}

states that robot $r1$ has capabilities r1at2_floor, r1at3_sanit, and r1at4_notify from the set of all capabilities, *Capability*. As shown later in this section, the robot's initial location *initLocation* is used in the description of the mission constraints, while the *velocity* is used later for the scheduling of tasks in Section 6.

Capabilities

The capabilities are declared as abstract signatures with a relation *do* mapping to the set of atomic tasks that can be performed with this capability. The relation *do* maps to a set because a capability can be used to do one or more atomic tasks.

There are three **fact**s related to the capabilities. The first fact states that all capabilities that appear as part of the solutions must be assigned to exactly one robot; i.e., each robot has its disjoint set of capabilities. As any robot r has a relation *hascapability* mapping to a set of capabilities $r.hascapability$, this can be thought of as multiple relations between one robot and multiple capabilities. We can access the robot from which a capability c originates by the inverse relation $hascapability.c$. By defining the parity of $hascapability.c$ to one, all capabilities appearing in the solutions are constrained to belong to one robot,

```
fact{all c: Capability | #hascapability.c=1}
```

The second fact states that "all robots appearing in the allocation must have assigned tasks", where *do* is the mapping function from capabilities to tasks,

```
fact{all r: Robot | #r.hascapability.do>0}
```

```
Capability
task : ATaskID
time : ℕ
succRate : ℕ
─────────────
0 < succRate ≤ 100
```

```
abstract sig Capability{
    do : set AtomicTask
}
fact{all c : Capability |
    #hascapability.c = 1}
fact{all r : Robot |
    #r.hascapability.do > 0}
fact{all c : Capability |
    #c.do > 0}...
//example :
lone sig r1at2 extends
    Capability{}
    {all d : do | d in at2}
...
```

Similarly, the third fact states that all capabilities that appear in the solutions must have assigned tasks,

```
fact{all c: Capability | #c.do>0}
```

Complementary to this section, Appendix C shows an example of what would happen if any of these three facts were removed.

The capabilities for each of the robots are defined explicitly in Alloy as signatures. For example, if robot $r1$ has the capability of performing task $at2$, we create a signature named $r1at2$ as an extension of the abstract signature *Capability*. The keyword **lone** means that an instance of the signature can appear in the solution at most once. If it does not appear in the model, it means that robot $r1$ is not using the capability $r1at2$ for the given solution. The time to complete a task (*time*) and the rate of success (*succRate*) from the Z specification are used later for the scheduling of tasks in Section 6.

Observation 1. Note that the last signature is equivalent to the less idiomatic solution removing the *lone* operator from "**lone sig** r1at2..." and adding "**fact**{#r1at2<=**1**}" constraining the appearance of $r1at2$ to be less than two. However, Alloy generated duplicated solutions when this alternative approach was used. This type of puzzling behaviour is briefly discussed in Jackson's "Software Abstractions" book on Alloy [162] (Section 4.5.1 on *facts*, pages 120-121). This is likely because adding the multiplicity operator *lone* removes the need to generate more than one atom of a signature from the beginning, at the creation of the system model.

Atomic tasks

Atomic tasks are defined as **abstract sig**natures with coordinates (x,y) mapping to the set of **Int**egers. For each atomic task in the Z specification, an abstract signature in the Alloy model is defined. Each of these abstract signatures has a constraint on the number of robots required to achieve *do* this task. The relation *do* is declared in the capability's signature, mapping a capability to an atomic task.

```
__AtomicTask________________
| id : ATaskID
| nRobots : ℕ
| locationID : LocationID
|_____________
| nRobots ≥ 1
|____________________________
```

```
abstract sig AtomicTask{
      x : one Int,
      y : one Int,
}...
//task example :
abstract sig at2
      extends AtomicTask{}
fact{all a : at2 | #do.a = 2}...
//instance task example :
one sig at2_1 extends at2{}
      {x = 4 y = 1}
one sig at2_2 extends at2{}
      {x = 10 y = 1}
...
```

For example, if there is an atomic task with identifier *at*2 that requires two robots to be completed, we declare,

abstract sig at2 **extends** AtomicTask {}

fact{**all** a:at2 | #do.a=**2**}

where the relation *do* is defined in the capabilities signature; and *do.a* returns the capability (or capabilities) of the robot (or robots) that task *at*2 is assigned to. Then, its arity is set to two (#do.a=**2**).

Finally, we create **sig**natures for each instantiated atomic task. We declare that they must appear once by using the **one** quantifier operator. The (x,y) coordinates of the locations computed by the pre-allocation stage are added as a constraint when declaring the atomic task instances. For example, in the right box above, there are two atomic task instances *at*2_1 and *at*2_2 of type *at*2, each requiring two robots to be completed at locations (4,1) and (10,1), respectively.

Mission constraints

There are six types of mission constraints specified in KANOA (see Section 4.2). In the allocation of tasks, only a subset of these is relevant. For example, if there is a spatial constraint where a robot *r*1 can only move within an area, the allocator must only allocate tasks inside this constrained area. The rest is used for the scheduling of Task in Chapter 6.

The mission constraints that impact the task allocation process are: (a) `spaceXY`, constrains the working space of a robot; (b) `allocateT`, allocates directly a task to a specific robot; (c) `closest`, allocates a task (or all tasks) to the closest robot(s); and (d) `maxTasks`, limits the number of tasks that can be assigned to a robot.

In this section, we show the specification of mission constraints in Alloy language in a generic form, where information that is retrieved from KANOA DSL is depicted in **bold** font.

• `spaceXY`. *spaceXY* restricts the working area of a robot. For every relation (*robotID*, *xy*, *glt*) $\mapsto v$ defined in Z to specify that a robot has a constraint on coordinate x or y ($xy \in XY$) to be greater or lower than ($glt \in GLT$) a given number ($v \in \mathbb{N}$), we define an Alloy fact as shown in the following box,

Constraints

$spaceXY :$
$\quad \mathbb{P}((RobotID \times XY \times GLT) \times \mathbb{N})$
$\ldots$

fact{ **all** *r* : ***robotID*** |
  all *c* : *r.hascapability* |
  all *do* : *c.do* | *do*.***xy*** ***glt*** ***v***}

Example 1. For a mission constraint: "restrict robot $r6$ to only work within an area where the y axis is *greater* than 10 units", defined in KANOA as,

robot r6 **work in y greater than** 10

and in the Z notation as (r6,y,greater)↦10, the following fact is encoded in the Alloy language,

fact{**all** r:r6 | **all** c:r.hascapability | **all** do:c.do | do.y > 10}

• `allocateT` and `allocatemT`. Constraints of type `allocateT` allocate a compound or atomic task to a robot, while `allocatemT` allocates a mission task to a robot. In the pre-allocation stage (described in Section 5.2), we parse these constraints and associate each such atomic task instance with the robot(s) responsible for its completion. Therefore, within our Alloy model, it is necessary to incorporate a fact for each of these pre-allocated atomic tasks (i.e., atomic tasks with robot associations). For any atomic task instance, we define the relation mapping the task into the robot capabilities (do) to be in the set of a specific robot (d **in** ***robotID***.*hascapability*).

Constraints

$allocateT : \mathbb{P}(TaskID \times RobotID)$
$allocatemT : \mathbb{P}(MissionTaskID$
$\quad \times RobotID)$
…

fact {**all** at : ***taskInstanceID*** |
 one d : $do.at$ |
 d **in** ***robotID***.*hascapability*}

Example 2. For a mission constraint "allocating atomic tasks $at1$ to robot r4",

allocate atomic task at1 **to robot** r4

given two instances ($at1\_1$ and $at1\_2$) required to accomplish the mission, the Alloy model includes two additional facts,

fact {**all** at: at1_1| **one** d: do.at | d **in** r4.hascapability}
fact {**all** at: at1_2| **one** d: do.at | d **in** r4.hascapability}

• `allocateTtogether` and `allocatemTtogether`. Similar to the last constraint, this constraint allocates a task to a single robot. However, we do not provide the robot so this can change from allocation to allocation (as long as the selected robot possess the necessary capabilities). In the fact on the left, $\boldsymbol{at_i}$ is of type ***taskInstanceID***. The Alloy model includes a fact for every reachable atomic task, so that the allocated robot (reachable through the inverse relation from the task to the *capability* followed by the inverse relation *do*) is equal to the robot for the next atomic task.

Constraints

$allocateTtogether : \mathbb{P}\ TaskID$

$allocatemTtogether : \mathbb{P}\ MissionTaskID$

...

fact $\{capability.do.\boldsymbol{at_1}$
$= capability.do.\boldsymbol{at_2}\}$
fact $\{capability.do.\boldsymbol{at_2}$
$= capability.do.\boldsymbol{at_3}\}$
...

Example 3. For the following mission constraint, where $m3$ consists of reachable atomic tasks at1_1, at1_2 and at1_3

`allocate mission task` m3 `to single robot`

the Alloy model contains the facts,

fact {capability.do.at1_1= capability.do.at1_2 }
fact {capability.do.at1_2= capability.do.at1_3 }

• `closest` and `closestmT`. Constraints of type `closest` allocate compound or atomic tasks to the closest robots, while `closestmT` allocates a mission task to the closest robots. Similar to the last constraints, `allocateT` and `allocationmT`, these constraints were parsed by the pre-allocation stage with the required robot(s) associated with the atomic task instances, if applicable. Hence, in Alloy, these two constraints, `closest` and `closestmT`, were already added as part of the last ones by defining facts for atomic task instances with an assigned robot. The Z schema does not contain a robot field as this is computed during the pre-allocation stage.

Constraints

$closest : \mathbb{P}(TaskID)$

$closestmT : \mathbb{P}(MissionTaskID)$

...

fact $\{$**all** $at : \boldsymbol{taskInstanceID} \mid$
  one $d : do.at \mid$
  d **in** $\boldsymbol{robotID}.hascapability\}$

Remember that the order matters when specifying tasks of the type *allocateT*, *allocatemT*, *closest* and *closestmT*. As explained in Section 5.2, if more than one of these constraints is specified and they affect the same atomic task instance, the last overrides the previous ones.

Example 4. For a mission constraint "atomic task *at*2 to the closest robot"

`allocate atomic task` at2 `to closest robot`

given three instances of *at*2 (*at*2_10, *at*2_15 and *at*2_20) required to accomplish the mission, the Alloy model introduces three additional facts,

```
fact {all at: at2_15| one d: do.at | d in r3.hascapability}
fact {all at: at2_20| one d: do.at | d in r3.hascapability}
fact {all at: at2_10| one d: do.at | d in r2.hascapability}
```

where robots *r*2 and *r*3 appearing for task *at*2 instances were found to be the closest.

Observation 2. Tasks already assigned to robots within the mission specification are included in the task allocation process for two reasons. First, these predetermined allocation constraints may conflict with other constraints, and we want Alloy to check for such conflicts; a simple example is when the problem specification assigns six tasks to robot *r*1, but there is a mission constraint preventing any robot from taking more than five tasks in which case Alloy won't find any possible solution. Second, we would like the diagrammatic representation of the allocations found by Alloy to include every task (see Figure 5.3) for explainability purposes. Additionally, to assess the Alloy Analyzer execution time increase when pre-allocated tasks are added to the model, we perform some experiments discussed in the evaluation of KANOA's tool in Section 7.2.4 showing that the increase in computation time is linear and *relatively* small compared to the number of tasks added.

• `maxTasks`. Constraints of the type `maxTasks` set the maximum number of tasks that a robot can accept. Hence, for every constraint in Z of the form $maxTasks(robotID) = v$, we add a `fact` constraining the arity of the transitive relation from robot *robotID* to its allocated tasks (*hascapability.do*) to a maximum number $v \in \mathbb{N}$.

Constraints

$maxTasks : \mathbb{P}(RobotID \nrightarrow \mathbb{N})$

...

fact{**no** r : $\boldsymbol{robotID}$ |
$\#(r.hascapability.do) > \boldsymbol{v}$}

Example 5. For a mission constraint "limiting the number of tasks that robot *r*1 can accept to 4",

`limit max number of tasks in robot` r1 `to` 4

the following fact is added to the Alloy file,

```
fact { no r:r1 | #(r.hascapability.do ) > 4}
```

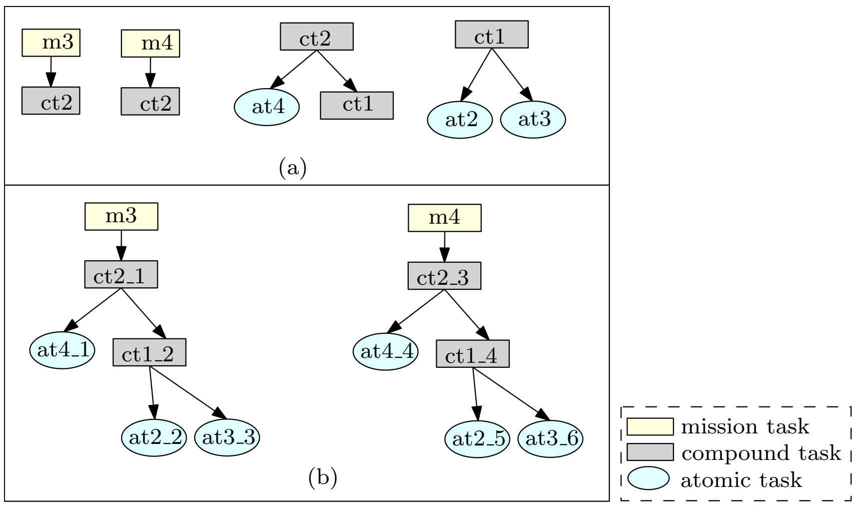


Figure 5.2: (a) Visualisation of task missions $m3$ and $m4$, each defined as performing task $ct2$; compound task $ct2$ is defined in terms of tasks $at4$ and $ct1$, while task $ct1$ is defined as doing $at2$ and $at3$. (b) Instantiation of all tasks needed to be completed done in the *pre-allocation* stage.

We have covered the different constraints applicable to the task allocation problem from KANOA's specifications. As we devise the description of the allocation problem in a declarative language, new mission constraints can be added easily into the Alloy model. For example, imagine there may be some allocations where a robot $r1$ is not assigned any task as its peers can complete the mission but we want to enforce deploying this robot for the mission as we know beforehand that it is more suitable for the job. This new constraint would be added as a new **fact** in the Alloy model. A second example is forcing the allocator to deploy all robots to help balance the distribution of tasks among them and ensure that no robots are left without tasks.

5.2 Pre-allocation

Once the user has specified the robotic problem in KANOA DSL as described in Section 4.2.1 (i.e., the *world model, task model, robots model* and *mission specification*), the pre-allocation parses the data and processes it through four phases. Phase 1 generates *instances* of all the necessary tasks to complete the whole mission. Next, phase 2 changes the location of the atomic tasks instances reachable from *mission tasks* that were defined with a location, and phase 3 pre-allocates atomic tasks to robots defined from any task specified in the *mission constraints* of the type `allocateT` and `closest`. Finally, phase 4 sets time windows (start and/or end time) of atomic tasks affected by *mission constraints* of the type `taskTime`. We show this process with the following example.

Example 6. Consider a scenario where the mission consists of two tasks: cleaning patient rooms *room*2 and *room*3 (as described in the hospital case study introduced in Section 4.3). These tasks are declared as *mission tasks*, *m*3 and *m*4, respectively, defined in KANOA DSL as,

```
m3 : compound task ct2 at location room2,
        m4 : compound task ct2
```

where *ct*2 is the task of cleaning a patient room, composed of multiple subtasks; first, notifying the patient (*at*4) that the robot is about to clean the room, followed by two tasks, cleaning the floor (*at*2) and sanitising the room (*at*3), that are declared as a single compound task *ct*1. The visual representation of these tasks is depicted in Figure 5.2a.

Phase 1. To accomplish *m*3 and *m*4, we first create instances of all necessary compound tasks, and atomic tasks that are to be allocated to the robots. Figure 5.2b shows the four compound tasks (*ct*2_1, *ct*1_2, *ct*2_3, *ct*1_4) and five atomic tasks (*at*4_1, *at*2_2, *at*3_3, *at*4_4, *at*2_5, *at*3_6) required for the missions.

Phase 2. We set the locations of atomic task instances [*at*4_1, *at*2_2, *at*3_3] to *room*2 by computing the leaf nodes of mission *m*3. To that end, we first set the locations of [*at*4_4, *at*2_5, *at*3_6] to the ones specified in the Task Model as *m*4 has no specified location overriding the default location for its tasks.

Phase 3. Consider the following two mission constraints allocating tasks to robots,

```
allocate compound task ct1 to closest robot,
   allocate mission task m3 to robot  r3
```

we first pre-allocate compound task *ct*1 subtask instances [*at*2_2, *at*3_3, *at*2_5, *at*3_6] to the closest robot. Then we assign subtasks of mission task *m*3 [*at*1_1, *at*2_2, *at*3_3] to robot *r*3. Hence, only task *at*4_4 is not pre-allocated to any robot. Notice that the order of these constraints matters.

Phase 4. Finally, the mission also has two constraints: (i) all patients must be notified (task at4) that their room is going to be cleaned before 50 time units, and (ii) all patient rooms must be cleaned before the end of the day at 110 time units.

```
compound task ct2 has end time: 110,
  atomic task at4 has end time 50
```

As such, we obtain all *ct*2 reachable atomic task instances, which, in this case, are the complete set ($at4\_1, at2\_2, at3\_3, at4\_4, at2\_5, at3\_6$) and add an *endtime* attribute for each element with the value of 110. Finally, we get the instances of the atomic task *at*4 ($at4\_1, at4\_4$) and set this attribute to 50.

Phase 3 creates two new types of objects, atomic and compound task instances. Phase 4 is performed in the pre-allocation stage although its result is used for the scheduling of tasks in Section 6. The algorithms used for the pre-allocation are described in the following paragraphs

Pre-allocation algorithms. Here we describe the algorithms used in the pre-allocation stage—these are automated as part of the KANOA tool. In the first phase (Algorithm 1), we instantiate each of the required atomic and compound tasks By *instantiation* we mean that each required task is defined with a unique identifier comprising: (a) the task identifier defined from the problem specification, (b) the underscore character, (c) a unique number. A counter (`counterAT`) is used to assign a unique number to the set of atomic tasks, and a second counter (`counterCT`) for the set of compound tasks.

We start the algorithm Instantiation retrieving the set of mission tasks (`mission Tasks`) from the problem specification defined in Kanoa DSL. If the mission task is to perform an atomic task (`mt.at`), we simply add the instance of the atomic task (`instantiatedTask.add()`). If the mission task is to perform a compound task (`mt.ct`), we instantiate each subtask by calling the function InstantiateSubtasks. It first adds the compound task instance, then generates instances of all its subtasks, and finally adds the instantiated subtasks one by one, adding the subtasks of any compound task recursively. The InstantiateSubtasks function is self-explanatory. It creates a new list `isubtasks` to add all subtasks of the compound task `ct`. Then it adds itself and all subtasks instantiated (lines 12, 16 and 20). For subtasks that are compound tasks are recursively instantiated (lines 22-28).

Algorithm 1 correctness. The functions Instantiation and Instantiate-Subtasks terminate and set the global variables counterAT,counterCT and instantiatedTask values when completed. It is linear in the number of mission tasks as: (i) the mission task consists of a single atomic task or (ii) the

Algorithm 1 Pre-allocation Phase 1 - Instantiate atomic and compound tasks required.

```
Require: Problem specification in Kanoa DSL parsed.
Require: global var counterAT,counterCT=0;  ▷ atomic and compound tasks counters
Require: global var instantiatedTask= new List<String>            ▷ instantiates tasks
 1: procedure INSTANTIATION(missionTasks)  ▷ Instantiate atomic & compound tasks
 2:     for mt in missionTasks do
 3:         if mt.at                              ▷ mission requires do to atomic task
 4:             counterAT+=1
 5:             instantiatedTask.add( mt.at.id+"_"+ str(counterAT) )          ▷ add task
 6:         else if mt.ct                      ▷ mission requires to do compound task
 7:             counterCT+=1
 8:             INSTANTIATESUBTASKS(mt.ct, counterCT)          ▷ instantiate subtasks
 9:         end if
```

Instantiate compound task and its subtasks.

```
10: function INSTANTIATESUBTASKS(ct , numOfTask)
11:     isubtasks = new List<String>                   ▷ ordered instantiated subtasks
12:     instantiatedTask.add( ct.id+"_"+ str(counterAT) )                   ▷ add task
13:     for subtask : ct.subtasks do
14:         if subtask is atomic task
15:             counterAT+=1
16:             isubtasks.add(subtask.id+"_"+ str(counterAT))          ▷ task instance
17:         end if
18:         if subtask is compound task
19:             counterCT+=1
20:             isubtasks.add(subtask.id"_"+ str(counterCT))           ▷ task instance
21:         end if
22:     for i:isubtasks do
23:         if subtask is atomic task
24:             instantiatedTask.add(i)                                       ▷ add task
25:         end if
26:         if subtask is compound task
27:             INSTANTIATESUBTASKS(task type of i, num assigned to i)
28:         end if
```

Algorithm 2 Pre-allocation Phase 2 - Overwrite location of tasks.

```
1: procedure CHANGELOCATION(missionTasks) ▷ Change location of tasks if added in
   mission task
2:     for mt in missionTasks do
3:         if mt.loc                                    ▷ mission has location
4:             a = GET_ ATOMIC_INST(mt)      ▷ atomic tasks from mission task
5:             CHANGELOCATION(a,mt.loc)     ▷ change location of each atomic task
6:         end if
```

mission consists of a finite number of compound tasks with a finite number of subtasks (see Section 4.2.1).

In the second phase (Algorithm 2), we deal with task missions defined by a location, for example, m1 : **compound task** ct1 **at location** room1. Any mission task that is specified with a location overwrites the location of any *reachable* atomic task. By *reachable* tasks we mean the task (atomic or compound) defined in the mission task, as well as any descendant of its compound task (see Task model in Section 4.2.1). To shorten the explanation, Algorithm 2 shows that for every mission task `mt`, if a location is provided `mt.loc`, we get the (previously instantiated) list of reachable atomic tasks, a=GET_ATOMIC_IINST(mt), and change the location on each of the tasks in the list to the one provided in the mission task.

Algorithm 2 correctness. The function CHANGELOCATION terminates as there is a finite number of mission tasks and atomic task instances reachable from each mission task. It is linear in the number of mission tasks defined with a location, and the number of reachable atomic tasks from each of these mission tasks (see Section 4.2.5).

Within the KANOA tool, we define the data type of mission constraints as `con`, with attribute `type` defining the type of mission constraint. It can take values: `allocateMT`, `allocateAT` or `allocateCT` meaning to allocate a mission, atomic or compound task, respectively; `closestRobotMT`, `closestRobotAT`, `closestRobotCT` and `closestRobotAll` assign a mission, atomic, compound task or all atomic tasks to the closest robot; and `taskTimeMT`, `taskTimeAT` and `taskTimeCT` define the time to start of end a mission task, atomic task or compound task, respectively.

In the third phase (Algorithm 3), tasks are assigned to robots if there exists any constraint of the type `allocateT` or `closest`. Algorithm 3 shows that each constraint is read in the order they appear in the problem specifications (line 2), hence, the order must be considered when defining mission constraints of these two types. It first checks the type of mission constraint (line 3). Then, it gets the list of instantiated atomic tasks reachable from the task defined in the mission constraint (GET_ATOMIC_INST); and

Algorithm 3 Pre-allocation Phase 3 - Allocated tasks from mission constraints.

```
 1: procedure AssignTasks2Robot(missionConstraints)          ▷ Assign robots to tasks
 2:     for con in missionConstraints do
 3:         switch con.type do
 4:             case allocateMT                                ▷ mission task to robot
 5:                 Assign2Robot(Get_ Atomic_Inst(con.mt) , con.robot)
 6:             case allocateAT                                ▷ atomic tasks to robots
 7:                 Assign2Robot(Get_ Atomic_Inst(con.at) , con.robot)
 8:             case allocateCT                                ▷ compound tasks to robot
 9:                 Assign2Robot(Get_ Atomic_Inst(con.ct) , con.robot)
10:             case closestRobotMT                            ▷ mission task to closest robots
11:                 Assign2ClosestRobot(Get_ Atomic_Inst(con.mt),con.robot)
12:             case closestRobotAT                            ▷ atomic tasks to closest robots
13:                 Assign2ClosestRobot(Get_ Atomic_Inst(con.at) , con.robot)
14:             case closestRobotCT                            ▷ compound tasks to closest robots
15:                 Assign2ClosestRobot(Get_ Atomic_Inst(con.ct) , con.robot)
16:             case closestRobotAll                           ▷ all atomic tasks to closest robots
17:                 Assign2ClosestRobot(Get_All_AT_Inst() , con.robot)
18:         end switch
```

lastly, it assigns each of the tasks in the list to a given robot (Assign2Robot) or the closest robot (Assign2ClosestRobot). In KANOA's tool, if a robot does not have the capability to complete an atomic task allocated through these mission constraints, an error will appear. Notice that atomic and compound tasks can be part of multiple other tasks. Task constraints of these types modify all the respective task instances. This gives the versatility to define very intricate allocations from the beginning and capture **expert knowledge** when, for example, we know that there is a set of robots that is more suitable to perform specific task types.

Lastly, the fourth phase assigns starts and end times to atomic task instances (Algorithm 4). It first checks for constraints of type related to modifying the task time (taskTimeMT, taskTimeAT, taskTimeCT), then obtains all atomic instances reachable from the respective task (Get_Atomic_Inst). Finally, it assigns a start (lines 16) or end time 18) to each atomic task instance.

Algorithms 3 and 4 correctness. The functions AssignTasks2Robot and AssignStartEndTime terminate because they iterate over a finite number of constraints, missions, atomic tasks and compound task instances. Missions can only contain tasks of type atomic or compound. Atomic tasks do not have subtasks, while compound tasks are limited to a finite number of subtasks and cannot contain themselves (either directly or through a transitive relationship,

Algorithm 4 Select initial or final state of atomic tasks instances constrained by mission constraints of the type `taskTime`.

```
1: procedure AssignStartEndTime(missionConstraints)    ▷ Assign start and/or end time
2:     for con in missionConstraints do
3:         switch con.type do
4:             case taskTimeMT                          ▷ mission task
5:                 at = Get_ Atomic_Inst(con.mt)
6:                 SetStartEndTime(at, con)
7:             case taskTimeAT                          ▷ atomic task
8:                 at = Get_ Atomic_Inst(con.at)
9:                 SetStartEndTime(at, con)
10:            case taskTimeCT                          ▷ compound task
11:                at = Get_ Atomic_Inst(con.ct)
12:                SetStartEndTime(at, con)
13:        end switch
14: procedure SetStartEndTime(at, con)
15:     for a in at do
16:         if con.end
17:             Assign_Endtime(a , con.robot)
18:         else con.start
19:             Assign_Startime(a , con.robot)
```

as defined in Section 4.2.3). Therefore, constraints related to compound tasks also terminate.

In this pre-allocation stage, we translated high-level constraints from missions and compound tasks into individual tasks, and instantiated these atomic tasks if necessary. The complexity of these pre-allocation algorithms is linear in the size of the KANOA problem specification described in Section 4.2.6. The next section describes how to allocate the atomic tasks into the set of available robots, or a subset of those.

5.3 Task Allocation in KANOA

We defined the task allocation problem using the Alloy language in Section 5.1. KANOA uses the Alloy Analyzer [55] as a task allocator to generate multiple allocations of tasks to robots, where each allocation complies with the requirements from the problem specification.

5.3.1 Task Allocation Generation

This section describes the parts missing to complete the Alloy model that can be provided to the Alloy Analyzer to find models that satisfy the task allocation problem specification. These parts are the set of integers and the commands to run the model (the *predicate* and *run* commands). For the latter, the Alloy Analyzer requires a finite *scope* to bound the maximum size of each of the signatures, i.e., the depth to which Alloy Analyzer searches for solutions. The computation of the different scopes is described within this section.

We proceed by explaining each of the missing statements in the Alloy model,

Integers. As we are using integers in the Alloy model (for example, to define the coordinate x where a task must be done), we must explicitly import the set of integers by adding the instruction,

```
open util/integer
```

Scope. In Alloy, the *scope* defines the number of *atoms* to create in the *Alloy universe*. For example, if we declare a scope of three for the robots, we can create allocations with up to three robots. However, if we define the scope of the robots to be *exactly* three, any structure (i.e., solution) found by the Alloy Analyzer must contain exactly 3 robots. We explicitly declare scopes for the: (a) capabilities, (b) atomic tasks, (c) robot signatures, as well as (d) the set of integers.

• To compute the scope of integers (***iScope***), we must know the range of integer numbers that should be included. To this end, we retrieve the largest integer int_{max} that appears in the model. Integer numbers $\boldsymbol{v} \in \mathbb{N}$ are used to describe the atomic task instances coordinates x and y, e.g.,

one sig at2_1 **extends** at2{} {x=$\boldsymbol{v}$ y=$\boldsymbol{v}$}

and in the *spaceXY* mission constraints[5], e.g.,

fact{**all** r:r6 | **all** c:r.hascapability | **all** do:c.do | do.y > $\boldsymbol{v}$}

As explained in Section 2.2, the scope of the integers set is defined by the bidwidth of int_{max}. Let int_{max} be the maximum integer among (a) the set of x's and y's coordinates

[5]Notice that the integers in arity constraints, for example, in the ***maxTasks*** mission constraint, are not considered in the scope as this is a restriction in the number of relations rather than a relation mapping to the set of integers or an inequality requiring the comparison of two integers.

of all atomic tasks, and (b) the integers in the **spaceXY** mission constraints, if any. We calculate the scope ***iScope*** of the integer set by,

$$\boldsymbol{iScope} = \lceil log_2(int_{max} + 1) \ + 1\rceil \tag{5.1}$$

• The capabilities' scope (***cScope***) is the sum of the number of robot capabilities, calculated over all the robots from the problem specification. This is the maximum number of capabilities that can appear in the model, however, capabilities may not appear in an allocation solution synthesised by Alloy if: (a) the robot that possesses this capability does not appear in the model, or (b) the robot does not use the capability to perform any task.

• The atomic tasks' scope (***aScope***) is **exactly** the total number of atomic tasks as instantiated by the pre-allocation stage.

• The robots' scope (***rScope***) is the total number of robots available.

Run predicate. To generate allocations of tasks to robots, we declare a **pred**icate statement and a **run** command. The Alloy Analyzer runs this command to generate models satisfying our "task allocation problem specifications".

We define the predicate and run commands as follows. The predicate (containing a list of statements that must hold) *TaskAllocation* does not contain any instructions as we have already defined all the constraints within the other parts of the Alloy model.

```
pred  TaskAllocation{ }
run TaskAllocation for iScope Int, cScope Capability,
          exactly aScope AtomicTask, rScope Robot
```

Example 7. For an allocation problem with ***iScope***=15, a total number of capabilities of the robots available ***cScope***=10, and ***aScope***=20 atomic tasks to be completed by ***rScope***=5 available robots, we add to the Alloy model,

```
pred  TaskAllocation{ }
run TaskAllocation for 5 Int, 10 Capability, exactly 20 AtomicTask, 5 Robot
```

The *run* command initialises the generation of allocations setting the maximum scope for the integers, capabilities and robots; and the exact number of atomic tasks. When running the Alloy Analyzer programmatically, we can also specify the maximum number of allocations to retrieve.

5.3.2 Allocation Visualisation

The Alloy Analyzer generates multiple task-to-robot allocations, if such allocations exist (for example, if there exists tasks defined in an area where robots are constrained not to

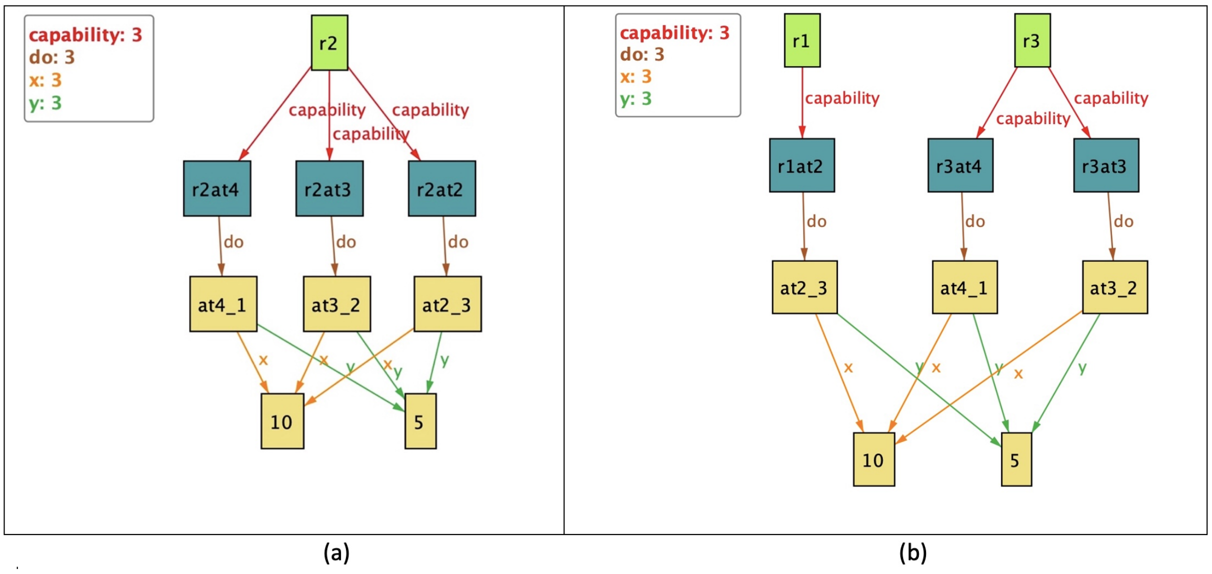
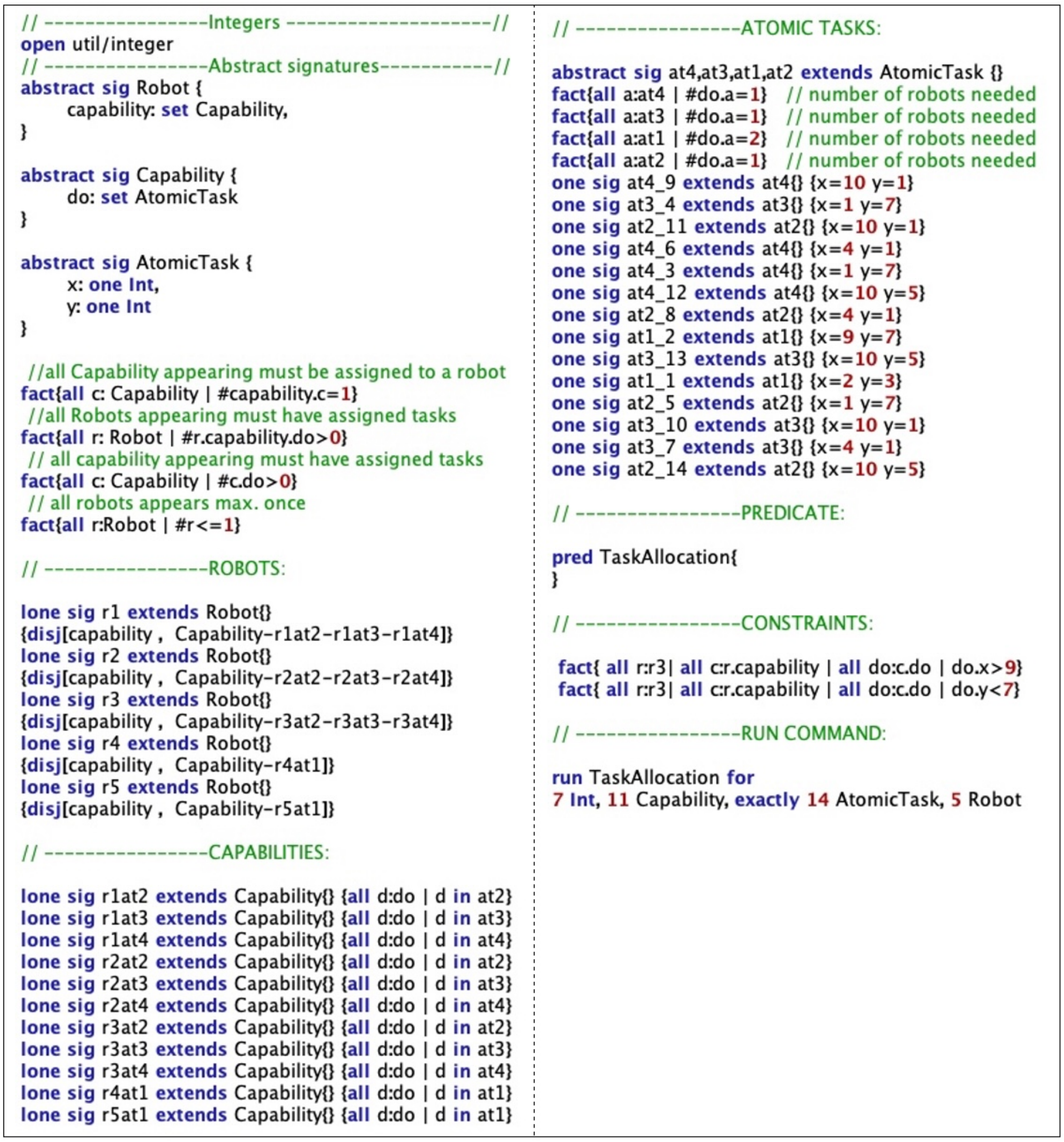


Figure 5.3: Visualisation of two different allocations of atomic task instances $at4\_1$, $at3\_2$ and $at2\_3$ in the Alloy Analyzer tool. (a) Robot $r2$ is assigned to do all tasks. (b) Robot $r1$ is assigned to task $at2\_3$ while robot $r3$ has to complete tasks $at4\_1$ and $at3\_2$.

enter, or when there is a task requiring more robots to be completed that the ones defined with the capability to perform the task, etc.). An advantageous aspect, particularly for explainability purposes, is its capability to visualize solutions using the Alloy Analyzer tool. We explain the Visualizer with the following example.

Example 8. Figure 5.3 shows an example of two allocations generated by Alloy Analyzer for a mission comprising three atomic tasks ($at2, at3, at4$) at *room1* with coordinates (10,5). Three robots are available, $r1, r2$ and $r3$; each robot has the capability of completing any of these tasks. The first allocation in Figure 5.3a shows the robot $r2$ at the top, using three of its capabilities $r2at4, r2at3$ and $r2at2$, to perform the three instantiated atomic tasks ($at4\_1, at3\_2$ and $at2\_3$) required to complete the mission. At the bottom, the location coordinates show that all tasks are to be performed at (10,5). Relations *hascapability*, *do*, x and y are also shown within the diagram.

A second allocation shown in Figure 5.3b shows two robots, $r1$ and $r3$. Robot $r1$ uses capability $r1at2$ to perform instantiated atomic task $at2\_3$. Meanwhile, robot $r3$ uses capabilities $r3at4$ and $r3at3$ to perform instantiated atomic tasks $at4\_1$ and $at3\_2$, respectively.



```
// ----------------Integers --------------------//
open util/integer
// ----------------Abstract signatures-----------//
abstract sig Robot {
      capability: set Capability,
}

abstract sig Capability {
      do: set AtomicTask
}

abstract sig AtomicTask {
      x: one Int,
      y: one Int
}

 //all Capability appearing must be assigned to a robot
fact{all c: Capability | #capability.c=1}
 //all Robots appearing must have assigned tasks
fact{all r: Robot | #r.capability.do>0}
 // all capability appearing must have assigned tasks
fact{all c: Capability | #c.do>0}
 // all robots appears max. once
fact{all r:Robot | #r<=1}

// ----------------ROBOTS:

lone sig r1 extends Robot{}
{disj[capability , Capability-r1at2-r1at3-r1at4]}
lone sig r2 extends Robot{}
{disj[capability , Capability-r2at2-r2at3-r2at4]}
lone sig r3 extends Robot{}
{disj[capability , Capability-r3at2-r3at3-r3at4]}
lone sig r4 extends Robot{}
{disj[capability , Capability-r4at1]}
lone sig r5 extends Robot{}
{disj[capability , Capability-r5at1]}

// ----------------CAPABILITIES:

lone sig r1at2 extends Capability{} {all d:do | d in at2}
lone sig r1at3 extends Capability{} {all d:do | d in at3}
lone sig r1at4 extends Capability{} {all d:do | d in at4}
lone sig r2at2 extends Capability{} {all d:do | d in at2}
lone sig r2at3 extends Capability{} {all d:do | d in at3}
lone sig r2at4 extends Capability{} {all d:do | d in at4}
lone sig r3at2 extends Capability{} {all d:do | d in at2}
lone sig r3at3 extends Capability{} {all d:do | d in at3}
lone sig r3at4 extends Capability{} {all d:do | d in at4}
lone sig r4at1 extends Capability{} {all d:do | d in at1}
lone sig r5at1 extends Capability{} {all d:do | d in at1}
```

```
// ----------------ATOMIC TASKS:

abstract sig at4,at3,at1,at2 extends AtomicTask {}
fact{all a:at4 | #do.a=1}   // number of robots needed
fact{all a:at3 | #do.a=1}   // number of robots needed
fact{all a:at1 | #do.a=2}   // number of robots needed
fact{all a:at2 | #do.a=1}   // number of robots needed
one sig at4_9 extends at4{} {x=10 y=1}
one sig at3_4 extends at3{} {x=1 y=7}
one sig at2_11 extends at2{} {x=10 y=1}
one sig at4_6 extends at4{} {x=4 y=1}
one sig at4_3 extends at4{} {x=1 y=7}
one sig at4_12 extends at4{} {x=10 y=5}
one sig at2_8 extends at2{} {x=4 y=1}
one sig at1_2 extends at1{} {x=9 y=7}
one sig at3_13 extends at3{} {x=10 y=5}
one sig at1_1 extends at1{} {x=2 y=3}
one sig at2_5 extends at2{} {x=1 y=7}
one sig at3_10 extends at3{} {x=10 y=1}
one sig at3_7 extends at3{} {x=4 y=1}
one sig at2_14 extends at2{} {x=10 y=5}

// ----------------PREDICATE:

pred TaskAllocation{
}

// ----------------CONSTRAINTS:

 fact{ all r:r3| all c:r.capability | all do:c.do | do.x>9}
 fact{ all r:r3| all c:r.capability | all do:c.do | do.y<7}

// ----------------RUN COMMAND:

run TaskAllocation for
7 Int, 11 Capability, exactly 14 AtomicTask, 5 Robot
```

Figure 5.4: Alloy specification for the hospital case study.

5.4 Hospital Case Study

We show the applicability of KANOA's task allocator using the hospital case study from Section 4.3. To this end, we generated an Alloy model based on the hospital problem specification[6] and used the Alloy Analyzer [55] to compute a set of possible solutions to the task allocation problem that complies with the set of requirements.

5.4.1 Allocation Model

The task allocation problem described in the Alloy language for the hospital case study is shown in Figure 5.4. This is generated automatically within the KANOA tool. In the left column, "open util/integer" import the set of integers. The abstract signatures for the types of Robot, Capability and Atomic Task are then defined, as well as four constraints (facts) that apply to the number of capabilities and robots present in the allocations, along with their relations.

The robots $r1$ to $r5$ available in the hospital and their capabilities are then defined. Subsequently, the capabilities associated with the robots and the relations *do*, mapping to the type of atomic task that each capability corresponds to, are added. For example,

```
lone sig r1at2 extends Capability{}{ all d:do | d in at2}
```

means that the robot capability $r1at2$ of robot $r1$ can be used to perform tasks of type $at2$. Then, the abstract signatures of each of the atomic tasks are added,

```
abstract sig at4,at3,at1,at2 extends AtomicTask{}
```

followed by the facts constraining the number of robots needed to complete each of these tasks; for example, two robots are needed to complete atomic tasks of type $at1$ is written as,

```
fact {all at1 | #do.a = 2 }
```

Next, the specific instances of the atomic tasks (for the pre-allocation stage) are defined; for example, an atomic task instance $at4\_9$ of type $at4$ required at a location with coordinates (10,1) is specified as,

```
one sig at4_9 extends at4{} { x=10 y=1 }
```

It then defines the empty predicate, *TaskAllocation*.

In the hospital case study, there are two constraints that the task allocator must consider: the restriction of robot $r3$ to move in an x coordinate greater than 9 units,

```
fact {all r: r3 | all c:r.hascapability | all do:x.do | do.x>9 }
```

and the restriction of robot $r3$ to move within a y coordinate less than 7 units,

```
fact {all r: r3 | all c:r.hascapability | all do:x.do | do.x<7 }
```

[6]This model was obtained using a tool that we implemented to automate the Alloy model generation process detailed earlier in this chapter, and that is described in Chapter 7.

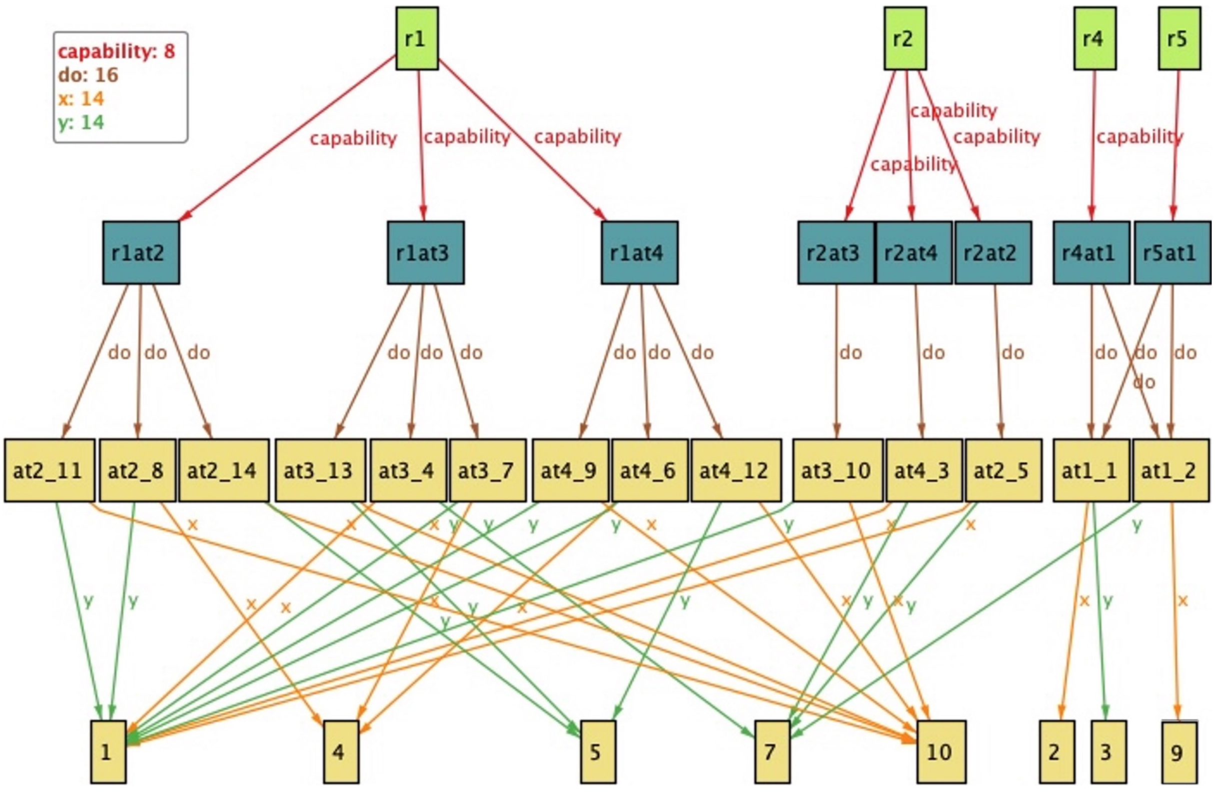


Figure 5.5: Example 1 of an allocation found by the Alloy Analyzer for the hospital case study. Robots $r1, r2, r4$ and $r5$ are deployed.

These two **fact**s are added into the model. Finally, it defines the *run* command and the different scope for the sets of **Int**egers, capabilities, atomic tasks and robots.

Figure 5.5 shows a solution found for the hospital case study where robots $r1, r2, r4$ and $r5$ are deployed. In this case, robot $r1$ has eight tasks to perform, while robot $r2$ has four. As expected, robots $r4$ and $r5$ are assigned all tasks of type $at1$ as these robots are the only ones equipped with the capability to accomplish this task. Robot $r3$ is not deployed in this solution.

Figure 5.6 shows a second solution where robots $r1$ and $r3$-$r5$ are deployed. Robot $r3$ has four allocated tasks ($at2\_11$, $at2\_14$, $at3\_13$ and $at4\_12$), required at two locations with coordinates (10,1) and (10,5). These comply with the mission requirement that robot $r3$ must only accept tasks with coordinates x>9 and y<7.

5.5 Discussion and Limitations

In the general case, if there are T tasks to be allocated into R robots, there are a total of R^T possible allocations. This is, it has exponential complexity with respect to the number of tasks, specifically $O(R^T)$. Nevertheless, the approach proposed in this chapter reduces this complexity significantly by incorporating a series of relevant constraints. As demonstrated with the hospital case study, in many practical instances of the task

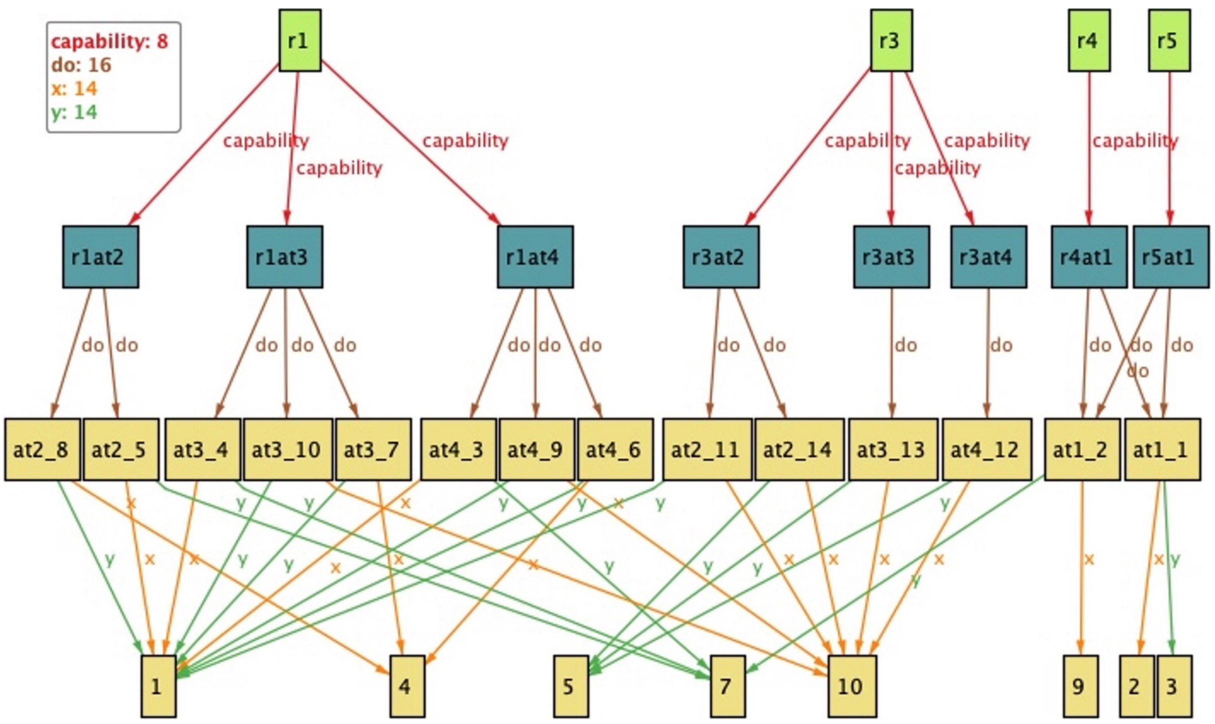


Figure 5.6: Example 2 of an allocation found by the Alloy Analyzer for the hospital case study where robot $r3$ is deployed.

allocation problem, constraints appear naturally with the heterogeneity of the robots, constrained working space, etc. In this case study, there are 5 robots and 14 tasks; this is $5^{14} = 6,103,515,625$ possibilities. However,

- there are only two robots ($r4$ and $r5$) that possess the capability to move objects. Given that moving objects requires both robots, all resulting task allocations must show these tasks alloated to $r4$ *and* $r5$. This reduces the possible allocations of move object tasks to one.
- There is also the spatial constraint over robot $r3$. This only allows $r3$ to accept tasks in an area covering six tasks (in rooms four and five). As $r1 - 3$ have the capability to complete these tasks, there are 3^6 possible task allocations.
- For the rest of the six tasks (rooms two and three), only $r3$ is not allowed to work. Hence, there are 2^6 possible task allocations.

This reduces the task allocation solution space to $(1 \times 2^6 \times 3^6) = 46,656$ feasible task allocations.

One of the advantages of using a constraint solver is that any solution returned is correct by construction. However, a disadvantage is that we cannot guarantee the optimality of the solutions with respect to, for instance, the optimisation objectives defined as part of the KANOA problem specification (minimising the robots' idle time, maximising the probability of mission success, and minimising the cost of travelling between locations). Using a SAT approach signifies that solutions are not distinguished in the sense that there is no ranking of the solutions regardless of some solutions being *better* than others. To partly compensate for this weakness, we rely on the user to specify any mission constraints into the KANOA DSL to guide the search process. For example, if it is known that there is a series of tasks to be done in a location that is easier to reach by one of the robots, this knowledge can be added as a mission constraint. Another example of such a constraint is dividing the robots to accept tasks only in specific areas of the workspace by adding spatial constraints.

We separated the allocation and scheduling task problems into two distinct stages. This decision allows for the incorporation of a diverse range of constraints while avoiding the complexity associated with the modelling of these as a single model. However, the evaluation of the quality of an allocation requires the generation of schedules from this allocation and the computation of performance metrics, such as the time required to complete all tasks and the total idling time when robots are inactive. Consequently, the detailed evaluation of the task allocator is conducted in conjunction with the task scheduler in Chapter 7. A limitation lies in the fact that a feasible allocation does not guarantee a feasible schedule. It is plausible that no permutation of tasks from a given allocation may align with the mission requirements, such as completing all tasks within a specified time frame.

Another limitation of the pre-allocation of tasks to robots (defined as part of the mission constraints) is that it can only pre-allocate one robot to each atomic task. This is a problem when a task requires more than one robot to be completed. For example, when defining a mission constraint "**`allocate atomic task`** at1 **`to robot`** r4", where task at1 requires two robots, it is not possible to pre-allocate this task to a second robot. This feature will be explored in future work.

Finally, the use of integers is discouraged in Alloy as there is usually another abstract description for the set of integers tailored for the application in hand [162]. In our case, we make use of this set to describe the locations where tasks are to be accomplished and define the scope of the integers based on the range of numbers appearing in the x and y coordinates appearing in these locations. This creates a set with unnecessary integer values that are not needed in the model and increases considerably the time to compute solutions to the allocation problem as the integer scope increases. For the hospital case study, the Alloy Analyzer ran out of memory for an integer scope of 12, increasing considerably the time to generate solutions as the integer scope grows beyond 6[7][8]. A possible solution for future work would be to create an ordered set containing only the integers used to describe the coordinates of the locations. Another solution involves fully abstracting away from

[7]A scope of 6 generates integers from -31 to 31 as explained in Section 2.2.

[8]Experiments ran on a MacBookAir, Apple M1, 8GB, macOS Monterey

coordinates by introducing a "zone" or "area" signature. The robot-to-task area mapping relations can then be automatically generated from Kanoa specifications, relieving Alloy from dealing with integers. This approach also offers increased flexibility in terms of the shape of workspace areas, eliminating the necessity for them to be rectangular, as is the case in the current version of the approach.

5.6 Related Work

In Section 3.3 as part of the answer to exploratory question EQ1, we proposed a general definition for the *task allocation* problem and broadly defined the different common techniques used for the allocation of tasks. These include algorithms that are part of the well-known MRTA (multi-robot task allocation) set of solutions [9], such as the integer linear programming [163], market/auction-based approaches [164, 165], and the multi-Travelling Salesman problem (mTSP) [166]. Modelling techniques including the optimal assignment problem (OAP), the fair division problem and the ALLIANCE efficiency problem are also covered by [9]. These algorithms provide a wide range of descriptions for the problem at hand. For example, OAP may describe a simple allocation problem where multiple tasks have assigned weights depending on the robot that performs them, then the objective is to minimize the total cost. The problem can be solved using, for instance, the Hungarian algorithm [167, 168].

Compared to *travelling* problems, *assignment* problems use binary variables to describe if a robot has to complete the task or not. In contrast, constraint solvers, such as Alloy, search for solutions that comply with all the constraints of the design space. It uses relational and set theory to reason about possible solutions. Hence, compared to Hungarian binary algorithms or Linear Programming, constraint solvers allow a richer description of the allocation problem (agents, capabilities, tasks, locations, constraints, etc.) and their relations, while still finding solutions that are not easily synthesized. Although optimality is lost in the process, we gain expressiveness of the requirements (stated in a declarative manner which makes it easier to expand, if necessary) and add complexity to the mission constraints that we can reason about. Optimising the allocation solutions is done in conjunction with the scheduling and plan synthesis problem in Chapter 6.

Internally, the Alloy Analyzer transforms the model into a series of boolean satisfiability formulae to leverage the power of SAT solvers [58]. Another approach involves the use of satisfiability module theories (SMT). SMT "extends boolean satisfiability with rules (theories) for domains such as linear arithmetic" [169]. In [169], an SMT was used in a (single) robotic arm scenario for the task motion planning problem. The crucial reason they claim is that some SMT solvers enable *incremental solving to add/remove constraints*, although this is also the case of SAT solvers, where new constraints can be added incrementally (as shown using Alloy). In further work, we will assess the use of

SMT solvers (if the newly tackled task allocation problems require to add such types of *rules* handled by SMT), such as Z3 [170] and compare it with our proposed approach.

5.7 Summary

The Alloy Analyzer is a tool used for the validation of declarative models specified in the Alloy language. It converts the model into a SAT formula, takes the set of constraints of the model and finds *structures* that satisfy them. The Alloy Analyzer can be used for the verification of properties and generation of counterexamples, as well as for the exploration of models and generation of sample structures.

In this chapter, we used the Alloy language and Alloy Analyzer for the description and generation of the allocation of tasks to robots, respectively. We described the building blocks of the Alloy model, showing the Z notation schemas from which information is obtained from the KANOA DSL. We illustrated the applicability of our approach using a hospital case study, for which we showed some of the allocations synthesised by the Alloy Analyzer.

Chapter 6

Task scheduling

This chapter presents the KANOA approach for addressing the task scheduling problem in multi-robot systems (MRS). This approach involves synthesising individual plans for robots to execute a series of tasks while adhering to both functional and non-functional requirements outlined in the problem specification encoded in the KANOA DSL (refer to Section 4.2). These requirements may encompass achieving a mission success probability exceeding 0.9, initiating task $t1$ immediately after $t2$, or concluding task $t3$ within a specified time frame. These examples highlight a significant challenge in MRS task scheduling—the interdependence of tasks. In the context of KANOA, task constraints are divided into two within the system specification: task constraints and mission constraints. Task constraints are defined as part of the *task model* when an atomic task is declared to require more than one robot to be completed, or when compound tasks are defined as ordered (where subtasks follow a specific order) or consecutive (where each subtask starts upon completion of the preceding one). Mission constraints are defined in the mission specification, encompassing constraints such as completion times, spatial constraints, and initial allocations of tasks to robots.

A second challenge in the scheduling of robot tasks arises from the size of the solution space. In a scenario where a robot is allocated n_t tasks (with no task interdependencies), there are $n_t!$ possible solutions (e.g., there are 3,628,800 possible ways of scheduling 10 tasks); and for n_r robots, the solution space grows to $n_r \times n_t!$. Consequently, the use of heuristics [47–49], meta-heuristics [88] and hybrid techniques [43, 92, 93, 102] has been adopted in the majority of recent solutions as shown previously in Table 3.4. The comprehensive review in Chapter 3 offers further insights into the application of these methodologies for addressing the task scheduling problem in robotic systems.

A third challenge arises from the variety of inherent uncertainty in MRS. Table 3.5 shows some common sources of uncertainty that researchers typically consider when modelling robotic systems. These include robots exploring partially known or unknown environments [22–25], the roughness of the terrain affecting mobile robots performance, the changes in the communication topology when adjustments have to be made to improve communication and failures [110, 112, 126]. In this chapter, we introduce KANOA's task scheduler. Our approach considers two types of uncertainties: the possibility of a robot failing to complete a task, and the possibility of a robot getting stuck when travelling from one location to another.[1] KANOA combines in a new way several techniques, including

[1]These types of uncertainties are studied in robotic systems as shown in Table 3.5. Other types of

probabilistic model checking and genetic algorithms, to address the task scheduling problem. We use the PRISM probabilistic model checker to reason about system uncertainties and a variant of the EvoChecker framework[2] to explore the solution space and generate Pareto-optimal solutions.

This chapter is organised as follows. Section 6.1 describes the task scheduling problem. Section 6.2 introduces the pre-scheduling stage for the grouping of robots and the transfer of the compound task constraints to their constituent atomic tasks. Section 6.3 presents the methodology to solve the task scheduling problem in KANOA, and Section 6.4 covers the optimisation of potential robot plan solutions. Section 6.5 demonstrates the applicability of KANOA through the hospital case study. Section 6.6 discusses the advantages and limitations of the KANOA scheduler. Finally, Sections 6.7 and 6.8 review related work and summarise the chapter, respectively.

6.1 Task Scheduling Problem

In this section, we describe the task scheduling problem. Informally, we define this problem as finding a sequence of possible *robot actions* (execute a task, travel, or stay idle) such that, for each robot, the execution of these actions ensures that all of its allocated tasks are completed with an optimal level (minimising the completion time, travelling cost and/or probability of success), satisfying all relevant constraints from the problem specification across the task schedules of all robots.

In previous chapters, we introduced the KANOA problem specification for multi-robot systems (Chapter 4) and a solution to the allocation of tasks to robots (Chapter 5). The task scheduling problem assumes the existence of: (i) a description of the system; and (ii) one or more task allocations generated by the KANOA allocator. This workflow is illustrated in Figure 6.1. The problem specification (1) describes the different aspects of the MRS mission, divided into the world, robot and task models, as well as the mission specification. The last describes the constraints and the list of tasks that robots must complete. This information is parsed, stored and used to generate feasible allocations in the pre-allocation and task-allocation stage (2). This chapter deals with stages (3) and (4). The result of the task scheduling process (4) is a set of (Pareto-)optimal robot plans.

Before formulating the scheduling problem, we review some of the previously defined concepts needed to describe the task scheduling problem.

uncertainties can be modelled in future work.

[2]In comparison to EvoChecker, the main difference is that we incorporated multiple models as part of the evaluation stage in the multi-objective evolutionary algorithm.

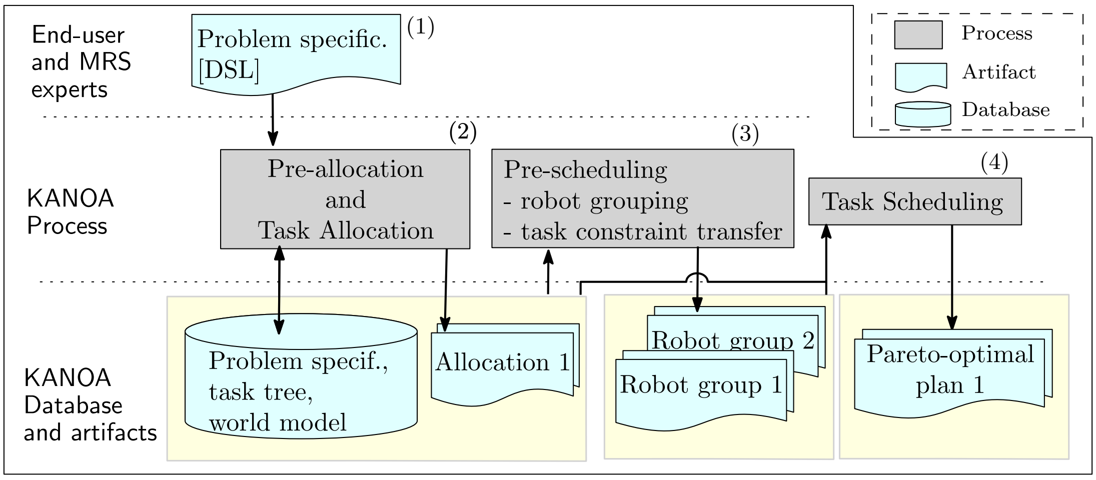


Figure 6.1: Overview of the KANOA workflow from the specification of the problem to the generation of robot plans .

Preliminaries

We define the multi-robot problem specification divided into:[3] the *world model*, the *tasks model*, the *robots model* and the *mission*, and formalised it in Z notation in Section 4.2.1. We refer to this as KANOA's problem specification. KANOA allows the specification of multiple objective and constraint requirements relevant to multi-robot systems. The complete list of these requirements is depicted in Table 6.1. These consist of four types of constraints (C1-C4) to consider for the allocation of tasks, and six constraints (C5-C10) applicable to the task scheduling. There are also three optimisation objectives (O1-O3) related to the probability of mission success, the robot's idling time and the total robots' travelling cost. Users of KANOA can specify one or more of these objectives to guide the synthesis of robot plans.

There are three types of tasks in KANOA[4]. First, atomic tasks $at \in AtomicTask$ are indivisible activities carried out by one or several robots, $at.nRobots \in \mathbb{N}$, at a given location $at.locationID$[5]. These have identifiers $at.id \in ATaskID$. Secondly, compound tasks $ct \in CompoundTask$ are formed by a sequence of (atomic or compound) subtasks, $ct.subtasks$; and are identified $ct.id \in CTaskID$. Compound tasks can be performed in no specific order, in a specific order (without forcing them to be consecutive), or consecutively and ordered (start the next exactly when the last subtask finishes); where $ct.constraint \in \{none, ordered, consecutive\}$ is related to constraints C5-C7, respectively, in Table 6.1.

A mission task $mt \in MissionTask$ instantiates an atomic or compound task to be carried out by robots. It has a unique identifier $mt.taskID$ that corresponds to the

[3]This is a summary of Section 4.2.1. We refer the reader to this section for further details.
[4]See KANOA problem specification in Section 4.2.1
[5]Where *locationID* is the location identifier.

Table 6.1: List of requirements (objectives and constraints) that can be specified in KANOA's problem specification and the stages (see Figure 6.1) where these are applicable.

ID	Name	Description	Addressed at stage	Type
C1	spaceXY	Limits the physical operational area of robots.	2	Constraint
C2	allocateT	Allocates a task to a specific robot.	2	Constraint
C3	closest	Allocates a task to the closest robot.	2	Constraint
C4	maxTasks	Bounds the maximum number of tasks accepted by a robot.	2	Constraint
C5	joint task	Synchronise two or more robots to complete an atomic task.	4	Constraint
C6	ordered task	Constrains a subtask (of a compound task) to start only when the last one is completed at some moment in the past.	4	Constraint
C7	consecutive task	Constrains a subtask (of a compound task) to start immediately after the last one is completed	4	Constraint
C8	taskTime	Constraints the time to start or finalise a task.	4	Constraint
C9	rateSucc	Sets minimum rate of mission success without failure.	4	Constraint
C10	time	Sets the time available to complete the mission.	4	Constraint
O1	maximiseSuccess	Maximise the probability of success without failure.	4	Objective
O2	minimiseIdle	Minimise the robots' idling time	4	Objective
O3	minimiseTravel	Minimise the travelling cost.	4	Objective

taskID of the relevant atomic or compound task. Mission tasks are instantiated into *AtomicTaskInst* and *CompoundTaskInst*.[6] A mission is defined as the set of mission tasks $m.tasks : \mathbb{P} MissionTask$, where $m : Mission$; the set of objectives $m.objectives \subseteq \{minimiseIdle, minimiseTravel, maximise\text{-}Success\}$; and constraints $m.constraints$ of types C1-C4 and C8-C9 defined in Table 6.1; and time-bound to complete the mission $m.time \in \mathbb{N}$ parameter, referring to constraint C10.

Section 5.1 defines an allocation *alloc* as a partial function between robots and a set of atomic task instances identifiers *ATInstanceID*, $alloc : Robot \nrightarrow \mathbb{P}\, ATInstanceID$. The available robots are defined in the problem specification. For example, for a problem specification $p : ProblemSpecification$, the set of available robots is given by $p.robotsModel$.

Now that we have defined the preliminaries, we proceed with the definition of the task scheduling problem.

[6]The tasks that robots must carry out are the atomic task instances. As two different mission tasks can refer to the same atomic or compound task, we instantiate these into *AtomicTaskInst* and *CompoundTaskInst*, respectively, as described in Section 5.1.

Task scheduling problem

A task schedule requires knowing the order in which tasks are executed. Hence, we first define a task permutation and a task permutation vector as follows.

Definition 6.1.1 (Task Permutation) *Given a non-empty set of atomic task IDs* $AT = \{at_1, at_2, \ldots, at_n\}$ *and a* permutation index $j \in \{1, 2, \ldots, n!\}$, *the* j*-th permutation of the* n *tasks is a sequence* $\langle at_{b_1}, at_{b_2}, \ldots, at_{b_n}\rangle$ *where* $\langle b_1, b_2, \ldots, b_n\rangle$ *represents the* j*-th permutation of the task ID indices 1, 2, ..., n according to the Lehmer code [171]. We use the notation* $perm(AT, j)$ *to denote this task permutation.*

Definition 6.1.2 (Task Permutation vector) *Given an allocation* a_i *with the set of robots* $R = \{r_1, r_2, ..., r_m\} = \mathrm{dom}\, a_i$*, and, for each of these robots* r*, a permutation index* j_r *corresponding to the order in which it will execute its tasks, we refer to the sequence* $\bar{j} = \langle j_{r1}, j_{r2}, ..., j_{r_m}\rangle$ *as the task permutation vector.*

We illustrate these definitions with the following example.

Example 9. For an allocation a deploying two robots $R = \langle r1, r2\rangle$ with assigned tasks $a(r1) = \{at1, at2\}$, $a(r2) = \{at3, at4\}$, the possible **task permutations** for each robot are,

$r1$ tasks permutations: $\langle at1, at2\rangle$ and $\langle at2, at1\rangle$

$r2$ tasks permutations: $\langle at3, at4\rangle$ and $\langle at4, at3\rangle$

and the possible **task permutation vectors**,

1) $\bar{j_1} = \langle 1, 1\rangle$. Robot r1 do $[at1, at2]$ and robot r2 do $[at3, at4]$
2) $\bar{j_2} = \langle 1, 2\rangle$. Robot r1 do $[at1, at2]$ and robot r2 do $[at4, at3]$
3) $\bar{j_3} = \langle 2, 1\rangle$. Robot r1 do $[at2, at1]$ and robot r2 do $[at3, at4]$
4) $\bar{j_4} = \langle 2, 2\rangle$. Robot r1 do $[at2, at1]$ and robot r2 do $[at4, at3]$

We can access a specific task permutation, such as $perm(a(r1), 1) = \langle at1, at2\rangle$, to retrieve the first task permutation of $r1$.

In addition to tasks, there are two more actions, idle and travel, that robots can perform and therefore interleave with task executions (if necessary). Accordingly, we define a robot plan as follows.

Definition 6.1.3 (Robot Plan) *A robot plan for a robot* r *that is assigned tasks* $a(r)$ *by an allocation* a *and is required to perform these tasks in the order given by the task permutation* j *is a sequence comprising a combination of:*

- $(travel, d)$ *pairs, where* $travel \in LocationID$ *is the identifier of a location that the robot needs to travel to, and* $d \in N$ *is the mean number of time units required to reach this location from the current position of the robot;*

- *$(idle, d)$ pairs, where idle specifies that the robot must idle for $d \in \mathbb{N}$ time units;*
- *$(dotask, d)$ pairs, where $dotask \in ATInstanceID$ is the identifier of an atomic task, and $d \in N$ is the duration needed to complete the relevant task by robot r,*

such that each task from the allocation $a(r)$ appears in the plan precisely once, and in the order specified by the task permutation j.

We use the notation $plan(perm(a(r), j))$ to denote such a plan, and term it "a plan synthesised for the jth permutation of the tasks allocated to r in the allocation a."

Example 10. If the first task permutation of a task allocation indicates that robot $r1$ must "perform task $at1$ (in location $l1$), then perform task $at2$ (in location $l2$)", a possible robot plan is: "travel to location $l1$, which takes 4-time units from the robot's initial position, and do $at1$ which takes 5-time units; idle one-time unit; travel to location $l2$ from $l1$ which takes 4-time units, and then do $at2$ taking 7-time units". This is represented as follows:

$$perm(a_i(r1), 1) = \langle at1, at2 \rangle,$$

and the possible robot plan for robot $r1$ is

$$plan(\langle at1, at2 \rangle) = \langle (l1, 4), (at1, 5), (idle, 1), (l2, 4), (at2, 7) \rangle$$

Notice that the initial position is not explicit in the plan but obtained from the problem specification. Also, the initial travelling point for *travel* actions is implicit from the location in the previous action.

We now define the set of plans consisting of one (and only one) plan for each robot in an allocation such that each robot with allocated tasks knows what to do at every time until the mission is completed.

Definition 6.1.4 (Task Schedule) *Given an allocation "a" for a set of robots $\{r_1, r_2, \ldots, r_m\} = \operatorname{dom} a$, a task schedule is a set of plans $\{plan_1, plan_2, \ldots, plan_m\}$ s.t. $plan_i$ is the valid robot plan for robot r_i, for $i = 1, 2, \ldots, m$. The completion of all these plans complies with the constraints of task ordering, consecutive, joint tasks, completion time and rate of success (as summarised in Table 6.1).*

Finally, we define the task scheduling problem as follows,

Definition 6.1.5 (Task Scheduling Problem) *Given a set of KANOA-generated task allocations for an MRS mission, the task scheduling problem consists of synthesising a set of task schedules. These schedules must be Pareto optimal with respect to one or more optimisation objectives allowed by the KANOA problem specification (maximise the probability of mission success, minimise the total idling time and/or minimise the total travelling cost).*

Example 11. For two allocations a_1, a_2, where robots $r1$ and $r2$ are deployed in allocation a_1, and $r1, r2$ and $r3$ in a_2, solving the task scheduling problem requires first finding task schedules consisting of robot plans for all robots in either allocation a_1 or a_2, where the completion of these plans ensures the successful completion of the mission. Synthesised task schedules for our running example follow one of these two patterns:

$$ts1 = \{plan_{a_1,r1}, plan_{a_1,r2}\}$$
$$ts2 = \{plan_{a_2,r1}, plan_{a_2,r2}, plan_{a_2,r3}\}$$

where (to simplify notation) $plan_{a_1,r1}$ is a possible plan for robot $r1$ with allocated tasks from allocation a_1. Solving the problem secondly requires retaining from these task schedules those that are Pareto optimal with respect to the optimisation objectives defined in the KANOA problem specification.

6.2 Pre-scheduling

Previously, we described the KANOA DSL to specify the multi-robot problem specification in Section 4.2.1 and solved the allocation of tasks to robots in Section 5.1. The outcome of the task allocation is a set of task allocations as shown in Figure 5.1. For each allocation found by KANOA, we reduce (where possible) the complexity of the scheduling, by identifying *groups* of robots that share *task dependencies*, and do not have such dependencies with any robot from outside their group. This process is done in a *pre-scheduling* stage, which we describe in the following section.

The goal of the task scheduler is to create robot plans that include all atomic tasks necessary to complete the mission. These plans must comply with any task constraint specified as part of the KANOA's task model. Hence, in the pre-scheduling stage, we also transfer existing compound task constraints into the atomic tasks (see Section 6.2.2).

6.2.1 Robot Grouping

We group robots with shared constrained tasks to simplify the models we will use for the scheduling process. A **constrained task** is a task that needs to be executed by more than one robot, or that needs to be executed (or, in the case of a compound task, to have its subtasks executed) in a certain order because of constraints specified in the MRS mission specification.

For the purpose of robot grouping, we represent an MRS mission as a tree structure as illustrated in Figure 6.2. In this representation, each mission task appears as a child node of the root, and has as descendants the compound and/or atomic tasks it maps to, with compound tasks as parent nodes for their subtasks. A **parent** relation on this tree structure is defined in the natural way. In particular, each subtask of a compound task

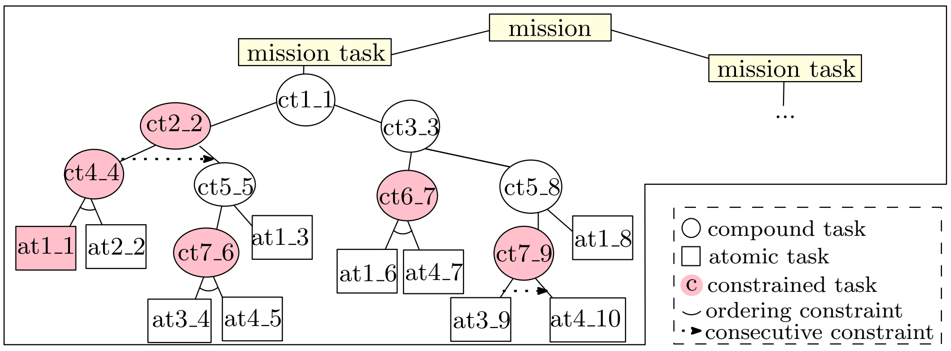


Figure 6.2: Tree of tasks conforming a mission. Nodes at level 1 represent mission tasks. Each "mission task" node has one child. Circle nodes indicate that all subtasks must be completed. Leaf nodes represent atomic tasks and do not possess subtasks, each of them has to be executed by one or more robots. We use arc links to show *consecutive* constraints and dotted arrows for *ordering* ones. Constraint tasks (depicted in pink) include tasks with any of these constraints and joint atomic tasks.

has that compound task as its parent.

We say that two tasks are **task dependant** if they share a constrained task.

Definition 6.2.1 ***(Task Dependency).*** *Two atomic tasks in a mission task tree are dependant iff they have a common ancestor compound task with an ordered or consecutive constraint over its subtasks.*

Example 12. Consider the left-side mission task depicted as a tree-structure in Figure 6.2. Constrained tasks are shaded (in pink), for example, $ct2\_2$ is a compound task with a *ordering* constraint on its subtasks, $ct4\_4$ and $ct5\_5$; this indicates that all tasks from $ct4\_4$ have to be completed before subtasks from $ct5\_5$. Hence, we say that all atomic tasks reachable from $ct2\_2$, $\{at1\_1, at2\_2, at1\_3, at3\_4, at4\_5\}$ are *dependant*, and independent from the rest of the atomic tasks as they share no constrained task with those. The rest of atomic tasks grouped by dependencies are: $\{at1\_6, at4\_7\}, \{at3\_9, at4\_10\}, \{at1\_8\}$.
Appendix B shows this example in KANOA DSL.

The **three-stage process** for robot grouping is depicted in Figure 6.3. The initial step (a) involves grouping atomic tasks based on task dependencies on higher nodes. **First**, KANOA computes the Breadth-First Search *BFS* algorithm pruning the mission tree (*missionTree*) whenever a constrained task is encountered or when an atomic task is reached. The algorithm then returns the task at which the tree was pruned. We define this set of tasks $tc_i \in TinstID$, where $TinstID \in ATInstanceID \bigcup CTInstanceID$ as,

$$BFS(missionTree) = \{tc_1, tc_2, ...\} \tag{6.1}$$

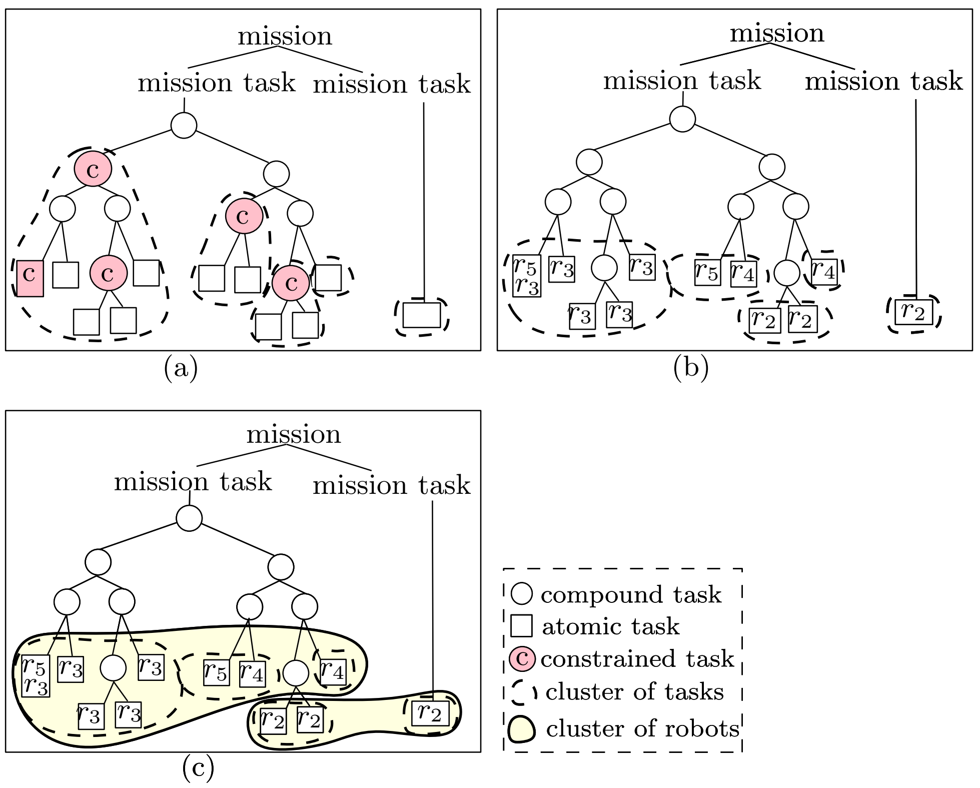


Figure 6.3: Grouping robots process. (a) Tasks are grouped if they are task-dependent. In the example, five subtrees were found. (b) For each subtree, the allocated robots to its atomic tasks are computed. (c) The transitive closure over the robots returns clusters of robots sharing constrained tasks directly or indirectly through other robots. In this example, there are two robot clusters $< r3, r4, r5 >$ and $< r2 >$.

Example 13. For the mission in Figure 6.2 and Figure 6.3a, $BFS(missionTree) = \{ct2\_2, ct6\_7, ct7\_9, at1\_8, ...\}$.

Second, given an allocation $a : alloc$, we obtain the robot(s) allocated to each task $t \in BFS(missionTree)$. We define this process as a function,

$$atRobots : alloc \times \mathbb{P}\ TinstID \nrightarrow \mathbb{P}(\mathbb{P}\ Robot) \tag{6.2}$$

and obtain $atRobots(a, BFS(missionTree))$ by acquiring for each $t \in BFS(missionTree)$:

1. if t is an atomic task, return the set of robots $R = \{r : Robot \mid t \in alloc(r)\}$ this task is allocated to.

2. if t is a compound task, first get its set of descendent atomic tasks. Perform 1 for each of these atomic tasks and combine their outcomes into a set R. Then, return this set.

For case 1 above, $\| R \| > 1$ only when t is a joint task, else $\| R \| = 1$. The result is illustrated in Figure 6.3b and the process explained in Algorithm 6. The function

Algorithm 5 Get groups of robots that share task dependencies.

```
Require: $missionTree$: tree structure build from all mission tasks in KANOA
Require: $A_i$: $i_{th}$ allocation of atomic tasks to robots
 1: procedure GetRobotsGroupedByTaskDependencies($missionTree$, $A_i$)
 2:     firstConstrainedTasks = $BFS(missionTree)$        ▷ bread-first search, see Section 6.2
 3:     robotListOfLists = [ ]                            ▷ initialise list for groups of robots
 4:     for t in firstConstrainedTasks do
 5:         if t is atomic task
 6:             robotListOfLists.add([list of robot(s) allocated to t in $A_i$])
 7:         else if t is compound task
 8:             set = {}                                  ▷ empty set
 9:             for at in reachable atomic tasks from t
10:                 set.addAll(robot(s) allocated to at in $A_i$)
11:             end for
12:         robotListOfLists.add([set.toList])
13:     return robotListOfLists
```

GetRobotsGroupedByTaskDependencies received a tree-structure of tasks representing a mission task *missionTree*, and an allocation A_i. It returns groups of atomic tasks that are only dependent on members of the same group robotListOfLists. The algorithm's correctness is ensured by employing breadth-first search (BFS) until it reaches either a constrained task or an atomic task. Its completeness is guaranteed, as it terminates after iterating over a finite number of tasks in the mission tree and a finite number of robots assigned in allocation A_i.

Example 14. For the mission in Figure 6.2 and an allocation a_1 with robots assigned to atomic tasks as shown in Figure 6.3b, $atRobots(a_1, BFS(missionTree)) = \{\{r3, r5\}, \{r4, r5\}, \{r2\}, \{r4\}, \}r2\}\}$.

Third, if a robot belongs to different sets in *atRobots*, these two groups are merged. We compute the transitive closure among all robots to combine sets if they share a robot (directly or through a transitive relation).

Definition 6.2.2 (Robot dependency relation) *The robot dependency relation is the reflexive binary relation $\boldsymbol{R}$ on the robots of an MRS such that two robots r_a and r_b are in this relation (i.e., $r_a \boldsymbol{R} r_b$) iff*

$$\exists\, R_j \in atRobots(alloc, BFS(missionTree)) \bullet \{r_a, r_b\} \subseteq R_j.$$

Definition 6.2.3 (Group of robots) *We define a robot group as a set of robots,*

$$group : \mathbb{P}\, Robot$$

Algorithm 6 Get groups of robots that share task dependencies.

```
Require: missionTree: tree structure build from all mission tasks in KANOA
Require: $A_i$: $i_{th}$ allocation of atomic tasks to robots
 1: procedure GetRobotsGroupedByTaskDependencies(missionTree, $A_i$)
 2:     firstConstrainedTasks = BFS(missionTree)        ▷ bread-first search, see Section 6.2
 3:     robotListOfLists = [ ]                          ▷ initialise list for groups of robots
 4:     for t in firstConstrainedTasks do
 5:         if t is atomic task
 6:             robotListOfLists.add([list of robot(s) allocated to t in $A_i$])
 7:         else if t is compound task
 8:             set = {}                                ▷ empty set
 9:             for at in reachable atomic tasks from t
10:                 set.addAll(robot(s) allocated to at in $A_i$)
11:             end for
12:         robotListOfLists.add([set.toList])
13:     return robotListOfLists
```

and all groups of robots in an allocation as,

$$groups : alloc \rightarrow \mathbb{P}\, group \tag{6.3}$$

such that

$$\forall\, g \in groups(a_i) \bullet \forall\, r_a, r_b \in g \bullet r_a \mathbf{R}' r_b,$$

where $\mathbf{R}'$ *is the transitive closure of* $\boldsymbol{R}$.

To represent the groups of robots for an allocation a_i and robots grouped by $atRobot(a_i, BFS(missionTree))$, we create the squared adjacency matrix where a value of 1 in the matrix elements is associated with two robots belonging to the same group. Then, we compute the transitive closure using Warshall's algorithm or variations [172]. Finally, we remove any duplicated row and retrieve the robots on each row left where there is a 1 as shown in the next example.

Observation 3. We adopted the term *group* avoiding the term *subteam* as in market-based approaches [164], as this refers to a subset of robots that coordinate to complete a task. This concept is implicit in our definition of *joint tasks*, where robots must coordinate to complete a task.

Example 15. The adjacency matrix of {{r3,r5},{r4,r5},{r2},{r4},}r2}} is given by,

$$M = \begin{array}{c|cccc} & r_2 & r_3 & r_4 & r_5 \\ \hline r_2 & 1 & 0 & 0 & 0 \\ r_3 & 0 & 1 & 0 & 1 \\ r_4 & 0 & 0 & 1 & 1 \\ r_5 & 0 & 1 & 1 & 1 \end{array}$$

and its transitive closure by,

$$M' = \begin{array}{c|cccc} & r_2 & r_3 & r_4 & r_5 \\ \hline r_2 & 1 & 0 & 0 & 0 \\ r_3 & 0 & 1 & 1 & 1 \\ r_4 & 0 & 1 & 1 & 1 \\ r_5 & 0 & 1 & 1 & 1 \end{array}$$

Rows are removed if they are repeated. Hence, two groups of robots are obtained. From row 1, $\{r2\}$, and from row 2, $\{r3, r4, r5\}$, as shown in Figure 6.3c.

6.2.2 Task Constraint Transfer

In KANOA's DSL, we specify task constraints as part of the compound tasks and atomic tasks definitions. Compound tasks can be defined with a constraint of type *order* or *consecutive*, or none. We must transfer these constraints to the constituent atomic tasks, which are the type of tasks to be allocated to robots and, ultimately, the ones appearing in the robot plans. We define two functions to capture these two compound task constraints passed into the atomic tasks.

First, we define the function *doneBefore* as,

$$doneBefore : ATInstanceID \nrightarrow \mathbb{P}\, ATInstanceID \tag{6.4}$$

mapping any atomic task to the set of atomic tasks that *have to be* completed before at_i can be *started*.

Second, for every atomic task at_i, we define the function,

$$justDone(at_i) = \begin{cases} at_{i-1} & \text{if } \exists\ at_{i-1} \text{ that has to be } \textit{just} \text{ completed to start } at_i \\ none & \text{otherwise} \end{cases} \tag{6.5}$$

which returns an atomic task at_{i-1}, iff there is a **consecutive** compound task constraining at_i to be started immediately after at_{i-1}.

KANOA's task model allows to specify compound tasks with compound tasks as subtasks. Hence, since compound tasks as tree-like structures, atomic tasks must comply with all constraints from the nodes above. Computing *doneBefore* and *justDone* for each atomic task is non-trivial (see also Appendix D). We propose Algorithm 7 to compute *doneBefore*(at_i) and *justDone*(at_i) for any atomic task at_i, while solving any conflict arising from multiple compound task constraints applied to a single atomic task.

For each atomic task (inputATList), this algorithm checks the highest compound task

For each atomic task (*at* in inputATList), this algorithm checks for the list of compound tasks reachable through a *parent* relation (line 4). If this is empty, the atomic task is defined as part of a mission task. If not empty (line 5), it iterates over this list of compound tasks. If a **consecutive** constraint is found (line 8), all reachable atomic tasks from the task are returned as a list (line 9). Then, the atomic task to the left of at_i is added to *justDone* (line 12). If an **ordered** constraint is found (line 14), atomic subtasks from the ordered task and to the left of the path of constrained tasks, are added to *doneBefore*.

Algorithm 7 Compute *doneBefore*(at_i) and *justDone*(at_i).

```
Require: inputATList: list of atomic tasks
 1: procedure TRANFERCONSTRAINTSTOATOMICTASKS(inputATList)
 2:     While inputATList:
 3:         at = inputATList.pop(0)
 4:         listCTsAbove: list of compound tasks in order from the highest constraint compound task
 5:         if listCTsAbove is not empty:
 6:             While listCTsAbove:
 7:                 ct0 = listCTsAbove.pop(0)
 8:                 if ct0.consec:                                              ▷ consecutive constrained
 9:                     listAT = t.reachableAT get all reachable atomic tasks
10:                     atIndex = index of at in listAT
11:                     if (atIndex!=0):
12:                         at.justDone = listAT[atIndex-1]
13:                         return
14:                 if ct0.ordered:                                                   ▷ order constraint
15:                     if at in ct0.subtasks:          ▷ only get previous tasks if ‘at’ part of subtasks
16:                         atIndex=index of at in ct0.subtasks
17:                         if (atIndex!=0):                                    ▷ if not the only subtask
18:                             taskToBeDoneBefore=ct0.subtasks[0:atIndex-1]          ▷ get previous tasks
19:                             for (i in taskToBeDoneBefore) do:
20:                                 if i is compound task:
21:                                     at.doneBefore.append(i.reachableAT)      ▷ add reachable atomic
    tasks
22:                                 if i is atomic task:
23:                                     at.doneBefore.append(i)                        ▷ add atomic task
24:                             end for
25:                     else (at not in ct0.subtasks):
26:                         subCT = listCTsAbove[0]                               ▷ get compound task
27:                         ctIndex = index of subCT in ct0.subtasks      ▷ get index of compound task
28:                         if (ctIndex!=0):
29:                             for i=0; i==ctIndex-1; i++ do:                  ▷ iterate over subtasks
30:                                 subtask=ct0.asubtasks[i]
31:                                 if subtask is an atomic task:
32:                                     at.doneBefore.append(subtask)
33:                                 if subtask is a compound task:
34:                                     at.doneBefore.append(subtask.reachableAT)
35:                             end for
```

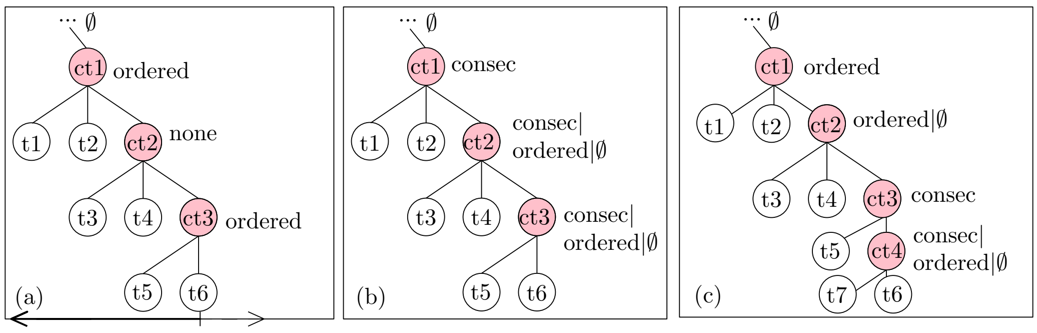


Figure 6.4: Task-trees shown higher node from *at*6 with a task constraint. In (a), the higher node has an ordered constraint; hence, it overtakes any constraint of tasks below if they are either order-constrained or with no constraint (∅). In (b), the highest constraint node has a consecutive constraint; hence, it overwrites any constraint from lower nodes. The image in (c) shows an example where the highest is an *ordered* and, at some lower node, there is a *consec* constraint. In this case, the *ordered* constraint overtakes any node until *consec*; thereafter, *consec* overwrites any node below.

Example 16. Figure 6.4 shows three examples of tree-like structures of compound tasks. We show how the algorithm works in these examples. We have an inputATList=$\{t1, t2, t3...t6, ..., tn\}$. Algorithm 7 obtains *doneBefore* and *justDone* for all atomic tasks. Here we show the output for $doneBefore(at6)$ and $justDone(at6)$. We only need the information on the task' tree-like structure from the left of task at7 (as shown with an arrow in Figure 6.4.a). Constraints are depicted within curly brackets, where ∅ means the compound task has no constraint.
In example (a), there are two compound tasks with order constraints (ct1 and ct3), hence,

$doneBefore(at6) = \{t1, t2, t5\}$
$justDone(at6) = null$

In example (b), there is a consecutive task ($ct1$) at the highest node, hence, this requires that all subtasks are done consecutively regardless of any other existing constraints (consecutive, ordered or none). constraint $t6$ to be done just after $t5$ is completed,

$doneBefore(at6) = \{\}$
$justDone(at6) = t5$

In example (c), $t6$ is constrained to be done just after $t7$ is completed (constraint transferred from $ct4$) and after $t1$ and $t2$ (constraint transferred from $ct1$),

$doneBefore(at6) = \{t1, t2\}$
$justDone(at6) = t7$

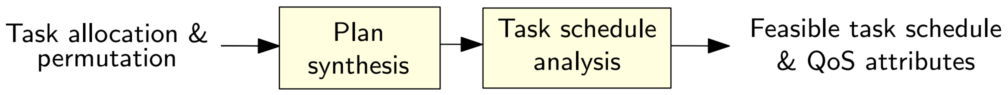

Task allocation & permutation
Plan synthesis
Task schedule analysis
Feasible task schedule & QoS attributes

Figure 6.5: High-level process to create a task schedule and its QoS attributes from a given allocation and permutation.

6.3 Task Scheduling in KANOA

Figure 6.5 shows the high-level process for generating a task schedule and a list of quality-of-service (QoS) attributes. Starting from a task allocation and a task permutation, the **plan synthesis** stage generates plans for each group of robots, if possible. If all robot plans are feasible, the **task schedule analysis** stage returns the feasible task schedule and the list of QoS attribute values. We provide more detail in the subsequent subsections.

6.3.1 Plan Synthesis

Section 6.1 defines a robot plan as a sequence of actions (*idle*, *travel* and *dotask*) and a task schedule as a set of these robot plans that comply with any requirement (from Table 6.1) defined in the KANOA problem specification. In this section, we are interested in finding task schedules that comply with constraints C1-C10. The problem of synthesising robot plans is approached as finding the adversary of an MDP modelling the behaviour of the robots, where the possible actions from an MDP state can cause a robot to idle or to complete the next task, as described in the following paragraphs.

Idling Model. We create an MDP $\mathcal{M}_{i,\bar{j},g}$ for each group of robots $g \in groups(i)$ in allocation i and permutation of tasks $\bar{j}$. $\mathcal{M}_{i,\bar{j},g}$ models the robot's actions, where a robot r in group g is modelled by the state variable s defined by the tuple,

$$s = (rorder, rtime, ridleT) \tag{6.6}$$

where *rorder* counts the tasks completed and increases by one every time robot r completes a task; *rtime* works as a clock increasing each time r travels and completes a task or when it spends time idling, while $ridleT$ counts only the time that r spends idling. An MDP state is given by the composition of each of the robot's state variables $s = (s_1, s_2, ..., s_n)$.

Variable *rorder* has a range of [0,n], where n is the number of tasks allocated to r by allocation i; *rtime* has a range of $[0, TT]$ set as the total time available to complete the mission (specified in the KANOA problem specification); and $ridleT$ a range of [0,*rmaxIdle*], where *rmaxIdle* is the maximum time that the robot is allowed to idle given the time limit TT. Time to idle, $ridleT$, can be omitted for any robot that lacks the time to idle while completing all assigned tasks within the specified time frame TT.

Observation 4. The maximum number of times a robot r can idle, *rmaxIdle*, is calculated beforehand as the total time minus the time spent travelling and doing all its tasks. The user can also specify the **hyperparameter** *maxIdle* directly in the KANOA tool to limit the idling time available for each robot. This is useful from an application perspective when we would like to synthesise plans with a bound on the time that they are not doing any task. From the methodology perspective, adding a bound on the idle time can result in a large reduction in the state space as thoroughly explained in Appendix E.

Observation 5. A bound on the possible idling time is also found in classical scheduling techniques such as job shop problems: "*In a general job shop system the number [of tasks] is arbitrary. Usually, in such systems, it is assumed that buffers between processors have unlimited capacity and a job after completion on one processor may wait before its processing starts on the next one. If, however, buffers are of zero capacity, jobs cannot wait between two consecutive processors, thus, a no-wait property is assumed*" [42].

We describe the MDP model with the help of the example in Figure 6.6. On the left, Figure 6.6a shows a representation of the Markov model for robot $r1$[7]. Robot $r1$ starts with two actions available: (1) travel and perform t1 (r1t1) or (2) idle (r1idle). When t1 is done, $r1$ can idle or travel and perform t2. When t1 and t2 are completed, $r1$ is done. All actions lead to a single state, i.e., transition probabilities are set to 1.

The maximum idle time for each robot (r$_i$idle) is constrained by the mission completion time (denoted as *time* in KANOA's DSL) minus the total time spent on travelling and completing assigned tasks. The last two values are known given the allocation and the permutation. Ideally, the idle time should be kept as small as possible to prevent state explosion; however, determining what qualifies as "small enough" can be challenging. We provide some insights into the selection of an idling time in Appendix E.

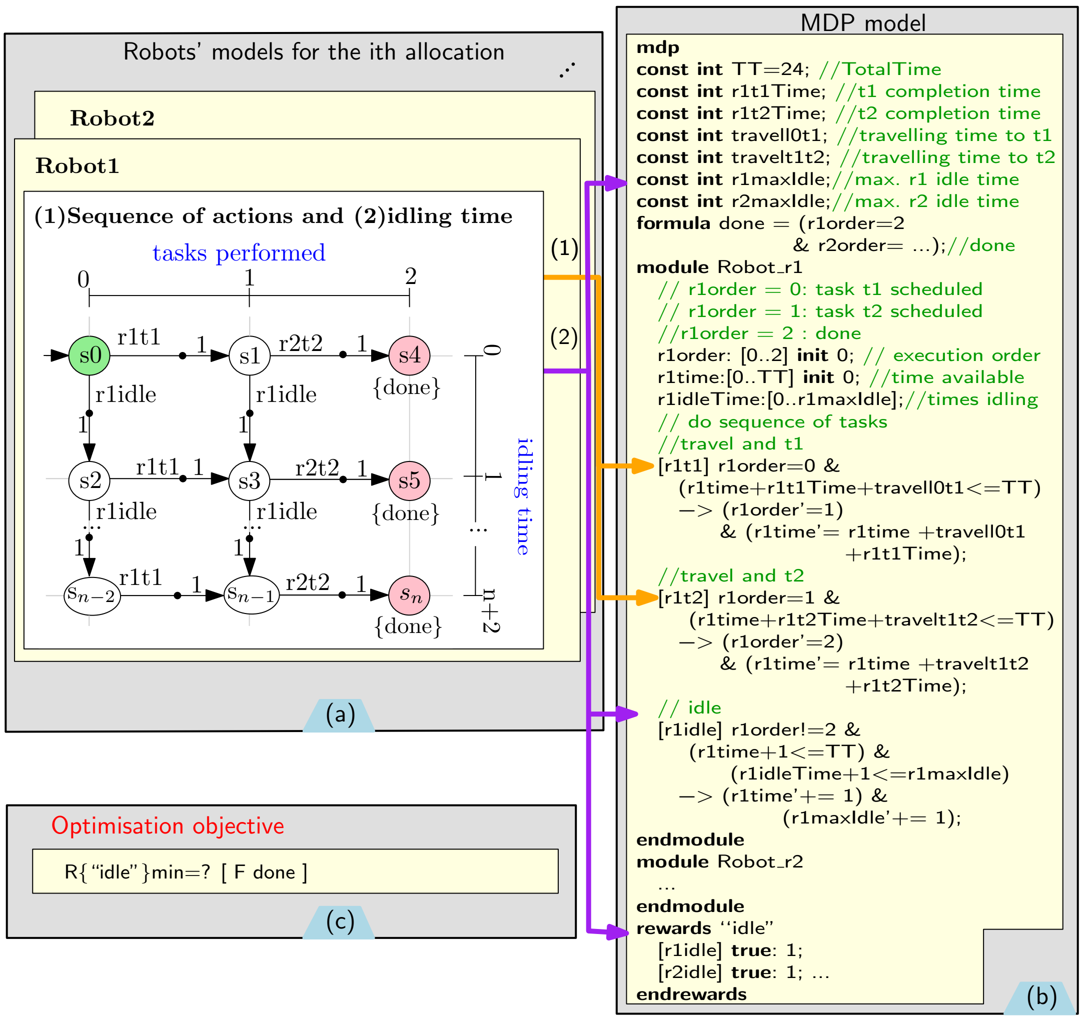
On the right, Figure 6.6b shows the MDP written in the PRISM language. Starting with the constant variables, TT is the total time available set to 24 time units. `r1t1Time` is the completion time for task t1 by robot $r1$. `travell0t1` is the travelling time from the initial location to the location where t1 is performed. The Boolean formula,

$$\textbf{formula}\ \texttt{done =(r1order=2 \& r2order=...)} \tag{6.7}$$

is True only when all robots have completed their tasks.

PRISM modules are used to model each robot separately. Robot $r1$ behaviour is

[7]We follow the way MDPs are represented from [60]. States are represented as circles. Outgoing arrows from a state are possible actions. Actions can have labels (e.g., $r1t1$). The number between states s$_i$ and s$_{i+1}$ is the probability of transitioning from s$_i$ to s$_{i+1}$; in our example, all non-final states have non-deterministic actions with a probability of 1 to reach the next state.



```
mdp
const int TT=24; //TotalTime
const int r1t1Time; //t1 completion time
const int r1t2Time; //t2 completion time
const int travell0t1; //travelling time to t1
const int travelt1t2; //travelling time to t2
const int r1maxIdle;//max. r1 idle time
const int r2maxIdle;//max. r2 idle time
formula done = (r1order=2
                 & r2order= ...);//done
module Robot_r1
  // r1order = 0: task t1 scheduled
  // r1order = 1: task t2 scheduled
  //r1order = 2 : done
  r1order: [0..2] init 0; // execution order
  r1time:[0..TT] init 0; //time available
  r1idleTime:[0..r1maxIdle];//times idling
  // do sequence of tasks
  //travel and t1
  [r1t1] r1order=0 &
    (r1time+r1t1Time+travell0t1<=TT)
    -> (r1order'=1)
        & (r1time'= r1time +travell0t1
                    +r1t1Time);
  //travel and t2
  [r1t2] r1order=1 &
     (r1time+r1t2Time+travelt1t2<=TT)
    -> (r1order'=2)
        & (r1time'= r1time +travelt1t2
                    +r1t2Time);
  // idle
  [r1idle] r1order!=2 &
     (r1time+1<=TT) &
         (r1idleTime+1<=r1maxIdle)
    -> (r1time'+= 1) &
              (r1maxIdle'+= 1);
endmodule
module Robot_r2
  ...
endmodule
rewards "idle"
  [r1idle] true: 1;
  [r2idle] true: 1; ...
endrewards
```

Figure 6.6: Model for the synthesis of plans for individual robots under task constraints and minimising the idling time as the optimisation objective. On the left, (a) shows a representation of the transition model. On the right, (b) the MDP model in PRISM language. The *idle* optimisation parameter is shown in (c). The layout in (a) is inspired by [173]).

defined in **module** `Robot_r1` with state variables `r1order` and `r1time`[8]. The orange arrows from Figure 6.6 shows the transitions to a next task; for example,

```
[] r1order=0 & (r1time+r1t1Time+travell0t1<=TT)
                    -> (r1done'=1)&(r1time'=r1time+travell0t1+r1t1Time);   (6.8)
```

represents the transition of $r1$ from an initial location into the location of t1. The *guard* checks that the robot is at the initial location (`r1order=0`) and that there is enough time to travel from the initial location of robot $r1$ to the location where t1 has to be done (`travell0t1`), as well as enough time to complete t1 (`travell0t1`),

```
(r1time+r1t1Time+travell0t1<=TT)                                        (6.9)
```

[8]State variables $\overline{r_1 tasks}$ depend on task constraints, as explained later.

The *update* moves the robot to a state where t1 is completed (`r1order'=1`) and the time of $r1$ reflects the travelling time and the time needed to complete t1 (`r1time'=r1time+travel10t1+r1t1Time`).

Next, the robot has the option to idle for one time step at a time, e.g.,

```
[r1idle] r1order!=2 & (r1time+1<=TT) -> (r1time+=1);
```
(6.10)

Robot $r1$ can choose to idle if it hasn't completed all tasks ($r1order! = 2$) and still has time available (`r1time+1<=TT`). When the transition is taken, r1's time updates ($r1time+ = 1$). Moreover, the *action* label `[r1idle]` is used to compute the idling time for robot r1 through $r1idle$ reward structure,

```
reward "r1idle"
    [r1idle] true:  1;
    ...
endreward
```
(6.11)

acting as a counter tracking the number of times a robot idles. These idle-related transitions are shown in Figure 6.6 by purple arrows.

Optimisation Objective. Finally, Figure 6.6c depicts the RPCTL formula used for the synthesis of the robot plans, `R{"idle"}min=?[F done]`. These read as follows: *what is the minimum "idle" reward such that eventually the state done is reached.* We use the PRISM Probabilistic Model Checker (PMC) to compute an adversary that, given Model A, minimises the robot's idle time. If an adversary is not found, no strategy complies with the set of constraints, for example, the tasks cannot be completed within the time available TT or, as formalised in the following paragraphs, no robot plan can comply with the set of task constraints (C5-C8). The outcome of the PMC is the set of **individual robot plans** and the **total idle time** required to complete the mission.

Plan synthesis. We generate robot plans by synthesising an MDP policy that minimises the idling time (R"idle"min=? [F done]). As described in background Section 2.3.3, an adversary resolves the non-determinism in an MDP. We are interested in deterministic policies such that, at any state s, we know exactly what action to take. Hence, our policy σ is a point distribution such that,

$$\alpha(s) = a \tag{6.12}$$

where a is a possible action for state s, $a \in A(s)$. We use the PRISM probabilistic model checker for the synthesis of adversaries for reward properties. Hence, a **feasible task schedule** (for the ith allocation and the $\bar{j}$ permutation) is obtained iff an adversary can be obtained for each robot group g in allocation i.

```
mdp
...
module Robot_r1
   // r1order = i: travel and do task jt1
   ...
   [jt1] r1order=i & (r1time+travelTi+r1jt1<=TT)
            & (r1time+travelTi = r2time+travelTj)
                 −> (r1order'=i+1) & (r1time'+=travelTi+r1jt1);
   ...
endmodule

module Robot_r2
   // r2order = j: travel and do task jt1
   ...
   [jt1] r2order=j & (r2time+travelTj+r2jt1<=TT)
            & (r1time+travelTi = r2time+travelTj)
                 −> (r2order'=j+1) & (r2time'+=travelTj+r2jt1);
   ...
endmodule
```

Figure 6.7: Joint task jt1 in robot $r1$ written in PRISM language as two transitions, first travel, then synchronise with other robots via the action label $[jt1]$. All transitions in other modules, with the same action label, "transition" at the same time. Changes compared to unconstrained tasks are highlighted in blue.

It is possible that the synthesis of an adversary is infeasible (for the ith allocation and $\bar{\jmath}$th permutation), due to violations of the task constraints explained in Section 6.3.2 and modelled as part of $\mathcal{M}_{i,\bar{\jmath},g}$.

6.3.2 Modelling Task Constraints

This section introduces the modelling of the task constraints as part of the MDP model, $\mathcal{M}_{i,\bar{\jmath},g}$. Joint tasks require the synchronisation between robot modules in the PRISM model so that robots meet in space and time to perform a particular task. On the other hand, consecutive and ordered tasks require that robots know when the previous task(s) has been completed to allow the next to start. Finally, constraints associated with the time that a task must have started or finished, require that robots can only perform these tasks within specific time windows. We formalise each of these constraints in the following paragraphs.

a. Joint tasks (constraint C5)

Figure 6.7 shows an example where robot $r1$ has allocated a **joint task** jt1 to be performed with the help of robot r2. Joint tasks require robots to start a task at the same time. This is modelled as part of the MDP transition's guard when $r1$ and $r2$ are ready to perform task jt1 (when $r1order = i$ and $r2order = j$, respectively). A proposition is added to check that both robots are at the same time ready to start jt1,

```
(r1time+travelTi = r2time+travelTj)                    (6.13)
```

where `travelTi` is the time needed by robot $r1$, to travel to perform jt1 from its current location (i.e., the location after performing the task at `(r1order=i-1)` or from its initial position if `(i=0)`). Similarly for `travelTj` and robot $r2$.

Robot variables transition simultaneously due to the *PRISM action* `[jt1]`. This means that in a single transition, different state variables belonging to different modules change at the same time. In this example, state variables related to the *order* and *time* of $r1$ and $r2$.

b. Ordered tasks (constraint C6)

Figure 6.8 shows an example of two ordered atomic tasks, r1t1 and r2t2. These are allocated to robots $r1$ and $r2$, respectively. Task r1t1 has to be done before r2t2. For each atomic task with an ordered constraint, a Boolean **formula** is added to check when the task is completed,

```
formula r1t1Done = r1order>=k+1;
formula r2t2Done = r2order>=j+1;                      (6.14)
```

When robot $r2$ has to perform r2t2 (at `r2order=j`), it can only perform this task when the previous task has been done, hence the proposition (`r1t1Done`) is added into the transition's guard. Also, as each robot has its own time as a state variable `r`$_i$`time`, the transition has to be taken only if the last robot has completed the task *in the past.* In other words, $r2$ is at the location of task r2t2 and ready to perform this task after $r1$ completed r1t1Done,

```
r2time+travelT2>=r1time                               (6.15)
```

Note that if both tasks are assigned to the same robot, the guard `r`$_i$`time>=r`$_i$`time` can be removed.

Finally, the update is extended to state that r2t2 has been completed

```
r2t2'=true                                            (6.16)
```

This is needed if, for example, other tasks require r2t2 to be completed.

c. Consecutive tasks (constraint C7)

Figure 6.9 shows an example of two consecutive tasks, r1t2 and r2t3. These are allocated to robots $r1$ and $r2$, respectively. Task r2t3 must start only *immediately after* r1t2 is completed. Task r1t2 starts at `r1order=j` and it '*just*' finishes at `j+1`, when the robot

```
mdp
formula r1t1Done = r1order>=k+1 ; //TaskDone
formula r2t2Done = r2order>=j+1; //TaskDone
...
module Robot_r2
  // r2order = j: do task r2t2
 ...
  [ ] r2order=j & (r2time+r2t2Time+travelT2<=TT)
         & (r2time+travelT2 >= r1time) & (r1t1Done)
             -> (r2order'=j+1)
                    & (r2time'+=r2t2Time+travelT2);
...
endmodule
```

Figure 6.8: Ordered tasks modelled as part of the MDP model guards and updates. Boolean variables are added to check when a task is completed. Changes compared to unconstrained tasks are highlighted in blue.

is ready to perform the next task. Hence, the following formulae are added to captured when robots *just* completed tasks r1t2 and r2t3,

```
formula r1t2Just=r1order=j+1;
formula r2t3Just=r2order=k+1;
```
(6.17)

A special case happens if a task with a consecutive constraint is also a **joint task**. For example, if *r1t2* is also a joint task, we consider the longest task completion time among the robots allocated with this task.

To model consecutive tasks, two conditions are added to the guard. First, we check that robot $r2$ has just completed task r2t3, i.e., `(r2order = k+1)`. The second checks that robot $r1$ is synchronised in time with r2 to start r1t2. As we must check $r1$ time after it has travelled to the desired location, this is modelled as,

```
(r1time)+travelT2=r2time
```
(6.18)

d. Task constrained by time (constraint C8)

Time constraints can be defined in the KANOA's mission specification defining the time after a task can start or the time by which it has to finish (`start after` or `end before`). Figure 6.10 shows an example where robot $r1$ is allocated tasks with time constraints.

In Figure 6.10.a, task $r1t4$ has to *start after* 10 time units. Hence, the guard's transition (at `r1order=i`) has an additional proposition. This states that the robot's clock must be at least 10-time units by the time it is ready to start its task (i.e., after travelling to the location where r1t4 has to be performed),

```
(r1time+travelT4>=10)
```
(6.19)

Similarly, in Figure 6.10.b, task $r1t5$ has to *end before or at* 15-time units. Hence, the additional guard's proposition (at `r1order=j`) checks the robot's clock to be lower or

```
mdp
formula r1t2Just = r1order=j+1; //'just'done
formula r2t3Just = r2order=k+1 ; //'just'done
...
module Robot_r1
  // r1order = j: do task r1t2
  ...
endmodule
module Robot_r2
  // r2order = k: do task r2t3
...
  [ ] r2order=k & (r2time+r2t3Time+travelT3<=TT)
         & (r1t2Just) //r1t2 'just' done
         & (r2time+travelT2 = r1time) //even iff same robot
           -> (r2order'=k+1)
                  & (r2time'+=r2t3Time+travelT3);
...
endmodule
```

Figure 6.9: Consecutive tasks modelled as part of the MDP model *guards* (in blue). In this example, task r1t2 has to be performed *immediately after* r2t3.

equal to 16 by the time the task is completed,

$$\texttt{(r1time+travelT5+r1t5Time<=15)} \tag{6.20}$$

Observation 6. Why do we model task constraints as part of the MDP and not as temporal logic formulae? It is possible to formulate some of the task constraints as RPCTL* formulae rather than model them as part of the MDP transitions. Consider the example of synthesising a robot plan for $r1$ that minimises the idle time (`R{"idle"}`) to reach a state where all tasks are completed (`[F done]`), and task r1t1 has to be performed in the first 15 time units. If task r1t1 is the third task to be done by $r1$, this can be written as `R{"idle"}[(F done) & (G((r1order=3) ⇒ (r1time>15)))]`. We opt to encode the task constraints directly into the model to reduce the number of states where possible, mitigating the state-explosion problem. Moreover, the probabilistic model checking of more complex formulae (compared to the reachability formulae proposed) requires more computation time.

6.3.3 Task Scheduling Analysis

We describe the task scheduling problem in Section 6.1 as finding task schedules (i.e., sets of robot plans) that comply with a set of requirements. These requirements encompass constraints and optimisation objectives detailed in Table 6.1. Optimisation objectives can also be referred to as quality-of-service (QoS) **attributes** to be minimised or maximised [50].

KANOA's DSL allows to specify one, two, or all three optimisation objectives $O \subseteq \{O1, O2, O3\}$, as defined in Table 6.1. For consistency and without loss of generality, we

```
mdp
...
module Robot_r1
  // r1order = i: do task r1t4
 ...
  [ ] r1order=i
        & (r1time+r1t4Time+travelT4<=TT)
        & (r1time+travelT4 >= 10)
            −> (r1order'=i+1)
             & (r1time'+=r1t4Time+travelT4);
...
endmodule
```

a)

```
mdp
...
module Robot_r1
  // r1order = j: do task r1t5
 ...
  [ ] r1order=j
        & (r1time+r1t5Time+travelT5<=TT)
        & (r1time+travelT5+r1t5Time <= 15)
            −> (r1order'=j+1)
             & (r1time'+=r1t5Time+travelT5);
...
endmodule
```

b)

Figure 6.10: Time constraints modelled as part of the MDP model. As an example, robot $r1$ has allocated task r1t4, which must start after 10-time units (a), and r1t5 before 15 (b).

assume that all attributes are intended to be minimized. In the case of "maximising the probability of success", we rephrase it as "minimise the probability of failure". We are interested in computing the optimisation attributes $attr_{1-3}$ associated with O1–O3 for each allocation i and permutation $\bar{j}$. Since robots in $(i, \bar{j})$ are divided into g groups, we initially derive attribute values for each group and then combine them. For example, KANOA calculates the probability of mission success for each group g in $(i, \bar{j})$ and computes the product of these probabilities across all groups to obtain the overall value of attribute $attr_1$[9]. We compute each attribute as follows,

$$attr_1 = \prod_g PMC(\mathcal{M}_{prob,g}, \phi_{prob}), \tag{6.21}$$

$$attr_2 = \sum_g PMC(\mathcal{M}_{idle,g}, \phi_{idle}), \tag{6.22}$$

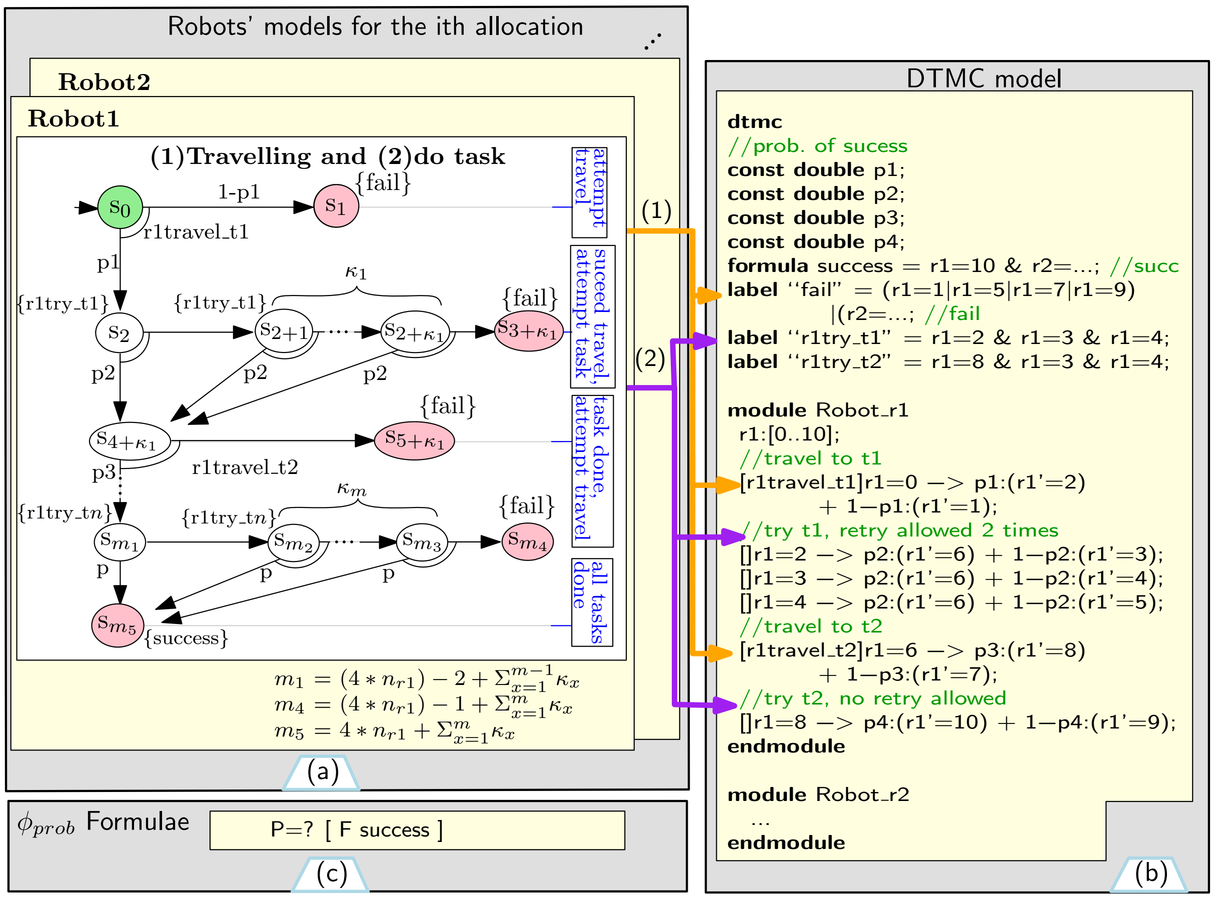
$$attr_3 = \sum_g f(\mathcal{M}_{travel,g}) \tag{6.23}$$

Attribute $attr_1$ indicates the probability of succeeding with a mission; it uses probabilistic model checking for the qualitative verification of the discrete Markov chain model, $\mathcal{M}_{prob,g}$ under property "ϕ_{prob} =P=? [F success]". The second attribute, $attr_2$, obtains the total robot idling time using the MDP model, $\mathcal{M}_{idle,g}$, and the property "ϕ_{prob} =R{"idle"}min=? [F done]". The third attribute, $attr_3$, refers to the total travelling cost of the robots captured by model $\mathcal{M}_{travel,g}$ and computed by a function f.

We refer to $\mathcal{M}_{(prob|idle|travel),g}$ as **specialised models** as they capture different aspects of the task scheduling process. We specify each of these as follows,

- **Probability of success model** ($\mathcal{M}_{prob,g}$). This DTMC models the probability of successfully transitioning between locations and successfully completing tasks. Each robot r in group g is modelled as a state variable s defined by the tuple, s=(rorder), where

[9]As these models depend on $i, \bar{j}$ we can write these as in Section 6.3.1, for example, $\mathcal{M}_{prob,i,\bar{j},g}$.



```
dtmc
//prob. of sucess
const double p1;
const double p2;
const double p3;
const double p4;
formula success = r1=10 & r2=...; //succ
label ‘‘fail” = (r1=1|r1=5|r1=7|r1=9)
              |(r2=...; //fail
label ‘‘r1try_t1” = r1=2 & r1=3 & r1=4;
label ‘‘r1try_t2” = r1=8 & r1=3 & r1=4;

module Robot_r1
 r1:[0..10];
 //travel to t1
 [r1travel_t1]r1=0 −> p1:(r1’=2)
          + 1−p1:(r1’=1);
 //try t1, retry allowed 2 times
 []r1=2 −> p2:(r1’=6) + 1−p2:(r1’=3);
 []r1=3 −> p2:(r1’=6) + 1−p2:(r1’=4);
 []r1=4 −> p2:(r1’=6) + 1−p2:(r1’=5);
 //travel to t2
 [r1travel_t2]r1=6 −> p3:(r1’=8)
          + 1−p3:(r1’=7);
 //try t2, no retry allowed
 []r1=8 −> p4:(r1’=10) + 1−p4:(r1’=9);
endmodule

module Robot_r2
  ...
endmodule
```

Figure 6.11: DTMC Model for the probability of succeeding with a mission without failure.

rorder tracks the robot's travelling, task attempts and failures. We describe the DTMC model with the help of Figure 6.11. On the left, Figure 6.11a shows the representation of the Markov model for robot $r1$. Robot $r1$ starts from state s0. It takes action labelled r1travel_t1, and with probability p1 it travels to the first task location (s2), while with probability 1-p1 fails (s1). At state s2 (labelled {r1try_t1}), the robot attempts to perform its first task and succeeds with probability p2. If it fails, it attempts to complete the task κ_1 times, each with a probability of p2 of succeeding and 1-p2 of failing. At the last attempt, if the robot fails to complete the task it ends in state $s_{3+\kappa_1}$. If it succeeds ($s_{4+\kappa_1}$), it travels to the next task with probability p3 and fails with 1-p3, and so on until the last task is completed at state s_{m_5}.

Note that each task is modelled using four states plus the times allowed for retry. For example, the first task requires states (s0, s1, s2 and $s_{3+\kappa_1}$) to travel (or fail travelling) and complete the task (or fail the task), plus κ_1 states to retry in case of a failed attempt. Hence, in s_{m_5}, $m5 = 4 * n_{r1} + \Sigma_{x=1}^{m}\kappa_x$. Let n_r be the total number of tasks for any robot r in allocation a_i, the total number of states is given by,

$$n_s = \Sigma_{r \in a_i}(4 * n_r + \Sigma_{i=1}^{m}\kappa_i) + (\#\,\mathrm{dom}\ a_i) \tag{6.24}$$

where the number of robots ($\#\,\mathrm{dom}\ a_i$) is added to account for the initial states (represented with index 0 for $r1$).

Figure 6.11b shows the DTMC written in the PRISM language, starting with: the probability values declared as constants p1,...,p4; the formula done defined as the conjunction of states where robots completed successfully all their tasks; label fail defined as the conjunction as a robot failing (at any state, declare by disjoint states for each robot); and, for explainability purposes, labels defining the states where robots are attempting to complete a task r1try_t1, r1try_t2. The robot's behaviour is defined by PRISM modules, where the state variable, s=(rorder), introduced before, is defined as the robot's name. For robot $r1$, this is declared as r1:[0..10];. As this robot has two tasks, t1 and t2, and t1 is allowed to be retried 2 times, the last state is 4*2+2=10 and the total number of states (for $r1$) is 11 as defined in Equation 6.24 (without counting robot $r2$). The rest of the PRISM module in this figure is self-explanatory.

Finally, Figure 6.11c depicts the PCTL formula P=?[F success], read as "*what is the probability of completing the mission (all tasks) successfully (without any failure)?*". We use this formula and the PRISM model checker tool to compute the attribute $attr_1$.

- **Idle time model** ($\mathcal{M}_{idle,g}$). Attribute $attr_2$ is obtained as a byproduct of the plan synthesis; when generating a plan that minimises the robots' idling time, the minimum idle time for the given (allocation,permutation) pair is also returned. Hence, Figure 6.6 depicts ϕ_{idle} and $\mathcal{M}_{prob}$ (denoted as $\mathcal{M}_{i,\bar{j},g}$ in Section 6.3.1).

- **Travelling cost model** ($\mathcal{M}_{travel,g}$). The travel cost model incorporates the *travel* actions (along with their corresponding times) extracted from plans generated during plan synthesis. The function f sums the travel times in these plans for each group, and subsequently, these values are combined to determine the total travel time associated with $(i, \bar{j})$.

Observation 7. Attribute $attr_1$ represents the probability of the mission being completed successfully. In the next section, we set the optimisation problem as the minimisation of all attributes. Hence, we are interested in minimising the probability of mission failure calculated as $1 - attr_1$. It is worth noticing that this is not the same as computing the probability of each group failing by modifying the PCTL formula ϕ_{prob} to 1-P=?[F success], as visualised in the example on Figure 6.12. This variant would minimise "the probability of (strictly) all tasks failing", as opposed to the failure of one or more tasks, which would result in mission failure. While KANOA minimises the probability of mission failure, its users may consider this alternative if it aligns better with the specific requirements of their application.

The actions in the MDP and DTMC models are consistent with each other. To establish this, we need to show that each PRISM robot module in the MDP model behaves identically to its counterpart in the DTMC. Both models are based on the same task permutation $\bar{j}$ and the same group of robots g. Each robot $r \in g$ has a module in the

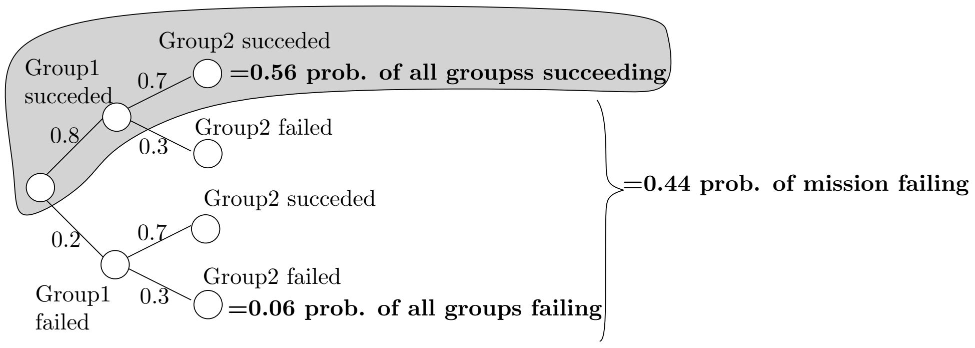


Figure 6.12: Example of calculating the probability of success.

MDP that follows the task sequence defined by $\bar{j}$ along with potential non-deterministic actions for idling. This task sequence is preserved in the DTMC model since it follows the same ordered sequence of tasks. Since the idling time is not treated as probabilistic, it can be omitted in the DTMC. As a reference to the reader, the conversion between MDPs and DTMCs is well-documented in the literature; for an example of this process, we refer the reader to [78] (Theorem 1).

6.4 Task Scheduling Optimisation.

Thus far, we described the plan synthesis process and generation of multiple specialised models to obtain QoS attributes. These attributes are computed for a given allocation and permutation pair, denoted as $(i, \bar{j})$. As mentioned briefly in the chapter's introduction, finding *optimal* solutions for a large number of robots and tasks is unfeasible as the solution space grows linearly with the number of robots but factorially with the tasks (for each task allocation). The solution space size (SS)[10] can be computed as follows. For each allocation a_i, the potential solutions are the multiplication of each robot's permutation of tasks. Hence,

$$\#SS = \sum_{a_i \in alloc} \prod_{r \in \text{dom } a_i} \#(a_i(r))!$$

where $\prod_{r \in \text{dom } a_i} \#(a_i(r))! = \#SS_i$ is the solution space size of allocation a_i.

Example 17. From Example 11, the four potential solutions for allocation a_i are: $\bar{j}_1 = \langle 1, 1\rangle$, $\bar{j}_2 = \langle 1, 2\rangle$, $\bar{j}_3 = \langle 2, 1\rangle$, $\bar{j}_4 = \langle 2, 1\rangle$. In this case, as the solution space is *small*, it is possible to evaluate each of these solutions. However, if robot $r1$ were allocated 8 tasks, this would be 8!×2!=80640 possible solutions for a_i.

[10]Without considering any constraints that can be specified in the problem specification and which may reduce this number.

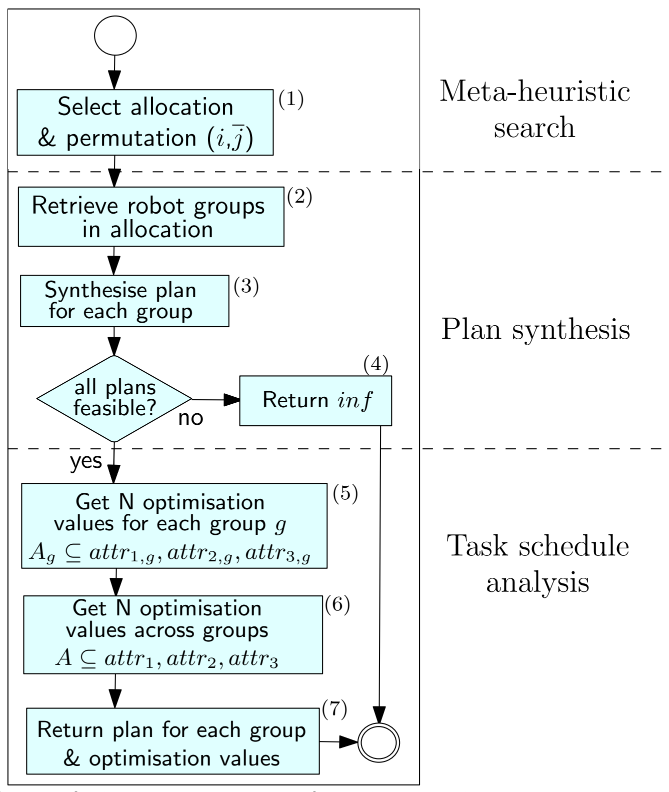

Select allocation & permutation $(i,\bar{j})$
(1)
Meta-heuristic search
Retrieve robot groups in allocation
(2)
Synthesise plan for each group
(3)
Plan synthesis
all plans feasible?
no
Return inf
(4)
yes
Get N optimisation values for each group g $A_g \subseteq attr_{1,g}, attr_{2,g}, attr_{3,g}$
(5)
Task schedule analysis
Get N optimisation values across groups $A \subseteq attr_1, attr_2, attr_3$
(6)
Return plan for each group & optimisation values
(7)

Figure 6.13: Workflow of the generation of a plan and computation of its attribute values for its evaluation.

To generate task schedules for large solution space multi-robot problems, KANOA utilises metaheuristic search based on genetic algorithms (GA),[11], producing approximate Pareto-optimal task schedules with respect to O for each task allocation. As previously said, the GA search method consists of the following steps: initialisation, evaluation (fitness assessment), selection, crossover, mutation and replacement. An initial population consisting of permutations $\bar{j}$ from allocation i are generated in the **initialisation** step. The **evaluation** (or fitness assessment) consists of finding the attribute values for potential solutions $(i,\bar{j})$, as explained in the following paragraphs. The fitness score for each solution is determined based on (up to) three objective functions, or *fitness functions*, defined in KANOA. The last four steps are carried out as dictated by the GA algorithm.

Figure 6.13 illustrates the process for the evaluation of potential solutions, referred to as individuals. Initially, the meta-heuristic search algorithm generates a population of potential solutions, i.e., a set of task permutations $\{\bar{j}_1, \bar{j}_2, ...\}$ for an allocation i. For each individual (1), KANOA retrieves the groups of robots in allocation i (2). Within each group, KANOA synthesizes robot plans according to the methodology outlined in Section 6.3.1 (3). If it is unable to generate a plan for all groups, the evaluation stage returns *inf*inite for all attributes (4), as the optimisation process aims to minimise these values. Conversely, if successful, the value of each attribute is computed for each group (5),

[11]See background Section 2.4.

followed by aggregation across all groups (6). These steps are summarised by (6.22) and (6.23). Finally, the optimisation values and the $(i, \bar{\jmath})$ pair are returned (7).[12]

We omitted the (implicit) reliance of equations 6.22-6.23 on an allocation i and a task permutation $\bar{\jmath}$. Let us refer to these equations as $attr_\iota(i, \bar{\jmath})$, where ι is the number of the attribute[13], and $i, \bar{\jmath}$ the variables from which the models are constructed. The output of the task scheduling optimisation process is a Pareto-optimal set PS of solutions satisfying constraints C1-C10, *non-dominated* with respect to the optimisation objectives $O \subseteq$ {O1,O2,O2}, $PS_i = \{\bar{\jmath} \in SS_i \mid (\nexists \bar{\jmath}\text{'} \in SS_i \bullet \bar{\jmath}\text{'} \prec \bar{\jmath})\}$ with the dominance relation $\prec: SS_i \times SS_i \to \mathbb{B}$ defined by,

$$\begin{aligned}&\forall \bar{\jmath}_1, \bar{\jmath}_2 \in SS_i \bullet \bar{\jmath}_1 \prec \bar{\jmath}_2 \Leftrightarrow \\ &\qquad (\forall \iota \in \mathbf{L} \bullet attr_\iota(i, \bar{\jmath}_1) \leq attr_\iota(i, \bar{\jmath}_2)) \land (\exists \iota \in \mathbf{L} \bullet attr_\iota(i, \bar{\jmath}_1) < attr_\iota(i, \bar{\jmath}_2))\end{aligned}$$

where $\mathbf{L} \subseteq \{1, 2, 3\}$ s.t. $k \in \mathbf{L}$ iff O$k \in O$. We define the Pareto front PF as the set of the attribute values corresponding to all the solutions from PS.

Observation 8. PRISM supports the specification of multi-objective properties [174]. However, as we opt to generate multiple specialised models, this feature cannot be exploited in the current version of KANOA. As a side note, in previous versions of KANOA [30, 93, 122], a single MDP model captured the behaviour of the robots for a given permutation. However, this PRISM feature could not be exploited either, because: (a) the property used to reason about the travelling cost was not a cumulative reward property (e.g., "what is the expected travel cost within t time units", written as, R=? [C<=t]) which is the only type of reward property accepted in multi-objective PRISM properties; and (b) PRISM only supports transition rewards in this context; state rewards are not [174].

[12]Notice that the $(i, \bar{\jmath})$ pair has shown to be a feasible solution (as all synthesised plans were feasible) and it contains enough information to synthesise the robots' plan. Hence, Figure 6.13 shows that the robot plans synthesised plans are returned.

[13]The values of ι consist of the indices of the optimisation objectives, s.t. O$\iota \in O$.

6.5 Hospital Case Study

In this section, we show the grouping of robots and the attribute values' generation for the hospital case study. Further evaluation is provided in Chapter 7 as part of the KANOA's tool evaluation. The hospital case study is introduced in Section 4.3.[14]

We assume that multiple task allocations are given, so robots know what tasks they must carry out but do not know the order of these tasks, and we consider an allocation that consists of deploying four robots $R = \langle r3, r4, r5, r1 \rangle$. The pre-scheduling stage identifies three groups of robots, $\{\{r3\}$,$\{r4, r5\}$, $\{r1\}\}$. As part of the multi-objective optimisation loop, the permutation $\bar{j} = \langle 78, 2, 2, 600 \rangle$ is evaluated.

We explain three models in this section, two MDPs and one DTMC; the rest follow the same structure and are provided in Appendix F. The configuration of the metaheuristic search and other KANOA's tool hyperparameters are explained in Chapter 7.

MDP, robot group 1. The MDP for group one is shown in Listing 6.1. Line 2 shows the time limit set to 120 minutes; lines 4-9 the travelling time between locations of its assigned tasks; lines 10-12 the time to complete each task; line 13 shows the maximum idling time, set to 20. Line 14 shows the formula *done*, defined as the state variable $r3done = 6$, coinciding with the robot completing the last task. Lines 16-17 show the constraints for ordered tasks. The behaviour of $r3$ is described in lines 19-32. Lines 20-22 show the state variables to track the order ($r3order$), the robot's clock ($r3time$) and the robot's time spent in idling ($r3idleTime$).

Line 23 completes task $at4\_9$, as shown in the label between square brackets. As this is the first task, the robot starts at $r3order = 0$. The second part of the guard checks if enough time is available to travel to $at4\_9$ location and complete it. The transition updates $r3order' = 1$ to move on to the next task and increases the robot's clock (by the time it took to travel and complete the task).

In line 25, the robot completes tasks $at2\_11$. As this task can only be performed after $at4\_9$, the guard also checks that the previous task *has been done in the past*, shown highlighted in yellow. This is previously explained in Figure 6.8. In this special case, as both tasks are assigned to the same robot, it is trivially true that the robot completed the previous task in the past ($r3time + travelr3at3\_10 >= r3time$). The update part in this line and the following tasks (lines 28-33) are similar to the previous task.

Line 32 shows the idle transition that the robot can take at any time as long as it has not completed its tasks ($r3done! = 6$) and there is time to idle a one-time unit ($r3time + 1 <= TT$). Lastly, line 36 shows the cost of 1 added to the "idle" reward structure when transition [r3idle] is taken.

[14]We modify the success rate in the hospital case study to be more than 20% for KANOA to generate solutions. While a 20% success rate might seem modest, its calculation method, as detailed in Figure 6.12, accounts for this perception.

```
mdp
const int TT=120;//total time available
//r3 velocity:2.0
const int travelr3at4_9=4 ;// from location: l3 (robot initial loc) to location: room4 (at4_9) distance:8
const int travelr3at2_11=0 ;// from location: room4 (at4_9) to location: room4(at2_11) distance:0
const int travelr3at3_10=0 ;// from location: room4 (at2_11) to location: room4(at3_10) distance:0
const int travelr3at4_12=2 ;// from location: room4 (at3_10) to location: room5(at4_12) distance:4
const int travelr3at3_13=0 ;// from location: room5 (at4_12) to location: room5(at3_13) distance:0
const int travelr3at2_14=0 ;// from location: room5 (at3_13) to location: room5(at2_14) distance:0
const int r3at4_9Time=1; const int r3at2_11Time=6;
const int r3at3_10Time=6; const int r3at4_12Time=1;
const int r3at3_13Time=6; const int r3at2_14Time=6;
const int maxIdler3=20;
formula done=(r3order=6);
//formulae for ordered tasks
formula r3at4_9Done =r3order>=1 ;
formula r3at4_12Done =r3order>=4 ;

module r3
 r3order:[0..6];
 r3time:[0..120];
 r3idleTime:[0..maxIdler3];
 [r3at4_9] r3order=0 &(r3time+r3at4_9Time+travelr3at4_9<=TT) →(r3order'=1) & . . .
      (r3time'=r3time+r3at4_9Time+travelr3at4_9);

 [r3at2_11] r3order=1 &(r3time+r3at2_11Time+travelr3at2_11<=TT) &
     (r3time+travelr3at2_11 >= r3time) & (r3at4_9Done) →(r3order'=2) & . . .
         (r3time'=r3time+r3at2_11Time+travelr3at2_11);

 [r3at3_10] r3order=2 &(r3time+r3at3_10Time+travelr3at3_10<=TT) &(r3time+travelr3at3_10 >=r3time) & . . .
      (r3at4_9Done) →(r3order'=3) &(r3time'=r3time+r3at3_10Time+travelr3at3_10);
 [r3at4_12] r3order=3 &(r3time+r3at4_12Time+travelr3at4_12<=TT) →(r3order'=4) & . . .
      (r3time'=r3time+r3at4_12Time+travelr3at4_12);
 [r3at3_13] r3order=4 &(r3time+r3at3_13Time+travelr3at3_13<=TT) &(r3time+travelr3at3_13 >=r3time) & . . .
      (r3at4_12Done) →(r3order'=5) &(r3time'=r3time+r3at3_13Time+travelr3at3_13);
 [r3at2_14] r3order=5 &(r3time+r3at2_14Time+travelr3at2_14<=TT) &(r3time+travelr3at2_14 >=r3time) & . . .
      (r3at4_12Done) →(r3order'=6) &(r3time'=r3time+r3at2_14Time+travelr3at2_14);
 [r3idle] r3order!=6 &(r3time+1<=TT) &(r3idleTime+1<=maxIdler3) →(r3time'=r3time+1) & . . .
      (r3idleTime'=r3idleTime+1);
endmodule

rewards "idle"
 [r3idle] true: 1;
endrewards
```

Listing 6.1: MDP model for robot group 1.

MDP, robot group 2. Listing 6.2 shows the MDP model of the second robot group. The difference with the previously described MDP is that tasks $at1\_1$ and $at1\_2$ must start after 10 minutes. As both tasks are joint, robots $r4$ and $r5$ must synchronise in time and space to complete these tasks.

```
1  mdp
2  const int TT=120;//total time available
3  //r4 velocity:2.0
4  const int travelr4at1_1=4 ;// from location: l4 (robot initial loc) to location: room1 (at1_1) distance:8
5  const int travelr4at1_2=4 ;// from location: room1 (at1_1) to location: room6(at1_2) distance:8
6  //r5 velocity:2.0
7  const int travelr5at1_1=5 ;// from location: l5 (robot initial loc) to location: room1 (at1_1) distance:9
8  const int travelr5at1_2=4 ;// from location: room1 (at1_1) to location: room6(at1_2) distance:8
9  const int r4at1_1Time=4; const int r4at1_2Time=4;
10 const int r5at1_1Time=4; const int r5at1_2Time=4;
11 const int maxIdler4=20; const int maxIdler5=20;
12 formula done=(r4order=2 &r5order=2);
13
14 module r4 //robot r4
15  r4order:[0..2];
16  r4time:[0..120];
17  r4idleTime:[0..maxIdler4];
18  [at1_1] r4order=0 &(r4time+r4at1_1Time+travelr4at1_1<=TT) &...
        (r4time+travelr4at1_1 = r5time+travelr5at1_1) & (r4time+travelr4at1_1 >= 10) →(r4order'=1) &...
        (r4time'=r4time+r4at1_1Time+travelr4at1_1);
19  [at1_2] r4order=1 &(r4time+r4at1_2Time+travelr4at1_2<=TT) &(r4time+travelr4at1_2 =r5time+travelr5at1_2)& ...
        (r4time+travelr4at1_2 >=10) →(r4order'=2) &(r4time'=r4time+r4at1_2Time+travelr4at1_2);
20  [r4idle] r4order!=2 &(r4time+1<=TT) &(r4idleTime+1<=maxIdler4) →(r4time'=r4time+1) &...
        (r4idleTime'=r4idleTime+1);
21 endmodule
22
23 module r5 //robot r5
24  r5order:[0..2];
25  r5time:[0..120];
26  r5idleTime:[0..maxIdler5];
27  [at1_1] r5order=0 &(r5time+r5at1_1Time+travelr5at1_1<=TT) &&...
        (r5time+travelr5at1_1 = r4time+travelr4at1_1) & (r5time+travelr5at1_1 >= 10) →(r5order'=1) &...
        (r5time'=r5time+r5at1_1Time+travelr5at1_1);
28  [at1_2] r5order=1 &(r5time+r5at1_2Time+travelr5at1_2<=TT) &(r5time+travelr5at1_2 =r4time+travelr4at1_2)& ...
        (r5time+travelr5at1_2 >=10) →(r5order'=2) &(r5time'=r5time+r5at1_2Time+travelr5at1_2);
29  [r5idle] r5order!=2 &(r5time+1<=TT) &(r5idleTime+1<=maxIdler5) →(r5time'=r5time+1) &...
        (r5idleTime'=r5idleTime+1);
30 endmodule
31
32 rewards "idle"
33  [r4idle] true: 1;
34  [r5idle] true: 1;
35 endrewards
```

Listing 6.2: MDP model for robot group 2.

The synchronisation of $at1\_1$ is highlighted in yellow. Line 18 shows robot $r4$ completing task $at1\_1$, as described in the label. The guard checks if it is ready to start the task ($r4order = 0$) and if there is enough time ($r4time + r4at1_1Time + travelr4at1_1 <$TT). It also checks if $r4$ is synchronised with $r5$ in time after travelling to the location of this task ($r4time + travelr4at1\_1 = r5time + travelr5at1\_1$) and if at least 10 time units have passed ($r4time + travelr4at1\_1 >= 10$).

Similarly in line 27, robot $r5$ checks that $r4$ is ready to synchronise and that its clock $r5time$ is at least 10 time units after reaching $at1\_1$'s location. Synchronising at $at1\_2$'s location is also ensured by lines 19 and 28. The rest of the model follows the same structure as the previous MDP.

```
1  dtmc
2  const double p_travel_r4at1_1=1.0 ;// from location: l4 (robot initial loc) to location: room1 (at1_1)
3  const double p_travel_r4at1_2=1.0 ;// from location: room1 (at1_1) to location: room6(at1_2)
4  const double p_travel_r5at1_1=1.0 ;// from location: l5 (robot initial loc) to location: room1 (at1_1)
5  const double p_travel_r5at1_2=1.0 ;// from location: room1 (at1_1) to location: room6(at1_2)
6  const double p_r4at1_1=0.97 ;const double p_r4at1_2=0.97 ;
7  const double p_r5at1_1=0.98 ;const double p_r5at1_2=0.98 ;
8  formula done=(r4=12 \& r5=12);
9
10 module r4 //robot r4
11  r4:[0..12];
12  //travel to at1_1
13  [r4travel_at1_1] r4=0→p_travel_r4at1_1:(r4'=2) +1-p_travel_r4at1_1:(r4'=1);
14  //try at1_1, retry allowed 2 times
15  []r4=2 →0.97:(r4'=6) +1-0.97:(r4'=3);
16  []r4=3 →0.97:(r4'=6) +1-0.97:(r4'=4);
17  []r4=4 →0.97:(r4'=6) +1-0.97:(r4'=5); //fail task at r4=5
18  //travel to at1_2
19  [r4travel_at1_2] r4=6→p_travel_r4at1_2:(r4'=8) +1-p_travel_r4at1_2:(r4'=7);
20  //try at1_2, retry allowed 2 times
21  []r4=8 →0.97:(r4'=12) +1-0.97:(r4'=9);
22  []r4=9 →0.97:(r4'=12) +1-0.97:(r4'=10);
23  []r4=10 →0.97:(r4'=12) +1-0.97:(r4'=11); //fail task at r4=11
24 endmodule
25
26 module r5 //robot r5
27  r5:[0..12];
28  //travel to at1_1
29  [r5travel_at1_1] r5=0→p_travel_r5at1_1:(r5'=2) +1-p_travel_r5at1_1:(r5'=1);
30  //try at1_1, retry allowed 2 times
31  []r5=2 →0.98:(r5'=6) +1-0.98:(r5'=3);
32  []r5=3 →0.98:(r5'=6) +1-0.98:(r5'=4);
33  []r5=4 →0.98:(r5'=6) +1-0.98:(r5'=5); //fail task at r5=5
34  //travel to at1_2
35  [r5travel_at1_2] r5=6→p_travel_r5at1_2:(r5'=8) +1-p_travel_r5at1_2:(r5'=7);
36  //try at1_2, retry allowed 2 times
37  []r5=8 →0.98:(r5'=12) +1-0.98:(r5'=9);
38  []r5=9 →0.98:(r5'=12) +1-0.98:(r5'=10);
39  []r5=10 →0.98:(r5'=12) +1-0.98:(r5'=11); //fail task at r5=11
40 endmodule
```

Listing 6.3: DTMC model for robot group 2.

DTMC, robot group 2. Listing 6.3 shows the DTMC model of the second group of robots. Lines 2-7 show the probabilities associated with travelling between locations and completing tasks for robots $r4$ and $r5$. Line 13 shows the transition from $r4$ initial location to the location of task $at1\_1$, succeeding with probability $p\_travel\_r4at1\_1$. Lines 15-17 show the attempts to complete task $at1\_1$, succeeding with 0.97 probability. The rest of the model follows the same structure. Line 8 shows the *done* formula showing the states where robots must arrive to say that the mission has been completed.

The attribute values for the allocation and permutation $(a_5, \bar{\jmath})$ after combining the values for each group are: $minIdle = 11.0$, $maxSucc = 44.37$, $minTravel = 41.0$.

6.6 Discussion and Limitations

Scalability. It is known that probabilistic model checking (and model checking techniques in general) suffer from the state explosion problem [29]. For multi-robot systems, this is exacerbated due to the (expected) large number of tasks. As KANOA is intended to be compositional, the module in charge of the plan synthesis can be replaced, or enhanced with different alternatives depending on the size of the model. Another research direction is to extend KANOA with alternatives for the synthesis of plans beyond PMC—many of which were identified in Chapter 3.

Centralised approach. We considered a centralised approach for the scheduling of tasks where all the information of the system (i.e., tasks, robots and the initial state of the system) is known by a central task scheduler. In real-world applications, this represents a disadvantage over decentralised, distributed and hybrid approaches, for example, as the failure of the central (task scheduler) manager represents the failure of the whole system. KANOA can be extended to support these approaches, as well as possible adaptations of the system if disruptions are encountered at runtime. For example, if a robot cannot complete one of its tasks, it can try to rearrange its plan and come back to this task later. If this cannot be done (for example, due to this task being a joint task that requires more robots to meet and coordinate), it can communicate to its closest peers to re-allocate this task among them. If this fails, the last resource would be to contact the central manager to reorganise this task among all available robots. Solutions such as [175] propose some of these steps. However, a trade-off is made between the resilience of the system due to these types of failures, and the loss of near-optimal solutions.

Partially-observable environments. In KANOA, we assume total knowledge of the tasks and robots. MDP alongside Mixed-Observed Markov Decision Process (MOMDP) are used in [98] to model single robots' behaviour. Then, a second module resolves any conflict arising for the set of synthesised plans using dynamic programming. The selection of MDP and MOMDP is done on a robot by robot basis depending on the information available. Although we do not consider partially known environments, KANOA's workflow can be augmented for this type of environment if required. As mentioned before, KANOA's behavioural analysis relies on a series of specialised models. The model in charge of the synthesis of plans can be modified into a series of MDPs and MOMDPs as described in [98] if the application at hand requires it.

Discrete time. We discretised the time to be modelled as part of the Markov Decision Process. However, this approach may not be optimal and presents limitations that need to be addressed for improved modelling accuracy and efficiency. One limitation is that if the time scale is *large*, there is the possibility that robots have to idle for a *large* number of time units increasing the number of states in the model and causing a state explosion. For the current version of KANOA, we avoid using the Continuous Time Markov Chain (CTMC) formalism, also supported by PRISM, as these are generally considered to be more prone to state explosion compared to DTMCs.

Implementation. The implementation of the task scheduler also restricts the number of tasks that a robot can handle—details of the implementation are further provided in Chapter 7. For instance, considering the maximum value of the int type in Java, which is 2147483647, (used to store the permutation of tasks) and the factorial of 16 being just below this limit (16! equals 2092278989), it follows that the tasks assigned to a robot must be limited to 16. Hence, scalability is also constrained by the implementation and further work is required to assess (and implement) alternatives in order to improve this limitation.

6.7 Related Work

The systematic literature review in Chapter 3 presents several strategies used by the research community for the scheduling of task problems in robotic systems. These are categorised in Table 3.4 into optimisation based techniques [48, 95, 101, 105, 110, 115, 129, 131], learning based [22, 91, 109], search based [23, 90, 138], logic based [25, 27, 28, 100, 104], hybrid [86, 134] and *others* [96, 107]. In this section, we discuss some of the most similar research to our proposed task scheduling method.

Given the wide variety of sources of uncertainties in multi-robot systems, it is not possible to capture this diversity into a single formalism. Consequently, multiple solutions have been proposed by the research community to tackle a subset of these uncertainties. Table 3.5 shows the classes and sources of uncertainties tackled by state-of-the-art solutions. For example; [22–25] propose solutions for partially known environments; [86, 87, 98, 105, 137] model the uncertainty of accepting and distributing new tasks at runtime; and [22, 101, 113, 127] the discrepancy between the belief state and the actual state, for example, by modelling the robots' model-drifting. KANOA adds to the body of knowledge a solution for the modelling of two types of uncertainties, "the probabilistic behaviour of task failures and retries" as a function of the heterogeneity in the robots' capabilities and "the probability of failing while travelling between locations". Moreover, we devise a method to compositionally annex several *specialised models* (as needed), to reason about other types of uncertainties in parallel.

The synthesis of MDP adversaries has been widely explored as a strategy for the synthesis of robot controllers [8]. MDP and MOMDP are used in [98] for the synthesis of robot plans depending on the information available. Each model considers one single robot task, while conflicts between synthesised plans are solved in a second stage. Using the PRISM model checker, [176] reports the synthesis of strategies for multiple scenarios, such as searching for objects and recharging at energy stations.

Although we also use probabilistic model checking, there are several differences in our approach. First, we simplify the MDP models by synthesising strategies to minimise the idling time instead of searching for the best permutation/sequence of tasks. The exploration to find optimal permutations is done by the meta-heuristic search-based approach, increasing the scalability of our solution. Second, we devise several specialised models to reason about different aspects of the quality of the robot plans. This is particularly important in MRS applications where the diversity of functional and non-functional requirements requires the computation and optimisation of multiple parameters. Third, we devise a series of rules for the MDP model for several tasks and time constraints in a correct-by-construction fashion. Lastly, we devise KANOA as a compositional approach, where the specialised models can be augmented, replaced or removed depending on the application in hand. For example, another model can be added to reason about the energy consumption assuming the robots have to recharge every certain amount of time.

GAs have been successfully applied for the task allocation and scheduling problem [177, 178], especially for the travelling salesman problem and variants [175, 179]. In, [179], the authors propose three different GA to solve the colored traveling salesperson problem (CTSP). This problem is not the same as the one we are solving. To understand the difference with KANOA, the constraints handled by CTSP are:

- every robot starts from and returns to location 0;

- a robot cannot visit the locations of other robots;

- a robot cannot start from another robot's location;

- each location must be visited by a robot exactly once;

- as some locations are shared, robots can enter this location and exit is required if this is the case;

- disconnected paths (called subtours) from an initial robot's location around locations are not allowed;

subject to a mono-objective optimisation function minimising the total travelling cost. Another difference is that the allocation and scheduling are modelled together and solver within the GA loop using integer linear programming. In [175], an extended version of the CTSP problem adds precedence of task constraints (which we refer to as ordered tasks) and is solved in a similar fashion within the presented genetic mission planner framework. Nevertheless, they don't consider other task constraints as KANOA do (aka consecutive and joint tasks).

6.8 Summary

In this chapter, we presented the KANOA approach for the task scheduling problem in multi-robot systems. Through the synthesis of individual robot plans, taking into account a spectrum of functional and non-functional, qualitative, and quantitative requirements as specified in the KANOA DSL, we have addressed a significant challenge in MRS task scheduling.

We used probabilistic model checking to reason about the probabilistic behaviour of the system. The probabilistic behaviour captures the possibility of failing with a task and several possible retries, as well as the possibility of failing while travelling between locations. The advantage of an exhaustive search by using PMC represented a disadvantage in the scalability of the problem as the number of tasks and robots increased. We have tackled the challenge of the size of the solution space by employing a hybrid approach that combines evolutionary-guided search with PMC. This allowed us to explore the solution space while optimising key objectives such as mission completion probability, robot idle time, and travel costs. The use of PMC not only aids in evaluation but also facilitates the synthesis of robot plans.

Overall, the KANOA approach offers a comprehensive solution to the task scheduling problem in MRS, leveraging the combination of probabilistic model checking and genetic algorithms, in a hybrid optimisation approach. We have illustrated the applicability of KANOA through a hospital scenario. Future research may explore extensions supporting the adaptation of the system and the degradation of requirements to enhance the scalability and robustness of the KANOA approach in tackling complex task scheduling challenges in multi-robot systems. Our contribution expands the existing body of MRS problem variants and solutions documented in the research literature.

Chapter 7

Tool-supported Mission Allocation and Scheduling Methodology

This chapter presents the KANOA tool-supported framework for the synthesis of robot plans under a set of complex constraints and multiple conflicting optimisation objectives. The complete list of supported constraints is presented in Table 6.1. Following the taxonomy for task allocation problems, KANOA's tool can solve single-task, multi-robot time-extended assignment (ST-MR-TA) [39] with cross-schedule dependencies (XD) [51] task allocation problems.

This chapter is organised as follows. The KANOA methodology is introduced in Section 7.1. Section 7.2 presents the evaluation assessing the performance and feasibility of the KANOA methodology and tool. Section 7.3 presents discussion and limitations. Finally, Sections 7.4 and 7.5 describe related work and a summary of the chapter, respectively.

7.1 End-to-end Methodology

The KANOA end-to-end framework is depicted in Figure 7.1. The workflow starts with the problem specification and KANOA's hyperparameters (1). The problem specification—designed in a separation-of-concerns fashion—allows clients, domain experts, and other stakeholders to collaborate on different parts of the problem: world model, task, robots' description, and mission specification. For example, MRS experts may focus on the description of the robots, their capabilities and probabilistic behaviour, while clients and other stakeholders define the task and mission for their specific scenario. KANOA's DSL for the specification of the problem specification is described in Section 4.2. KANOA's *hyperparameters* are associated with different stages of the framework. For example, for the optimisation process (5), the MRS experts can define the number of iterations and population size for the multi-objective optimisation algorithm.

The pre-allocation component (2) processes the problem specification and stores it into KANOA's database. As the mission allows the repetition of tasks at different locations, this step also instantiates any task to be allocated to robots.[1] It also performs a *model-to-model transformation* to generate the task allocation model. The task allocator (3) generates multiple allocation models, consisting of feasible allocations of the tasks into a

[1]See section 6.2.

set or subset of available robots.

Continuing, the pre-scheduler component (4) generates groups of robots that share task constraints, and transfers any task constraint from compound tasks to atomic tasks. In stage (5), the task scheduling stage integrates three components: an elitist multi-objective optimisation genetic algorithm (GA) component, a task scheduler component, and n quality of service (QoS) analysers. The GA component explores the solution space, which consists of task allocations and their task permutations. The task scheduler creates feasible task schedules based on these allocation-permutation combinations. Each QoS analyser quantifies one of the optimisation objectives outlined in the problem specification. Internally, during the GA evaluation stage, the GA component employs the task scheduler and the QoS analysers to guide the exploration of the solution space. Finally, the plan selector chooses a task schedule, containing individual robot plans through a ranking system.

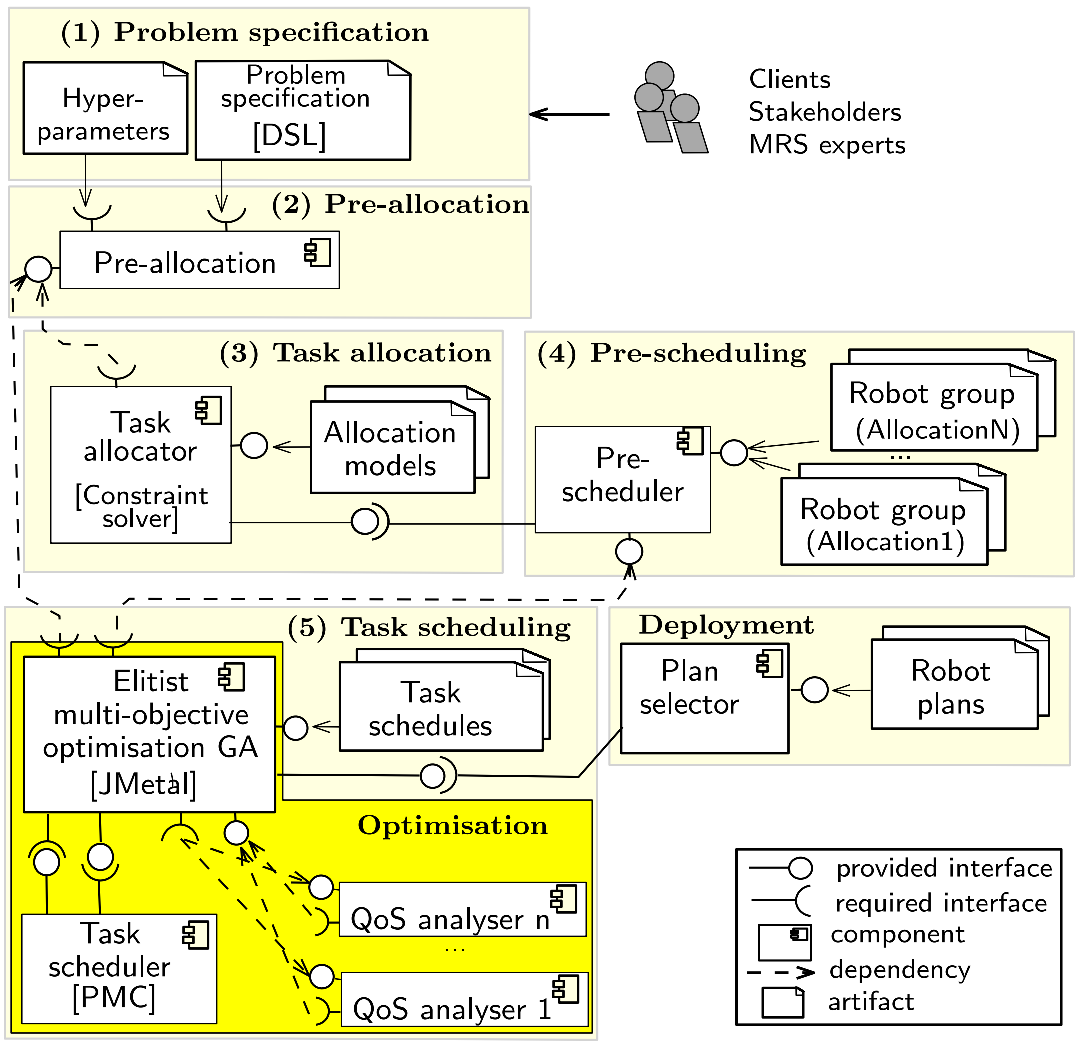
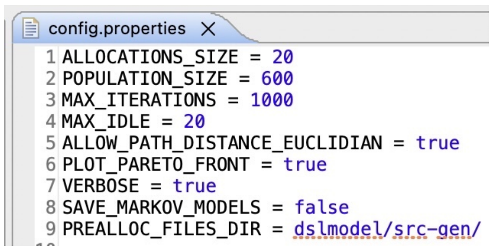


Figure 7.1: KANOA tool workflow.



```
1 ALLOCATIONS_SIZE = 20
2 POPULATION_SIZE = 600
3 MAX_ITERATIONS = 1000
4 MAX_IDLE = 20
5 ALLOW_PATH_DISTANCE_EUCLIDIAN = true
6 PLOT_PARETO_FRONT = true
7 VERBOSE = true
8 SAVE_MARKOV_MODELS = false
9 PREALLOC_FILES_DIR = dslmodel/src-gen/
```

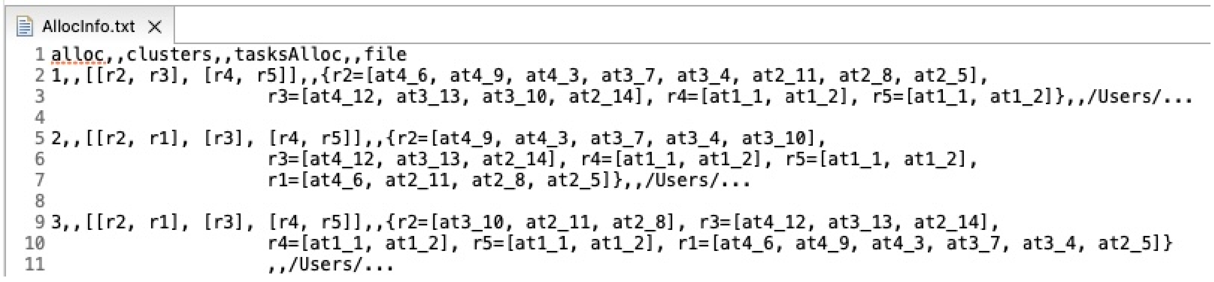
Figure 7.2: Config.prop file containing KANOA hyperparameters.

Implementation. We automated the KANOA approach by implementing a tool for the generation of feasible robot plans. KANOA's domain specific language in (1), Figure 7.1, was implemented using the Eclipse EMF supported Xtext framework [147]. We used Ecore and Xtend [180] for the model-to-model transformation from the DSL into the Alloy model. The remaining components of the toolchain are developed in Java (2-5). For the task allocation, we integrated the Alloy Analyzer [55] as the task allocator component (3). We implemented the NSGA-II multi-objective optimisation GA algorithm in JMetal [181] in task scheduling (5). The PRISM probabilistic model checker [71] was used as the task scheduler and for the QoS analyser components, for the synthesis of robot plans and their analysis, respectively (5). The KANOA source code is available from our GitHub repository at https://github.com/Gricel-lee/MRS-ThesisMaterial.

KANOA framework starts with two input specification files. An example of the "problem specification file" is previously shown in Figure 4.7, whereas an example of the hyperparameters' file is illustrated in Figure 7.2. For the latter, the first line states the number of allocations to obtain and lines 2-3 refer to the population size, as well as the number of iterations of the genetic algorithm. Line 4 is optional and defines an upper bound for the time that robots are allowed to spend idling. Line 5 is optional and allows the generation of all paths. These paths are computed as the Euclidean distance between any two locations declared in the problem specification.[2] Continuing, lines 6-7 enable the visualisation of the Pareto front of solutions and provide feedback printed into the console, respectively. Line 8 allows saving the Markov model files created by the task scheduler. Lastly, line 9 is the relative path to any generated file from the problem specification by KANOA.

If the VERBOSE hyperparameter is set to true in line 7, KANOA also generates an **Allocinfo.txt** file. An example of this file is shown in Figure 7.3. The first line is the header with each column's name separated by ",,": number of allocation (alloc), robot groups (clusters), allocation of tasks to robots (tasksAlloc) and the allocation file generated by the Alloy Analyzer (file).

[2]In a scenario with n locations, where each location has a path to every other location, there would be $(n-1)!$ paths between any two distinct locations. This parameter aims to facilitate testing the tool without the tedious task of explicitly declaring each of these paths. Internally, KANOA computes the Euclidean distance rather than generating all paths as part of the problem specification's file.



```
 1 alloc,,clusters,,tasksAlloc,,file
 2 1,,[[r2, r3], [r4, r5]],,{r2=[at4_6, at4_9, at4_3, at3_7, at3_4, at2_11, at2_8, at2_5],
 3                     r3=[at4_12, at3_13, at3_10, at2_14], r4=[at1_1, at1_2], r5=[at1_1, at1_2]},,/Users/...
 4
 5 2,,[[r2, r1], [r3], [r4, r5]],,{r2=[at4_9, at4_3, at3_7, at3_4, at3_10],
 6                     r3=[at4_12, at3_13, at2_14], r4=[at1_1, at1_2], r5=[at1_1, at1_2],
 7                     r1=[at4_6, at2_11, at2_8, at2_5]},,/Users/...
 8
 9 3,,[[r2, r1], [r3], [r4, r5]],,{r2=[at3_10, at2_11, at2_8], r3=[at4_12, at3_13, at2_14],
10                     r4=[at1_1, at1_2], r5=[at1_1, at1_2], r1=[at4_6, at4_9, at4_3, at3_7, at3_4, at2_5]}
11                     ,,/Users/...
```

Figure 7.3: File with task allocations' information.

Lines 2-3 show the first allocation. Starting with the allocation identifier, 1; followed by the groups of robots $[r2, r3]$ and $[r4, r5]$; the tasks allocated to each robot show as a dictionary; and the Alloy's model global path. Similarly, allocations 2 and 3 are depicted in lines 5-7 and 9-11, respectively.

Context. In Section 1.2 we defined the research objectives of this thesis. These relate to (1) the specification of task allocation and scheduling problems for MRS, (2) the development of a method for solving these two relevant MRS problems, (3) the integration of the proposed methods in an end-to-end methodology, and (4) provide tool support for their adoption and future development.

In Chapter 4, we presented and evaluated a catalogue for the specification of quantitative robotic patterns, and the KANOA DSL for the problem specification of task allocation and scheduling problems; both relevant to research objective (1). Chapter 5 and Chapter 6, we presented a methods for solving the task allocation and scheduling problems captured by the KANOA DSL; relevant to research objective (2). This section presented the integration of such methods and their tool implementation, aligned to research objectives (3) and (4), respectively. To assess the effectiveness and feasibility of these contributions, the following section presents a systematic evaluation of the proposed KANOA framework. Additionally, we examine its scalability through two case studies and analyse the impact of various hyperparameters.

7.2 Evaluation

We carried out extensive experiments to evaluate KANOA's effectiveness, scalability and configurability.

This section is divided as follows. Section 7.2.1 introduces the research questions we explore. Section 7.2.2 describes the case studies. Section 7.2.3 shows the experimental setup. Section 7.2.4 contains the experimental tasks, results and discussion.

7.2.1 Research Questions

Our evaluation aims to answer the following research questions,

RQ1 (Effectiveness). *Do plans synthesised by KANOA achieve better trade-offs among the optimisation objectives than those generated through random search?* We use established metrics for evaluating the quality of the Pareto fronts of solutions generated both by KANOA and random search (a common baseline in the metaheuristic search-based community [50, 182, 183]) within a limited time frame, assessing how they compare with respect to the quality of service attributes associated with the different optimisation objectives supported by KANOA.

RQ2 (Scalability). *How does the computational overhead for KANOA's solution synthesis grow with increasing mission size, number of allocations, and number of pre-allocated tasks?* We performed experiments to assess how the execution time of KANOA increases for MRS missions with a growing number of tasks and task constraints, under different numbers of allocations, and with different numbers of pre-allocated tasks.

RQ3 (Configurability) *How do KANOA hyperparameters affect the quality of the solutions?* We assess the impact of two hyperparameters associated with the number of task allocations and the number of iterations. We assess the trade-off between increasing "optimality" (in terms of attaining better values for the quality attributes of the optimisation objectives) at the expense of increasing the execution time when the number of allocations increases.

For RQ1, we use random search as our baseline. KANOA's unique combination of features makes it challenging to compare with existing approaches. These differentiators include probabilistic modelling of potential failures during robot travel, a finite number of task retries, and the optimisation of multiple objectives (see Table 6.1).[3]

7.2.2 Case Studies

To answer the questions posed in the previous section, we extensively evaluate KANOA on two multi-robot case studies. We selected two distinct case studies from the domain of mobile robots following standard practice in empirical software engineering [184, 185]. Their key differences are summarised in Table 7.1.

Hospital case study. The first case study consists of the hospital scenario outlined in Section 4.3. This scenario is further discussed in Sections 5.4 and 6.5, to illustrate the task allocation and scheduling solutions, respectively.

[3]Comparing KANOA to other algorithms, either by extending these approaches to meet KANOA's requirements or by simplifying KANOA's case studies, is left for future work.

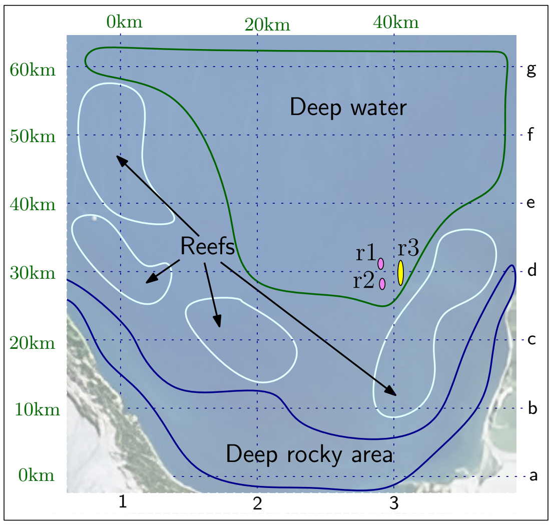


Figure 7.4: Bo-alpha case study deployment area. The mission consists of measuring temperature and salinity levels in every location x∈{1,2,3} y∈{a, b, c, d, e, f, g}, except for (a,1) and (b,1). UUVs $r1 - r3$ start at location (3,d).

Bo-alpha case study. The second case study is based on a variant of the Bo-alpha mission deploying unmanned underwater vehicles (UUVs) from [186, 187]. The mission involves sampling a designated ocean area to assist oceanographers in acquiring data on water salinity (AT1) and temperature (AT2). These tasks are defined together as a compound task CT1 with a consecutive constraint so that both measurements are taken together.

Within the deployment area, three UUVs ($r1 - r3$) must collect two samples from 19 points shown as the intersection of dotted lines in Figure 7.4. Each location is identified with the letter l, followed by a number and a letter as shown in the axes of this figure. The UUVs start at location $l3d$. It is known that certain areas pose greater challenges for their access due to underwater obstacles such as coral reefs, kelp forests and rocky zones. Two of the robots, $r1$ and $r2$, are notably smaller in size compared to $r3$, thus enhancing their likelihood of navigating and completing tasks in these challenging areas. Each sampling location belongs to one of three areas: deep waters, deep rocky areas or reef zones. Robot $r3$ cannot enter the reef zone to avoid damaging the reef and itself, and cannot take any task too close to the coastline set at x<5km. Robots can retry tasks in deep water up to three times, but not in the other two zones as it is considered high-risk. Furthermore, paths within the reef zone have an 85% probability of the UUVs transitioning successfully and all tasks in the reef zone have a lower probability of completion.

To ensure that the robots have enough battery to return home, we are interested in minimising the travelling cost and recalling all robots after 600

Table 7.1: Key characteristics of the two case studies. *See Table 6.1 for constraints and optimisation objective full descriptions.

	Hospital case study	**Bo-alpha case study**
Domain	Medical and household	Exploration and research
Number of robots	5	3
Number of atomic tasks in the mission	14	38
Constraints types*+	C1 (Constrains operational area), C2 (Allocates a task to a robot), C5 (Joint tasks), C6 (Ordered tasks), C8 (Time to start or finalise a task), C9 (Minimum rate of mission success), C10 (Maximum completion time).	C1, C4 (Limit number of tasks in a robot), C9, C10
Optimisation objectives*	O1 (Maximise the probability of success), O2 (Minimise the robots' idling time), O3 (Minimise the travelling cost).	O2, O3

+C7 (forces subtasks of a compound task to be performed consecutively) is added as part of the research question RQ2. Constraints C3 (allocates a task to the closest robot) is similar to C2.

minutes. Finally, to help with a fair partition of the tasks, we limit each task allocation to a maximum of 16 tasks per robot. The description of the robots, initial locations and their probabilities of task completion is defined in Table 8.2. The problem specification is shown in Appendix G Figure 8.8.

7.2.3 Experimental Setup.

Experimental setup. Results were obtained using a NUC Extreme i9-12900, 64 GB RAM, 3200 MHz, Ubuntu 22.04; KANOA was run within the Eclipse IDE version 2022-09. We used the multi-objective optimisation algorithm NSGA-II implemented in the JMetal library [181]. We followed the suggested default parameters from the JMetal library (summarised in [188]) as:

Selection of parents: binary tournament; *mutation probability* = 1/(number of attributes); *mutation*: polynomial; *crossover probability* = 0.9; *crossover*: simulated binary crossover (SBX).

These default values are also considered in EvoChecker [50], which served as an inspiration for the KANOA's task scheduler. We used default settings for the Alloy Analyzer[4] and the PRISM model checker.

[4]By default, Alloy Analyzer selects the SAT4J SAT solver to generate solutions. For faster performance on small problems, MiniSat and ZChaff solvers are available (for some OS) [57]. For larger problems, Berkim solver is documented to perform the best [59]. We kept the default configuration as the time to return the solutions for our case studies is in the scale of seconds. This can be configured for other case studies if needed.

Table 7.2: Pareto front quality indicators [188]. Intuitively, the IGD indicates how close a set of solutions is to the overall best set of solutions (the Pareto front reference), while the HV indicates how much volume has been covered, indirectly getting closer to the Pareto front reference.

Acronym	**Name**	**Brief description**	**Requires**
IGD	Inverted generational distance	Euclidean distance between solutions in PF_{ref} and the nearest solutions in PF_c. Smaller is better.	Reference Pareto front (PF_{ref})
HV	Hypervolume	The volume covered by solutions in PF_c with respect to the nadir point. Bigger is better.	Nadir point

To account for random variations from the intrinsic probabilistic behaviour of the search algorithms used (KANOA and random) we ran all experiments five times. The following section answers our research questions **RQ1-3** through these case studies. Each answer is structured in three parts: 1) experimental setup, 2 experimental tasks and 3) results and discussion.

7.2.4 Experimental Tasks, Results and Discussion.

RQ1 (Effectiveness)

Do plans synthesised by KANOA achieve better trade-offs among the optimisation objectives than those generated through random search?

RQ1 experimental setup. We compared KANOA against randomly generated task schedules for both case studies. To this end, we:

1. Used the first part of KANOA (cf. Figure 7.1 stages 1-3) to generate 5,000 task allocations available to be sampled randomly. This covers the total of 1,728 feasible allocations found for the hospital case study, and a large number for the Bo-Alpha case study (compared to KANOA). By using the allocation of KANOA, we ensured that tasks were allocated only to robots that are equipped to perform them.[5]

2. Randomly generated task schedules in two steps. First, randomly selecting a task allocation and second, randomly selecting a possible task permutation from this allocation.

We ran KANOA and random search for 400 s (∼6.5 minutes) assuming that feasible

[5]In [189], their random search algorithm can allocate tasks to robots that cannot perform them. Then, a *no − capability* failure detects this inconvenience. In our case, we only consider feasible allocations, resulting in a better comparison of KANOA against a random search baseline.

Table 7.3: Reference Pareto front of feasible solutions found for the Hospital case study.

	Idling time minIdle (min)	Success rate maxSucc (%)	Travel cost minTravel (m)	Synthesised by
1	18.0	40.040	23.0	Random search
2	11.0	60.874	34.0	Kanoa
3	11.0	67.638	36.0	Kanoa
4	18.0	60.874	29.0	Kanoa
5	18.0	69.203	55.0	Kanoa

plans must be transmitted to the robots within a limited timeframe before deployment[6], taking snapshots every 20 seconds. Then, we compared the performance between the two alternatives at every time instant in which the snapshots were taken. We set the number of iterations to its minimum value ($n_i = 2$)—KANOA searches for feasible robot plans one allocation at a time; this low number of allocations allows KANOA to sort the best task schedules initially found while also searching through as many allocations as possible within the time limit.

RQ1 experimental tasks. For each case study, the experimental tasks (**e1**) for RQ1 consisted of saving the set of feasible solutions found at every 20 s time step. We collected the data over five runs using KANOA and our baseline.

To quantitatively assess the quality of the solutions found at every time step, we use two quality indicators, the inverted generational distance (IGD) and the hypervolume (HV) described in Table 7.2. The IGD requires a reference Pareto front, preferably the optimal set of Pareto front solutions. However, in many cases, including ours, this is unknown. In these cases, a reference Pareto front is obtained from the set of all solutions available [190]. For HV, we are required to obtain the *nadir* point defined as the worst possible value for each optimisation objective [188, 191]. Hence, we generated the reference Pareto front (PF_{ref}) and the nadir reference point from the joint set of all solutions found. Then, we generated the Pareto front of solutions at every time step and computed the Pareto quality indicators as described in Table 7.2.

RQ1 results and discussion. (**e1**). For the **hospital** case study, Table 7.3 shows the reference Pareto front consisting of five solutions (see also Appendix H) Columns 2-4 show the values of the three attributes (idling time, success rate, and travel cost); the last column indicates that four out of five solutions were synthesised by KANOA. The nadir (worst) point was found at $minIdle = 51.0, maxSucc = 20.385, minTravel = 87.0$.[7]

Figure 7.5 shows the quality indicator (IGD and HV) box plots at every time increment, and the total number of feasible solutions found. Two boxes at every time step

[6]We focus on design time because verifying robot plans with our approach is time-consuming. In critical applications, a delay of minutes might not be unacceptable at runtime.

[7]d

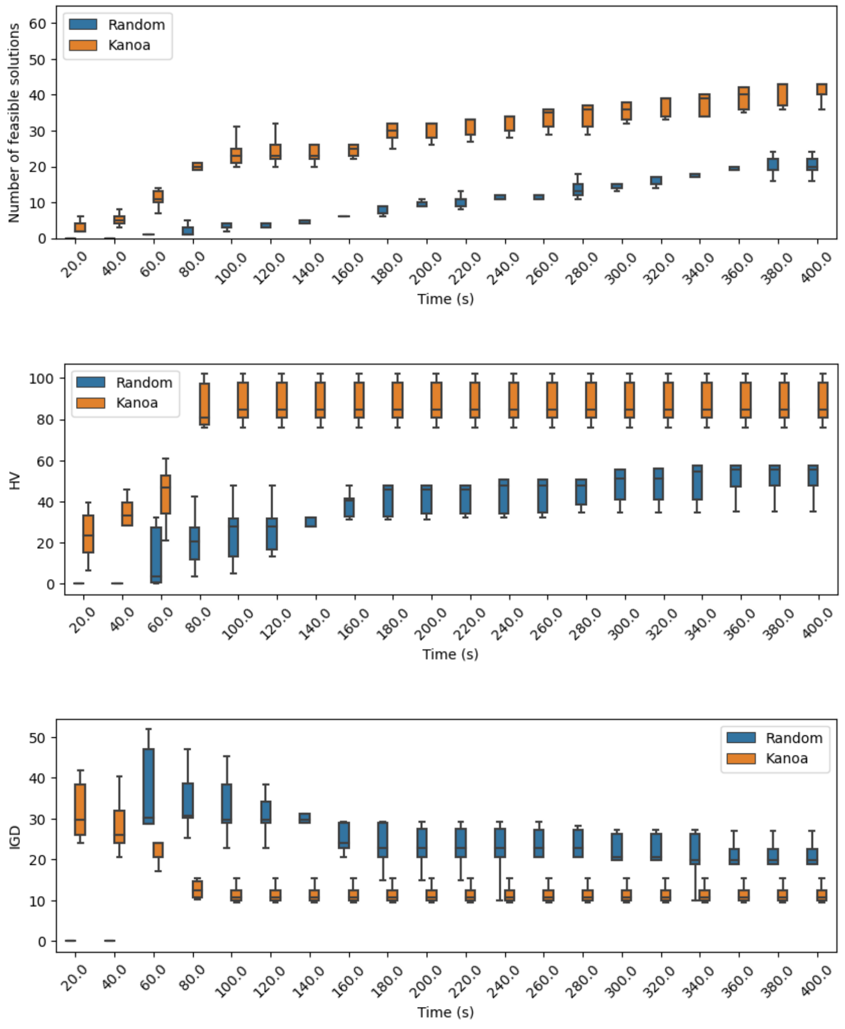


Figure 7.5: Hospital case study: number of solutions found by KANOA vs random generation (top), and quality indicator values (HV: middle, IGD: bottom) of computed Pareto fronts at every time step.

show the results for the KANOA and randomly generated solutions, respectively. For the hospital case study, KANOA's task scheduler systematically checked up to 34 allocations for solutions. Random search sampled from all the 1,728 allocations found. The top box plot shows the number of solutions found by KANOA, with a median value of 44 solutions by the end of the 400 s, compared to 21 found through random search. Both search algorithms show a decrease in the values of IGD over time and a gain in the HV, indicating better solutions. However, KANOA converges more quickly and produces higher-quality solutions in the Pareto optimal set, for example, after 80 s the IGD median reaches a steady value of 12 and the HV's a value of 85,000 approximately.

In contrast, the randomly generated solutions seem to continue varying in a monotonic trend. An interpretation of this trend in comparison with KANOA is that, as the random algorithm has more information available on which to sample from, it is likely that it

Table 7.4: Reference Pareto front of feasible solutions found for the Bo-alpha case study.

	Idling time minIdle (min)	Success rate maxSucc (%)	Travel cost minTravel (m)	Synthesised by
1	18.0	63.136	515.0	Kanoa
2	18.0	72.409	576.0	Kanoa

avoids staying in local optimal solutions. However, this may take longer as information from the already found solutions is not exploited to generate similar optimal solutions instead of exploring the search space.

For the **Bo-alpha** case study, Table 7.4 shows the two points comprising the bi-objective reference Pareto front.[8] The nadir is set to $maxSucc = 50.746, minTravel = 879.0$. Through experimentation, we realised that for this case study, the time to generate solutions through the random algorithm is considerably larger than using KANOA as the task allocator must first generate a large number (5000) of allocations from which to sample randomly. To make a fair comparison, we start the timer after the task allocation is completed.

Figure 7.6 shows the quality indicator (IGD and HV) box plots at every time increment, and the number of feasible solutions found over time. Two boxes at every time step show the results for the KANOA and randomly generated solutions, respectively. KANOA' experiments employ 36 allocations, while random search sampled from 5000 allocations. The top box plot shows the number of solutions found by KANOA, with a final median value of 97, compared to 47 by random search. Both search algorithms show a decrease in the values of IGD over time and a gain in the HV, indicating that the Pareto front contains progressively better solutions as time increases. Both quality metrics show better (median) values for KANOA compared to the randomly generated Pareto front of solutions. Note that there is a time interval between 140 s and 240 s during which KANOA cannot find any solutions. This behaviour is given because KANOA systematically selects an allocation, and then iterates through its possible task permutations before moving to the next allocation. Hence, it is possible that, for some allocations, KANOA might not find a solution to optimise due to the constraints of the problem at hand. In contrast, the random algorithm follows a linear trend when finding feasible solutions. A solution to optimise KANOA as it iterates through allocation is to stop the optimisation process of an allocation if it cannot find any feasible solutions after a certain number of evaluations.

[8]These were obtained from the join set of solutions found by RQ1 and RQ3.

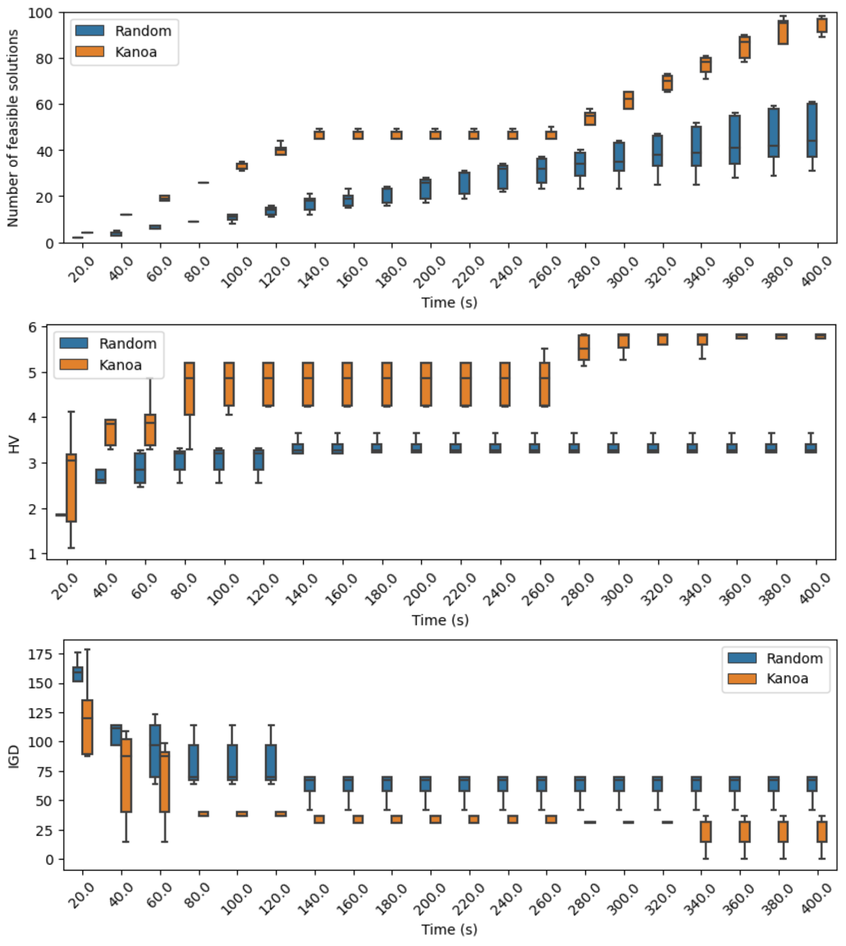


Figure 7.6: Bo-alpha case study: number of solutions found by KANOA vs random generation (top), and quality indicator values (HV: middle, IGD: bottom) of computed Pareto fronts at every time step.

RQ1 findings summary. The best robot plan solutions (i.e., from the joint Pareto-optimal set) were found by the KANOA framework, with the exception of one solution in the hospital case study. Additionally, for both case studies, KANOA was able to generate a greater number and better task plan solutions over time compared to our baseline.

RQ2 (Scalability)

How does the computational overhead for KANOA's solution synthesis grow with increasing mission size, number of allocations, and number of pre-allocated tasks?

RQ2 experimental setup. We systematically increased the number of tasks, task constraints, and robots required for a single *joint* atomic task to find the limit to which these scale in both case studies. We chose to conduct the evaluation for this research question with a single joint task, given that increasing the number of tasks without dependencies among them will only result in multiple clusters of tasks that can be processed independently, and where computation time will be dominated by the largest cluster.

We also assessed the impact of increasing the number of allocations. As the time needed by the task allocator to generate n_a allocations is *much shorter* than the time needed by the task scheduler to go over the same number of allocations, we assessed the time taken by the task allocator and scheduler separately. For the task allocator, we set 1,000 as the maximum number of allocations to generate—this should provide a reasonable estimate of the expected computational time required to generate up to n-allocations. Since the task scheduler requires more time to complete (due to the formal verification of the robot plans' stage), we limited it to iterating over 100 of these allocations—assuming this should be a reasonable estimate of the expected computational time required to iterate over up to n-allocations. Finally, we set the number of iterations to n_i =100.[9]

Finally, we assessed the overhead of adding these pre-allocated tasks, as the pre-allocation of tasks to specific robots (required by several types of constraints supported by KANOA) increases the size of the Alloy model. To assess the pre-allocation overhead, we extended the hospital case study with an additional robot, $r6$, with 1, 25, and 50 pre-allocated tasks.

RQ2 experimental tasks. Three different experiments were conducted to answer RQ2. For each case study, we:

e2.1. Created variants of the case studies by systematically increasing the number of robots n_r, tasks n_t, task constraints, and the number of robots required for a joint task, observing and reporting the problem instance sizes for which KANOA can no longer generate solutions.

e2.2. Varied the number of allocations from 10 to 1000 in increments of 20 and obtained the time required by the task allocator to generate these allocations. We also executed the task scheduler to obtain plans for up to 100 allocations and obtained the time needed to iterate through them. In both setups, we report the computation time required.

e2.3 Extended the hospital case study with an additional robot, $r6$, with 1, 25 and 50 pre-allocated tasks. For each variant, we report the average execution time required to generate 10 allocations.

RQ2 results and discussion. (e2.1). Evaluating scalability in terms of tasks, robots, and task constraints proved to be a challenging task due to the intricacy of these and their impact on the ability to find solutions. In this section, we summarise the findings for both case studies after running the variants shown in Table 7.5. The table shows the original case studies (yellow-shaded rows); variants for which task schedules were

[9]We bound it to 100 as completing a single run over 100 allocations for the Bo-alpha case study takes approximately 20 hours. We collected 5 runs for each case study.

successfully synthesised; and variants for which no solution was found representing the limit to which varying the number of tasks, task constraints and robots scale (grey-shaded rows).

The findings of the **hospital** case study are summarised as follows:

- A maximum of 7 ordered tasks are manageable. Exceeding this limit results in the inability to generate schedules that adhere to both mission and task constraints.
- The addition of consecutive tasks is not feasible, as there is insufficient idle time available to coordinate the robots designated for its tasks.
- A joint task can accommodate up to ten robots. Exceeding this number leads to memory overflows in the PRISM model checker.
- Up to 5 joint tasks can be integrated successfully. Beyond this, the task scheduler cannot find a schedule that meets the task constraints.

Joint tasks require the synchronisation of robots in space and time. Hence, as more joint tasks were added, finding permutations that allow robots to meet in space and time became more difficult.

A similar scalability problem takes place when ordering and consecutive constraints are added. For this case scenario, adding a single consecutive task resulted in no plans being synthesised. Consecutive constraints are more restrictive than ordering constraints in the sense that require the previous subtask to be completed *at same time instant* in which the next one starts. Hence, it is more difficult (compared to ordering constraints) to obtain plans, as consecutive constraints require (a) a valid permutation of the tasks found by the task scheduler, and (b) enough idling time for robots to wait until their peers complete the previous tasks.

Scaling the number of robots when task constraints exist may result in the inclusion of all robots with similar capabilities within the same probabilistic model. In this case study, the increase in the number of (cleaner) robots resulted in a state explosion of the MDP models generated within the optimisation loop.

The findings of the **Bo-alpha** case study are summarised as follows[10]:

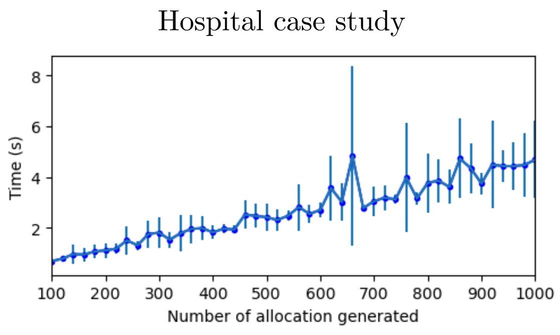
- A maximum of 9 ordered tasks are manageable. Exceeding this limit results in no solutions due to limited idling time.
- A maximum of 5 consecutive tasks are manageable. Exceeding this limit results in no solutions due to limited idling time.

[10]To test the scalability of the task constraints, we had to lower the number of tasks and systematically increase them while adding task constraints.

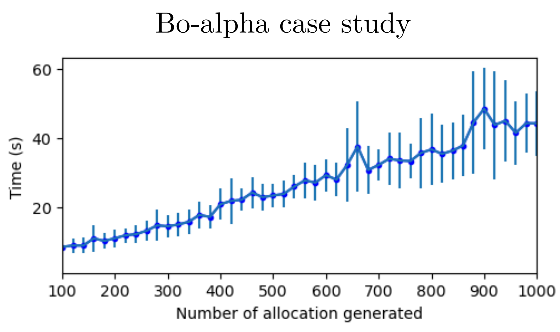
Table 7.5: Variants of the case studies varying robots, tasks and task constraints. The name of the variant contains information about the case study (H for hospital, B for Bo-Alpha); followed by "-r", the number of robots (n_r), "t" and the number of atomic tasks (n_t). Next, after the second hyphen, the maximum number of robots needed for a joint task, consecutive tasks and ordered tasks are displayed after the "j", "c" and "o", respectively.

Variant identifier	**Description of variant**	**Plans synthe-sised**	**Feedback on failure to get solutions**
H-r5t14-j2c0o4	Hospital original case study^{+}	✓	-
H-r5t17-j2c0o5	increase to 5 ordered tasks	✓	-
H-r5t23-j2c0o7	increase to 7 ordered tasks	✓	-
H-r5t29-j2c0o9	increase to 9 ordered tasks	×	No schedule found complying with tasks constraints.
H-r5t14-j2c1o5	increase to 1 consecutive tasks	×	Not enough idling time available.
H-r6t14-j1c0o4	1 joint task requiring 3 robots	✓	-
H-r8t14-j1c0o4	1 joint task requiring 5 robots	✓	-
H-r13t14-j1c0o4	1 joint task requiring 10 robots	✓	-
H-r14t14-j1c0o4	1 joint task requiring 11 robots	×	PRISM runs out of memory
H-r5t15-j3c0o4	increase to 3 joint tasks	✓	-
H-r5t17-j5c0o4	increase to 5 joint tasks	✓	-
H-r5t19-j7c0o4	increase to 7 joint task	×	No plan found complying with tasks constraints.
B-r3t38-j0c0o0	Bo-alpha original case study	✓	-
B-r3t2-j0c1o0	1 consecutive task*	✓	-
B-r3t6-j0c3o0	3 consecutive tasks*	✓	-
B-r3t10-j0c5o0	5 consecutive tasks*	×	Not enough idling time available.
B-r3t10-j0c0o5	5 ordered tasks*	✓	-
B-r3t14-j0c0o7	7 ordered tasks*	✓	-
B-r3t18-j0c0o9	9 ordered tasks*	✓	-
B-r3t22-j0c0o11	11 ordered tasks*	×	Not enough idling time available.
B-r3t2-j2c0o0	2 joint tasks, 3 robots per task	✓	-
B-r5t2-j2c0o0	2 joint tasks, 5 robots per task	×	Not enough idling time available.

$^{+}$Four ordered tasks of three atomic tasks each, and two join tasks for a total of 14 atomic tasks.
*Each ordered or consecutive task consists of two atomic tasks (check salinity level and temperature).
**Adding cleaner robots.



(a)



(b)

Figure 7.7: Time required to obtain up to 1000 allocations.

- A joint task can accommodate up to five robots. Exceeding this limit results in no solutions due to limited idling time.[11]

Although this is not an exhaustive evaluation of all possible combinations of robots, tasks and task constraints, we can conclude that scalability is an issue as more constraints of tasks appear. We also identified key characteristics to improve the scalability of KANOA: i) only generate permutations that comply with the order of tasks dictated by the task constraints (see the following observation box), ii) increase the idling limit automatically by degrading this constraint (when the bound in the idling time is identified as a problem to generate solutions), iii) limit the number of robots and tasks in an MDP model to avoid running out of memory (caused by the state-explosion problem). These solutions are not straightforward to implement, and additional efforts are necessary to address these challenges effectively.

Observation 9. The current version of KANOA generates permutations based on the number of tasks allocated to each robot. Creating an algorithm to discard the permutations that are not feasible due to task constraints could increase the scalability in terms of the number of constraints in tasks.

(e2.2) Figure 7.7 shows the mean time needed by the task allocator to generate up to 1000 allocations for the (a) hospital and the (b) Bo-alpha case studies. Both case studies show a linear trend in their execution time as more allocations are generated. For the **hospital** case study, the mean time to generate 1000 allocations is 4.68 s. For the **Bo-alpha** case study, this time goes up to 44.12 s. Although the Bo-alpha case study does not have any task constraints, it has more tasks and task locations than the hospital case study. The task allocator, implemented through the Alloy Analyzer, generates atoms for all the locations and their coordinates. This accounts for the increase in time shown

[11]For this variant, two joint tasks were added, each requiring five robots.

when comparing these case studies results.

The variance in the time required to obtain allocations grows with the number of allocations. Visually inspecting Figure 7.7, there is a smaller variance between 100 and 300 allocations generated, compared to the range between 800 and 1000. Although this shows that the execution time deviates from the expected time as more allocations are needed, it also shows that, for both case studies, less than 1000 allocations can be computed in less than a minute.

Figure 7.8 shows the mean time needed by the task scheduler to go through up to 100 allocations for the (a) hospital and the (b) Bo-alpha case studies. For the **hospital** case study, several spikes can be observed, for example, between 20 and 40, and around 80.[12] These spikes appear when allocations take longer than usual due to two possible reasons: i) the probabilistic models are more complex[13] or ii) more solutions were found[14]—increasing the time considerably (and hence creating a spike that can be observed in the plot).

For the **Bo-alpha** case study, the mean time shows a linear trend with no spikes compared to the previous case study. As tasks in the Bo-alpha case study are not constrained, robots are modelled separately in every task allocation. Hence, the *spike* pattern is not visible in Figure 7.8b. The graph also shows that, for this case study, no feasible solutions were found for task allocations 2, 4-7 and 9-15. This information can be used in future versions of KANOA to avoid spending time on allocations where it is *difficult* (or impossible due to the problem constraints) to synthesise solutions setting a limit on the number of failed evaluations.

Finally, notice that the average time to complete 100 allocations is 1.44 e4 s (4 hr approximately) for the case study and more than 1.4 e5 s (38.8 hr approximately) for Bo-alpha.[15]. This time scale must be considered by the domain experts and KANOA's users when defining the hyperparameters of the problem.

(e2.3) Figure 7.9 shows the execution time to allocate 1, 25 or 50 pre-allocated tasks to robot *r6* in a variant of the hospital case study. This figure shows a positive linear trend with an increase of approximately 19.65 ms in the execution time for every pre-allocated task. Although there is an increase in time as the number of pre-allocated tasks increases, we can conclude that the task allocation is still computed in a *reasonable* amount of time in the order of seconds with tests adding up to 50 pre-allocated tasks. Moreover, the

[12]Times deviating from the mean (at 14, 16, 22, 74 and 86 allocations) are considered outliers and can be smoothed.

[13]In terms of states and transitions, for example, when more robots are modelled together or a robot is assigned by a larger number of tasks compared to other allocations.

[14]Hence, the scheduler creates the MDP and DTMC models for all the groups of robots, rather than stopping when one of the models can't generate a feasible plan.

[15]These times are considerably longer compared to RQ1 as we increased the number of iterations from 2 to 100, and the number of allocations from <40 to 100

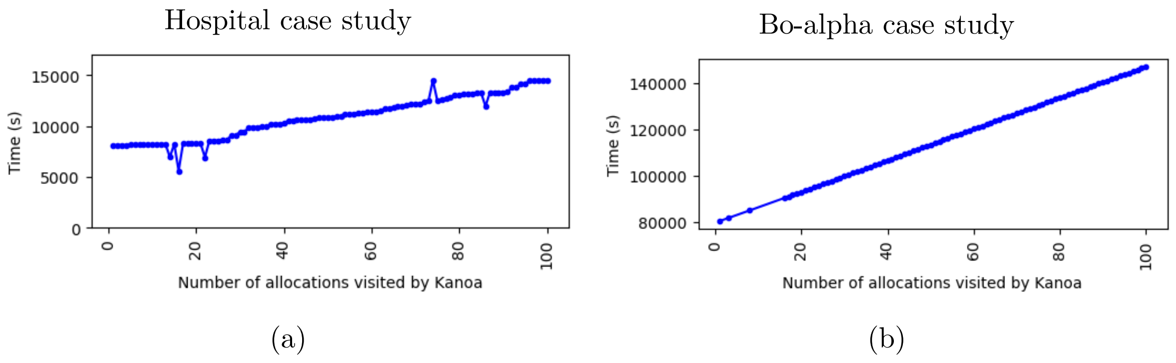


Figure 7.8: KANOA's task scheduler time to iterate over up to 100 allocations.

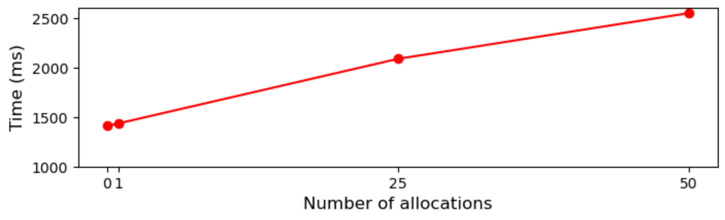


Figure 7.9: Average execution time (over 5 runs) to compute the task-allocation stage when pre-allocated tasks exist.

execution times were fairly consistent over the 5 trials ran for each configuration.[16]

RQ2 findings summary. Scalability is heavily influenced by the number of robots, tasks, and task constraints involved. Limitations were described with the help of Table 7.5. Linear trends are noticed in the computational time taken by the task allocator and the task scheduler. Similar trends are observed for predefined tasks. The time taken by the task allocator is considerably smaller than the time taken by the task scheduler. These findings applied to both case studies.

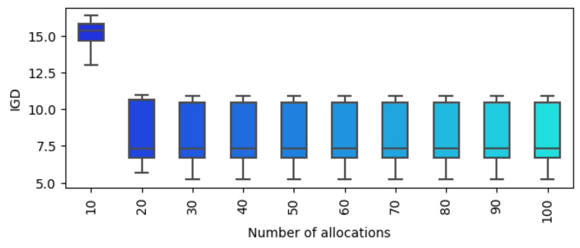
RQ3 (Configurability)

How do KANOA hyperparameters affect the quality of the solutions?

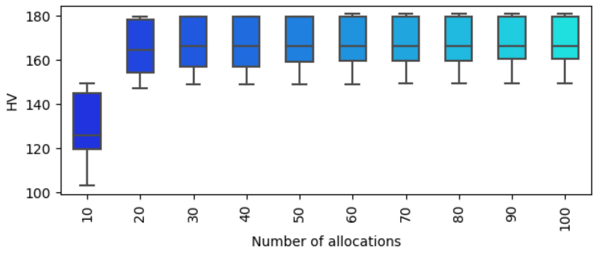
RQ3 experimental setup. Finally, we evaluate how two parameters, i) the number of *allocations* and ii) the number of *iterations* for the KANOA task scheduling, affect the quality of solutions. To assess the impact that the number of allocations has on the Pareto optimal set of solutions we use the data from $e2.2$, obtained by running KANOA for 100 allocations.

Similarly, to evaluate the impact of more number of iterations n_i, we compare the

[16]The standard deviation was too small to be shown in the Figure



(a)



(b)

Figure 7.10: For the hospital case study, quality indicators from solutions found after the task-scheduler iterate over $n_a = 10, 20, ..., 100$ number of allocations.

data obtained for experiments $e1$ and $e2.2$ with $n_i = 2$ and $n_i = 100$, respectively. Given that the data for the hospital case study in $e1$ covers up to 34 allocations, we limits our comparison to this number of allocations. . We compare the median values of the IGD and HV quality indicators.

RQ3 experimental tasks. The experimental tasks are divided into two. For each case study, we:

e3.1. Run KANOA through 100 allocations (data obtained in $e2.2$). Generated the Pareto fronts of solutions found through n allocations, $n = 10, 20, ..., 100$. This generates 10 different variants. For each variant, compute the Pareto quality indicators HV and IGD.

e3.2. Compare the IGD and HV median values obtained by configuring KANOA with $n_i = 2$ against n_i =100 iterations—running KANOA for 34 allocations.

RQ3 results and discussion. (e3.1) For the **hospital** case study, Figure 7.10a and 7.10b show the IGD and HV values, respectively, for the Pareto front of solutions at intervals of 10 allocations, and up to 100 allocations. Lower IGD values are indicative of better quality of the Pareto front of solutions. Comparing the box at 100 allocations and the initial one at 10 allocations reveals a significant improvement in the IGD of the respective solutions. This trend is also observed in the HV box plot, where higher indicates better performance. It is worth noting that there is no difference in the quality of the solutions after 20 allocations. This is because the reference Pareto front of solutions (computed among all solutions from RQ1 and RQ3 data) contained solutions found among the first 19 allocations.

For the **Bo-alpha** case study, Figures 7.11a and 7.11b show the IGD and HV values of the Pareto front of solutions found from 10 to 100 allocations, respectively. Both quality indicators show an improvement as the number of allocations increases. In this case, adding allocations beyond the 90th allocation has no impact on the quality of the

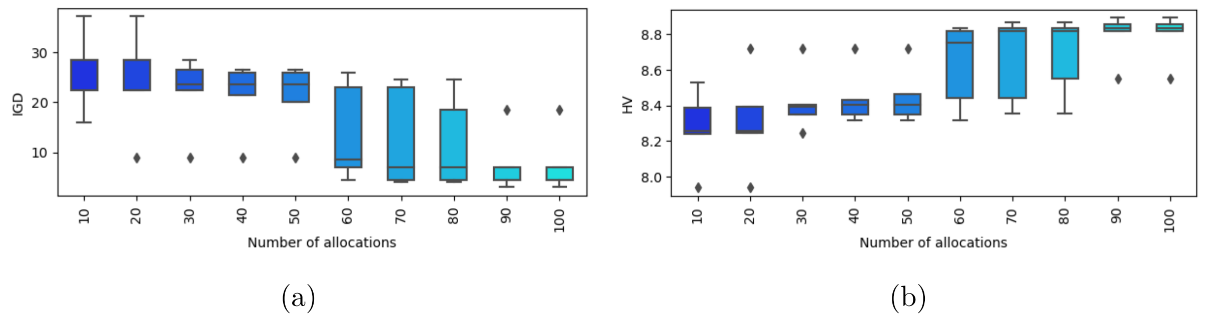


Figure 7.11: For the Bo-alpha case study, quality indicators from solutions found after the task-scheduler iterate over $n_a = 10, 20, ..., 100$ number of allocations.

solutions.

(e3.2). To compare the impact of the number of iterations for the **hospital** case study, we compare the quality of solutions found with $n_i = 2$ (see Figure 7.5 at 400 s) against $n_i = 100$ (depicted at every 10 allocations in Figure 7.10). The median values at 34 allocations are,

With $n_i = 2$ iterations: IGD value of 11.0 and 83.2 HV.
With $n_i = 100$ iterations: IGD value of 7.4 and 164.1 HV.

For the **Bo-alpha** case study, we follow the same process comparing the quality of solutions found with $n_i = 2$ (Figure 7.6 at 400s) against $n_i = 100$ (depicted every 10 allocations in Figure 7.11). The median values at 34 allocations are,

With $n_i = 2$ iterations: IGD value of 33.0 and 5.8 HV.
With $n_i = 100$ iterations: IGD value of 24.0 and 8.4 HV.

Therefore, based on these quality indicators, in both studies more iterations improve the quality of the solutions—for IGD, lower values are preferable; for HV, a greater. This is as expected, as the more iterations are available, the more chances the evolutionary optimisation algorithm has to select, evolve, sort and ultimately uncover *better* solutions.

RQ3 findings summary. Increasing the number of allocations doesn't necessarily improve the quality of the Pareto-optimal solutions. However, increasing the number of iterations increases the quality of the solutions in both case studies.

7.3 Discussion and Limitations

Evaluation of the scalability of KANOA. Evaluating KANOA in terms of tasks, robots in a joint task and task constraints revealed itself to be a complex task. From RQ2,

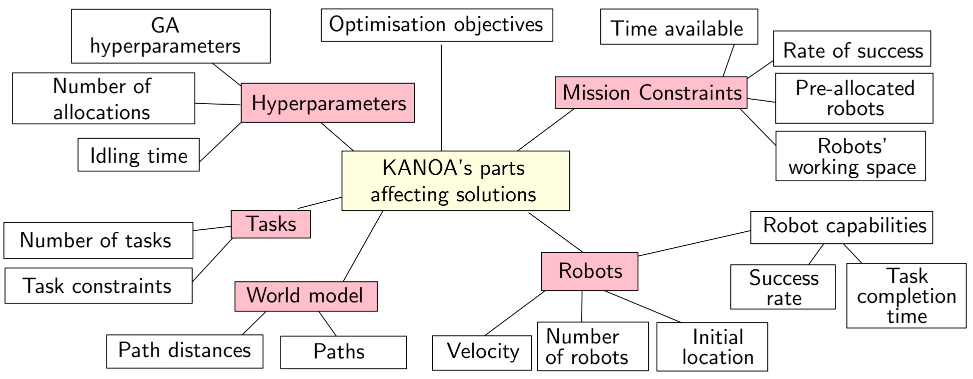


Figure 7.12: KANOA's parts (pink rectangles) and variables (white rectangles) affecting the synthesis of solutions.

comparing the results using different case studies shows that the existence of solutions does not only depend on these three variables. Actually, results in RQ2 are also dependent on KANOA's configuration. For example, when multiple task constraints exist, increasing the population size allows the GA to generate more task permutations, increasing the possibility that some of these comply with task ordering constraints.

A second example is when multiple consecutive task constraints exist. Robots allocated with subtasks of these constraint tasks are required to synchronise in time. Hence, some robots may need to wait until other robots finish the preceding tasks. If the idling time is too short, robots run out of time to wait and complete the rest of their tasks. These complex relations are further discussed in the following paragraphs. In the current evaluation, we considered a subset of the possible variants of our two case studies to evaluate the scalability KANOA. Although the results were very useful to understand the limitations in scalability, further evaluation is required to provide a detailed reference of such limitations.

Parts affecting KANOA's solutions. Figure 7.12 depicts the different parts of KANOA affecting the quality and existence of solutions. These are classified into five *parts*: hyperparameters, mission constraints, robots, world model and tasks. These are subdivided into 18 variables depicted in white rectangles. These variables are defined as part of the problem specification or as hyperparameters of KANOA's tool.

The *hyperparameters* of the system are subdivided into three: the configuration of the GA (i.e., the population size and the number of iterations), the number of allocations and the idling time. The *tasks* variables consist of the number of (atomic and compound) tasks and the number of task constraints (ordered, consecutive and joint). The *world model* variables consist of the number of paths and the path distances; the latter affects the scalability of the task allocator as described in Chapter 5, and the idling time which in turn affects the size of the MDPs as described in Chapter 6.

Robots' variables are divided into the velocity, number, initial location and capabilities of the robots. The robot's capabilities are subdivided into the success rate and completion time of the task performed by these capabilities. As an example, the robots' initial location and their velocity impact the time required to meet other robots at a task location. If the distance to the first task is too far for some robots, others would have to idle for longer.

Finally, the *mission constraints* are as described in KANOA's problem specification: time available to complete the whole mission, the minimum rate of success, pre-allocated tasks to robots and spatial constraints limiting the working space of robots. Finding a better approach to evaluate and assess these dependencies is still an open question we will address in further work.

Task allocation selector. Given that KANOA selects and optimises one task allocation at a time, this approach might result in local minima (when the first synthesised task allocations do not contain solutions belonging to the *true Pareto optimal*). In future work, we will explore the possibility of (a) generating an initial population for the genetic algorithm with previously evaluated task allocations from which solutions (i.e., feasible permutations) were found (b) ranking the allocations depending on the quality of these solutions and (c) modifying the GA to encode the number of the allocation as part of the chromosome and modify the *crossover* stage to only allow the crossover between chromosomes of the same allocation.

Additionally, our observations suggest that in highly restrictive scenarios, the majority of solutions turn out to be infeasible. In such instances, the genetic algorithm struggles to identify an initial viable solution to kickstart the optimisation process and generate similar feasible solutions. Our future efforts aim to develop an algorithm capable of generating only viable sequences of tasks.

Limited number of robots. The scalability of robotic mission planning using logic-based programming languages is discussed in [92], where the ability to devise plans for up to 36 robots is demonstrated. However, instead of employing logic planning techniques like model checking (or PMC as in our case) which suffer from a state-explosion problem, their approach converts the problem into a mixed-integer linear program (MILP) and uses off-the-shelf solvers for the synthesis of plans. KANOA's modular design allows for the integration or substitution of different techniques within its framework. Particularly, since plan synthesis via PMC is hindered by the state explosion problem, other established methods like MILP[17] (and other techniques described in Chapter 3) could be explored in future development.

However, it should be noted that multiple case studies and related approaches report a number of robots in their evaluation similar to the ones we have employed in ours. For example, [22, 49] and [85] report deploying less than five robots; [86, 88, 91] up to 10,

[17]Although MILP comes with the expense of solving the set of a large set of constraints which becomes computationally expensive, as pointed out by [100].

and [87] up to 12.

Multiple robot plans. It is possible that when the robots are about to be deployed, the initial configuration of the system that was considered for the planning has already changed, so the selected robot plans are not feasible anymore. For example, in a scenario where robots are finishing their previous plans and a new plan is about to be deployed, some of the robots may no longer be available as they run out of battery and require recharging. To this end, we maintain an *archive* of feasible task schedules, as shown in stage (5), Figure 7.1. Hence, the robot plans can be swapped before deployment for a new task schedule that complies with the new set of available robots. Similarly, it is possible that not every task allocation in stage (3) generated by stage (3) is used by the task scheduler. Nevertheless, these are archived within KANOA. These allocations can be used in the future to generate more plans in case the task schedules saved in (5) are no longer valid as the system evolves.

Quality indicators. To evaluate the quality of solutions produced by KANOA against those from random search, as well as to compare various versions differing by the number of allocations, we employ Pareto front quality indicators. Despite adhering to conventional methods, it is important to note that these indicators are subject to misinterpretation; as conveyed in [190]: "IGD prefers uniformly distributed solutions and HV is in favour of knee solutions." Knee solutions in the context of multi-objective optimisation are specific points on the Pareto front where making a small improvement in one objective would require a large sacrifice in another objective. These solutions are desirable because they represent a sweet spot of trade-offs between competing objectives. Predicting the shape of the Pareto front is challenging, but conducting more runs can enhance confidence in the accuracy of the quality indicators [190]. To increase the reliability of our evaluation, we will conduct additional experimental runs in subsequent research.

Evolutionary optimisation (GA) loop. For the current version of KANOA, we selected the NSGA-II evolutionary optimisation algorithm and tuned it as described in Section 7.2.3. Further evaluation is required to assess the impact on the quality of solutions of the current configuration against different setups and algorithms, such as SPEA and MOCell (available within the JMetal package [181]).

7.4 Related Work

As part of a carried out systematic review on multi-robot task allocation and scheduling, related work relevant to KANOA's framework is described in Chapter 3. Sections 5.6 and 6.7 also provide related work specific to the task allocation and task scheduling parts of KANOA. In this section, we compare some of the most similar works to KANOA which incorporate formal methods as part of the task allocation and scheduling problem in robotic systems.

Table 7.6 shows KANOA compared to five other studies. The second and third columns indicate what type of problem they solve. Two of them, [25] and [28], assume that the allocation of tasks is known beforehand. Meanwhile, [192] solves the allocation of tasks as part of the path planning solution, while [193], [194] and [195] model the allocation and scheduling of tasks together. The path planning algorithm in [192] uses the so-called Devises for Assisting Living (DALi) algorithm, which in turn is based on a variant of the Dijkastra's algorithm. In [196], Integer Linear Programming (ILP) is used for the allocation of tasks, while constraint programming for the scheduling of tasks for a variant of the multi-travelling salesmen problem.

In [193], a declarative programming language was proposed for the specification of robotic missions, and an SMT-based solver was used for the synthesis of robot plans. All of these approaches, except for [193], [196] and KANOA, use model checking for the generation of robot plans—the outcome of the scheduling stage.

In [193], the authors propose a ranking system ordering robot plans based on the trajectory cost to select the "optimal" task schedule from the set of solutions found. Work in [194] and [195] minimise plans generated from LTL specifications through model checking. Then, they provide optimal guarantees of the synthesised plans. However, this is a mono-objective algorithm minimising the total mission time. In [196], ILP is used for the task allocation minimising the overall span of the mission in terms of travelling time. Subsequently, the task scheduling part is implemented using a constraint solver to address task dependencies. The scheduling process is further optimised with the same objective—minimise the overall mission span—by modelling it as a min-max optimisation problem. In contrast, in KANOA we can model up to three objectives as a result of using a multi-objective evolutionary algorithm.

Key characteristics of each approach are shown in column five. One of the main differences between KANOA and other approaches is the types of constraints that our framework can handle. To maintain clarity, we refer the reader to Table 6.1 for the full list. Also, dividing the problem into the allocation and scheduling of tasks also allowed a broader list of constraints to be added, as (a) some of the constraints only concern one of these two stages in the generation of plans, and (b) it reduces the complexity of encoding both problems into a single one. This is also highlighted in [196].

Many of the compared approaches consider the avoidance of obstacles as part of the mission constraints [28, 192, 193]. KANOA assumes that the planning captured by the locations and paths in the world model already contains enough information to avoid fixed obstacles. Adaptation of plans due to mobile obstacles is one of the tasks we will explore in future work. Additional future research directions can explore key characteristics of other approaches, such as adding guarantees when the travelling time diverges from the expected time and partial knowledge of the robots' actions.

Table 7.6: Comparison between KANOA and similar state-of-the-art approaches for the scheduling of multi-robot tasks.

Approach	**Allocation technique**	**Scheduling technique**	**Optimisation technique [objectives]**	**Key characteristics**
KANOA	Constraint solver	PMC and GA [probability of success, idle time, travel cost]		- Robots heterogeneity - Groups of robots modelled independently - Multi-objective optimisation - Probabilistic robots' behaviour when travelling and attempting tasks -User-friendly DSL
[25]	-	Model Checking	-	- Partial knowledge of robots' actions
[28]	-	Model Checking	-	- Online and offline - Obstacle avoidance - Limited local information available to robots - Fully distributed
[192]	Path planning	Model Checking	-	- Obstacles avoidance - Various road conditions
[193]	SMT-based		Ordering by cost [trajectory cost]	- Failure detection and re-planning - Online and offline - Dynamic obstacle avoidance - Robots heterogeneity
[194]	Model Checking		Graph optimisation [total time]	- Optimality of synthesised plans under LTL specifications
[195]	Model Checking		Graph optimisation [total time]	- Non-deterministic travelling actions - Joint tasks - Deviations from optimal trajectory (upper/lower deviation values) - Optimality of synthesised plans under LTL specifications
[196]	Integer Linear Programming (ILP) & Constraint Programming (CP)		ILP [trajectory cost] and CP [mission time]	- Parallel-tasks single-robot execution - Tasks precedence - Robots heterogeneity

7.5 Summary

This chapter has presented a comprehensive overview and evaluation of KANOA, a tool-supported methodology for task allocation and scheduling for multi-robot systems. The end-to-end framework is presented from the initial problem specification using a domain-specific language to the final generation of robot plans. This process integrates several formal methods: constraint solving for task allocation and probabilistic model checking for the synthesis of robot plans. The latter is leveraged as part of a multi-objective optimisation algorithm to quantitatively assert about the Pareto optimally of feasible robot plans. The chapter also elaborates on the system's capability to handle heterogeneous groups of robots and complex task dependencies, showcasing its utility through two case studies: the deployment of a group of robots in a hospital, and UUVs for collecting samples in the ocean.

The evaluation section of the chapter provides a thorough analysis of KANOA's performance, highlighting its efficacy in generating Pareto optimal task allocations and schedules that enhance operational efficiency while adhering to mission-critical constraints. We show a significant statistical improvement in the quality indicators compared to our baseline. The chapter also discusses potential threats to validity, such as the generalisability of results and the scalability of the approach to larger and more complex scenarios.

In conclusion, this chapter highlights the compositional multi-stage aspect of KANOA, emphasizing its integrated use of formal methods for the generation of correct-by-construction solutions, its flexibility in accommodating various MRS configurations, and its capability to optimising multiple mission-critical parameters.

Chapter 8
Conclusions and Future Work

This thesis has addressed major limitations in two important areas of mission-critical multi-robot systems (MRS). The first one is the unambiguous specification of quantitative robotic mission requirements and the formulation of the problem of generating individual robot plans in a user-friendly language. The second one is the task allocation and scheduling problem for multi-robot heterogeneous systems under multiple functional and non-functional constraints and optimisation objectives.

Multiple previous projects have addressed problems in these two areas. Specifically, logic languages such as linear temporal logic (LTL) have provided ways to describe missions with precise semantics and in an unambiguous manner. Similar languages have extended the capabilities of LTL to reason about time constraints, such as signal temporal logic (STL), which has been successfully applied in the synthesis of robot plans. To the best of our knowledge, prior to this work, there was no existing specification catalogue that compiles quantitative aspects of robotic missions, such as probabilistic behaviour and associated costs, while maintaining the unambiguity of the specifications.

Regarding task allocation and scheduling, variants of these problems have been solved through different methods, some of which were discussed in Chapter 3. However, as MRS are applied in increasingly complex and safety-critical environments, it becomes challenging to identify a single method that fully captures the complexities of the application domain and integrates them into the synthesised robot plans.

Considering these limitations, we defined four research objectives in Section 1.2:

> *1. To ease the definition of an MRS task and scheduling problem for realistic sets of complex constraints and optimisation objectives[...];*
>
> *2. To devise an MRS task allocation method capable of managing a diverse set of constraints, [... and a] task scheduling method that generates individual robot plans optimised to meet multiple mission-critical requirements;*
>
> *3. To integrate the methods from Objective 2 into an end-to-end methodology for MRS task allocation and scheduling that employs formal methods at each stage of its robot schedule generation process.*
>
> *4. To provide tool support for the adoption and testing of the developed techniques for the definition of MRS specifications and the generation of task schedules.*

Within the rest of this chapter, we summarise our research contributions, which address these four research objectives. The contributions are divided into three parts that reflect the stages of the end-to-end framework developed throughout the thesis. First, the contributions in the area of robotic mission specifications address research objective 1, and are presented in Section 8.1. Second, our solutions for the multi-robot task allocation and scheduling problems address research objectives 2 and 3, and are summarised in Sections 8.2 and 8.3. Finally, the KANOA's tool-supported framework contribution addresses research objective 4, and is described in Section 8.3. Each of Sections 8.1–8.3 also provides suggestions for future research directions.

8.1 Robotic Mission Specification

8.1.1 Research Contributions

To support the formalisation of quantitative robotic mission requirements, we have introduced the semantics of the QUARTET repository of missions. QUARTET provides a catalogue for robotic mission developers and practitioners to specify mission-related quantitative specifications. These specifications are initially specified in a domain-specific language. We have proposed a semantics to allow for their translation into Reward Probabilistic Computation Tree Logic (RPCTL) and implemented this in a tool-supported framework. The QUARTET catalogue can be used by practitioners and designers to define a wide range of robotic missions and mission requirements in an unambiguous language without dealing with the intricacies of writing them directly in logic language.

This catalogue concentrates on requirement specifications, which leaves out the syntax and semantics needed to capture essential details of MRS problems such as task allocation and scheduling. The end-to-end tool-supported framework provided by KANOA allowed the description of a complex set of functional and non-functional requirements for the synthesis of robot plans, as well as important details relevant to task allocation and scheduling problems. KANOA's input is the robotic problem specification described in our user-friendly domain-specific language. Following a separation-of-concerns approach, stakeholders and developers can focus on writing different complementary parts of the robotic mission specification.

8.1.2 Further Research Directions

In the area of robotic mission specification, we identify the following research directions to explore in future work:

- **Generalisation of KANOA problem specifications**. The current problem specification of KANOA is restricted to the type of tasks, constraints and objectives described in KANOA's DSL. Future versions of KANOA will generalise the presented problem specification to account for the increasing diversity of multi-robot applications and types of constraints involved (e.g., restricted temporal availability of certain components of the system).

- **Support for RPCTL extensions and other logic languages tailored to robotic mission specifications**. RPCTL was successfully used as part of the QUARTET catalogue of robotic mission requirements and in the robot plan synthesis in KANOA. RPCTL is well-suited to reason about bounded-time requirements, probabilistic behaviour between states of the system, and costs of the system associated with states and transitions. As this logic is not intended to be domain-specific but to generalise well across multiple applications in different fields, intrinsic aspects of robotics are not directly representable in RPCTL. To address this gap, a promising future research direction concerns the formalisation of MRS mission requirements that are not currently representable using RPCTL that may be related to different aspects of the domain. Some of these include adversarial behaviour of parts of the environment, which can be used to provide guarantees about worst-case scenarios for plans and can be encoded in temporal logics such as probabilistic alternating-time temporal logic with rewards (rPATL) [197], or fine-grained temporal properties, which can be captured with languages such as signal temporal logic (STL) [100].

- **Non-functional SLEEC requirements for robotics.** Related to the two previous points, an important area already mentioned as part of the literature review in Chapter 3, is co-bot applications (i.e., applications where robots and humans interact). These were identified as a new category of robot system applications in which robots must be equipped to satisfy non-functional requirements that allow their integration in a *social, legal, ethical, empathetic* and *cultural* manner. The elicitation process of these types of requirements, referred to as the SLEEC rules, is presented in [198]. The addition of SLEEC rules into KANOA is intended for further work.

8.2 Task Allocation

8.2.1 Research Contributions

We addressed the multi-robot task allocation problem using the Alloy Analyzer constraint solver. Our approach successfully addressed the allocation by taking into account the heterogeneity of the robots' capabilities and the spatial constraints outlined in the KANOA mission specification. Internally, KANOA solves the generation of robot plans by dividing the problems of task allocation and scheduling into separate stages. By modelling task allocation independently from task scheduling, we isolated the constraints specific to task allocation. This approach allowed us to focus solely on those constraints in a subset derived from KANOA's problem specification. We formalised constraints in the Alloy language that pre-allocate tasks to specific robots, limit the number of tasks assigned to each robot, and define the spatial boundaries within which robots can operate. We also contributed a specification of the problem in Z notation as an intermediate step to make the transition between Kanoa's DSL and the Alloy specification in a consistent manner, given that the latter is based on Z notation. Moreover, this formalisation can be reused as a reference to perform translations of KANOA problem specifications into other formal languages beyond Alloy.

Our findings demonstrate the practicality of using constraint solvers like the Alloy Analyzer for components of the multi-robot plan synthesis problem formally specified in mathematical-based languages such as Z notation (specifically for task allocation to robots). The applicability and effectiveness of this approach are further validated through a hospital case study.

8.2.2 Further Research Directions

For task allocation, future research directions include:

- **Degradation of requirements**. Consider the example where all robots have to be deployed and work in a specific area. At the same time, there is a requirement stating that robot $r1$ cannot enter this area. In this simple example, no allocation is returned as the two requirements conflict. Resolution of requirements can happen through a human-in-the-loop approach where feedback is provided to the user on what conflicts were detected. Another solution is to systematically degrade some of the requirements such that an allocation can be obtained even when the original requirements were (slightly) modified or removed.

- **Scalability**. One of the problems of using constraint solvers is that the whole set of *atoms* related to robots, tasks and allocation coordinates have to be modelled individually—the maximum or exact numbers of these atoms are defined as part of the defined *scope*. Hence, the scalability of our solution suffers as more atoms are added. Future research is needed to overcome this limitation. For example, by modelling the working area of the robots as discrete areas rather than coordinates, we avoid using unnecessary atoms to model the integers representing specific x and y coordinates.

- **Limitations of constraints solvers**. Another disadvantage of using off-the-shelf constraint solvers is that the computation of solutions is done hidden from the user using a series of techniques to optimise the execution time while returning as many solutions as possible. This is also the case with the Alloy Analyzer (which uses the Kodkod solver under the hood). The complexity of the model itself, including the number of constraints and how they interact, can affect whether Alloy can find a solution. Complex interdependencies can make the solution space difficult to traverse, increasing the likelihood that Alloy might not find an existing solution [55]. Hence, it is important to acknowledge that the Alloy Analyzer does not always return a solution *even if one exists*. Further work must be done to ensure that a solution is found if it exists.

- **Decomposition of tasks.** Several studies consider the decomposition of tasks to describe alternative ways of completing a mission [51, 199]. While our current work does not encompass this decomposition, in principle it could be incorporated into the Alloy problem specification.

8.3 Task Scheduling and KANOA Tool

8.3.1 Research Contributions

In the task scheduling phase, we generate a set of Pareto-optimal robot plans that conform to the mission critical requirements defined in the KANOA problem specification. These plans were synthesised using probabilistic model checking (PMC). KANOA incorporates a variety of specialised models to analyse different facets of the robot plans. The verification results from some of these models facilitate the multi-criteria optimisation of the synthesised robot plan.

As part of the task scheduling solution, three different models were proposed for the quantitative reasoning of robot schedules. The analysis of these models is handled in the evaluation phase by KANOA's meta-heuristic evolutionary search algorithm. The search algorithm explores the solution space and optimises feasible plans. Given that PMC, the technique used for the robot plan synthesis, is susceptible to the state explosion problem, robots are grouped depending on the constraints of their allocated tasks in a pre-scheduling stage. This strategy effectively reduces the states and transitions compared to a single-model approach.

The practicality of KANOA's task scheduler in conjunction with the task allocator, is supported by the delivered compositional tool-supported KANOA framework. Through an extensive evaluation, we demonstrated the applicability and effectiveness of KANOA for the synthesis of Pareto-optimal robot plans under a complex set of constraints and multiple conflicting objectives.

8.3.2 Further Research Directions

To continue the work in task scheduling, we identify the following research directions:

- **Specialised models**. As part of the scheduling of tasks, we develop a framework to integrate a series of specialised models that capture the idling of the robots, the probabilistic behaviour of travelling between locations and the travelling cost. Two of them use probabilistic model checking while the third is a simpler specification of the arithmetic required to sum the total travelling cost of the robots. Additional models reasoning about safety properties, communication between robots, model drifting and other parts of the system can be added to the current set of models, expanding the set of classes of properties that the approach can handle.

- **Integration with other scheduling techniques.** As mentioned in the literature review (Chapter 3), there are multiple existing techniques for the scheduling of tasks in multi-robot systems, such as HTN planners and PDDL solvers. As KANOA's plan synthesis is compartmentalised inside the meta-heuristic search loop, we can augment (or replace) KANOA's method to generate task schedules. In other words, a series of task schedulers can be available for KANOA to select depending on the problem at hand. Future work will define what type of schedulers can be added and how is the selection of the task scheduler done.

- **Runtime task scheduling and self-adaptation**. Due to the inherent sources of uncertainty associated with the robots, their environment, and changes in the robot mission itself, modification of plans to accommodate such changes at runtime is necessary. Multiple studies have taken steps towards perpetual scheduling of tasks [200, 201] and adaptation of these in the presence of disturbances [202, 203]. In future work, we intend to extend KANOA with runtime capabilities for the adaptation of plans. Two of the main decisions to be made are: *what types of disturbances are to be considered as triggering adaptation actions?* and *what stages of the KANOA framework will be extended with runtime capabilities and how will these be modified (e.g., following the MAPE-k architecture [204])?*.

- **Hybrid architectures**. As KANOA's task scheduling is designed as a centralised architecture, the resilience of the system is at risk when the central *task allocator and scheduler* unit fails. Future effort is needed to extend KANOA to decentralised, distributed or hybrid architectures.

- **Scalability**. The task scheduler was designed with a novel series of *measures* to avoid the state-explosion problem inherited from probabilistic model checking. These measures include the grouping of robots, the use of a meta-heuristic evolutionary algorithm for the exploration of the solution space, and the creation of specialised models that capture only specific aspects of the scheduling problem, rather than modelling all aspects together. However, as part of the evaluation in Chapter 7, one of the primary scalability drawbacks is the selection of the permutation of tasks within the evolutionary algorithm. As mentioned in the evaluation chapter, in future work we plan to create an algorithm to select only feasible permutations of tasks complying with the ordering of the tasks' requirements.

- **Digital twins**. Significant advancements in multi-robot digital twin scheduling have become a vital component in narrowing the multi-robot simulation-reality gap. For instance, the development of a re-scheduling digital twin for multi-robot arm applications is detailed in [205]. Various features of KANOA, such as the creation of multiple specialised models are appealing for near real-time applications including digital twins, as it would allow the online and offline verification of different aspects of the robot plans. A critical consideration in real-time applications is managing the time required to generate and analyse these plans. Ensuring that a plan is synthesised within a specific time frame presents significant challenges and is an area earmarked for further research.

REFERENCES

[1] C. Menghi, C. Tsigkanos, M. Askarpour, P. Pelliccione, G. Vazquez, R. Calinescu, and S. Garcia, "Mission specification patterns for mobile robots: Providing support for quantitative properties," *IEEE Transactions on Software Engineering*, 2022.

[2] C. Menghi, C. Tsigkanos, P. Pelliccione, C. Ghezzi, and T. Berger, "Specification patterns for robotic missions," *Transactions on Software Engineering*, vol. 47, no. 10, pp. 2208–2224, 2019.

[3] R. C. Cardoso, G. Kourtis, L. A. Dennis, C. Dixon, M. Farrell, M. Fisher, and M. Webster, "A review of verification and validation for space autonomous systems," *Current Robotics Reports*, vol. 2, no. 3, pp. 273–283, 2021.

[4] B. Devlin-Hill, R. Calinescu, J. Cámara, and I. Caliskanelli, "Towards scalable multi-robot systems by partitioning the task domain," in *Annual Conference Towards Autonomous Robotic Systems.* Springer, 2022, pp. 282–292.

[5] H.-Y. Lee and C. C. Murray, "Robotics in order picking: evaluating warehouse layouts for pick, place, and transport vehicle routing systems," *International Journal of Production Research*, vol. 57, no. 18, pp. 5821–5841, 2019.

[6] A. Jevtić, A. F. Valle, G. Alenyà, G. Chance, P. Caleb-Solly, S. Dogramadzi, and C. Torras, "Personalized robot assistant for support in dressing," *IEEE Transactions on Cognitive and Developmental Systems*, vol. 11, no. 3, pp. 363–374, 2018.

[7] A. Di Lallo, R. Murphy, A. Krieger, J. Zhu, R. H. Taylor, and H. Su, "Medical robots for infectious diseases: lessons and challenges from the COVID-19 pandemic," *IEEE Robotics & Automation Magazine*, vol. 28, no. 1, pp. 18–27, 2021.

[8] Y. B. Sebbane, *Multi-UAV planning and task allocation.* Chapman and Hall/CRC, 2020.

[9] A. Khamis, A. Hussein, and A. Elmogy, "Multi-robot task allocation: A review of the state-of-the-art," *Cooperative Robots and Sensor Networks*, pp. 31–51, 2015.

[10] M. V. Espina, R. Grech, D. De Jager, P. Remagnino, L. Iocchi, L. Marchetti, D. Nardi, D. Monekosso, M. Nicolescu, and C. King, "Multi-robot teams for environmental monitoring," *Innovations in Defence Support Systems*, pp. 183–209, 2011.

[11] R. Almadhoun, T. Taha, L. Seneviratne, J. Dias, and G. Cai, "A survey on inspecting structures using robotic systems," *International Journal of Advanced Robotic Systems*, vol. 13, no. 6, 2016.

[12] P. R. Wurman, R. D'Andrea, and M. Mountz, "Coordinating hundreds of cooperative, autonomous vehicles in warehouses," *AI magazine*, vol. 29, no. 1, 2008.

[13] K. S. Yakovlev, A. Andreychuk, T. Rybeckỳ, and M. Kulich, "On the application of safe-interval path planning to a variant of the pickup and delivery problem." in *ICINCO*, 2020, pp. 521–528.

[14] B. Siciliano and O. Khatib, "Robotics and the handbook," in *Springer Handbook of Robotics.* Springer, 2016, pp. 1–6.

[15] D. S. Drew, "Multi-agent systems for search and rescue applications," *Current Robotics Reports*, vol. 2, pp. 189–200, 2021.

[16] B. Mishra, D. Garg, P. Narang, and V. Mishra, "Drone-surveillance for search and rescue in natural disaster," *Computer Communications*, vol. 156, pp. 1–10, 2020.

[17] R. De Lemos, D. Weyns *et al.*, "Software engineering for self-adaptive systems: research challenges in the provision of assurances," in *Software Engineering for Self-Adaptive Systems.* Springer, 2017, no. 3, pp. 3–30.

[18] D. Weyns, B. Schmerl, V. Grassi *et al.*, "On patterns for decentralized control in self-adaptive systems," in *Software Engineering for Self-Adaptive Systems II.* Springer, 2013, pp. 76–107.

[19] D. Weyns, N. Bencomo, R. Calinescu, J. Cámara, C. Ghezzi *et al.*, "Perpetual assurances for self-adaptive systems," in *Software Engineering for Self-Adaptive Systems III. Assurances.* Springer, 2017, pp. 31–63.

[20] M. Askarpour, C. Tsigkanos, C. Menghi, R. Calinescu, P. Pelliccione, S. García, R. Caldas, T. J. von Oertzen, M. Wimmer, L. Berardinelli *et al.*, "Robomax: Robotic Mission Adaptation eXemplars," in *International Symposium on Software Engineering for Adaptive and Self-Managing Systems.* IEEE, 2021, pp. 245–251.

[21] S. M. Hezavehi, D. Weyns, P. Avgeriou, R. Calinescu, R. Mirandola, and D. Perez-Palacin, "Uncertainty in self-adaptive systems: a research community perspective," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 15, no. 4, pp. 1–36, 2021.

[22] E. Seraj, L. Chen, and M. C. Gombolay, "A hierarchical coordination framework for joint perception-action tasks in composite robot teams," *IEEE Transactions on Robotics*, vol. 38, no. 1, pp. 139–158, 2021.

[23] Y. Xu, Z. Zhang, J. Yu, Y. Shen, and Y. Wang, “A framework to co-optimize robot exploration and task planning in unknown environments,” *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 12 283–12 290, 2022.

[24] Y. Jiang, F. Yang, S. Zhang, and P. Stone, “Task-motion planning with reinforcement learning for adaptable mobile service robots,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2019, pp. 7529–7534.

[25] C. Menghi, S. Garcia, P. Pelliccione, and J. Tumova, “Multi-robot LTL planning under uncertainty,” in *Formal Methods*, no. 22. Springer, 2018, pp. 399–417.

[26] J. Woodcock, P. G. Larsen, J. Bicarregui, and J. Fitzgerald, “Formal methods: Practice and experience,” *ACM Computing SURveys*, vol. 41, no. 4, pp. 1–36, 2009.

[27] P. Schillinger, M. Bürger, and D. V. Dimarogonas, “Simultaneous task allocation and planning for temporal logic goals in heterogeneous multi-robot systems,” *The International Journal of Robotics Research*, vol. 37, no. 7, pp. 818–838, 2018.

[28] P. Yu and D. V. Dimarogonas, “Distributed motion coordination for multirobot systems under LTL specifications,” *IEEE Transactions on Robotics*, vol. 38, no. 2, pp. 1047–1062, 2021.

[29] C. Baier and J.-P. Katoen, *Principles of model checking.* MIT press, 2008.

[30] G. Vázquez, R. Calinescu, and J. Cámara, “Scheduling multi-robot missions with joint tasks and heterogeneous robot teams,” in *Towards Autonomous Robotic Systems*, no. 22. Springer, 2021, pp. 354–359.

[31] S. Khanna and S. Srivastava, “The emergence of AI based autonomous UV disinfection robots in pandemic response and hygiene maintenance,” *International Journal of Applied Health Care Analytics*, vol. 7, no. 11, pp. 1–19, 2022.

[32] R. Bogue, “Robots in a contagious world,” *Industrial Robot: the International Journal of Robotics Research and Application*, 2020.

[33] K. Parnell, S. Merriman, S. Getir Yaman, K. Plant, and R. Calinescu, “Resilient strategies for socially compliant autonomous assistive dressing robots,” in *Proceedings of the First International Symposium on Trustworthy Autonomous Systems*, 2023, pp. 1–9.

[34] R. I. Brafman, D. Tolpin, and O. Wertheim, “Probabilistic programs as an action description language,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, no. 13, 2023, pp. 15 351–15 358.

[35] H. L. Younes and M. L. Littman, "PPDDL1. 0: An extension to PDDL for expressing planning domains with probabilistic effects," *Techn. Rep. CMU-CS-04-162*, vol. 2, p. 99, 2004.

[36] C. Menghi, C. Tsigkanos, T. Berger, and P. Pelliccione, "PsALM: Specification of dependable robotic missions," in *IEEE/ACM International Conference on Software Engineering*, no. 41. IEEE, 2019, pp. 99–102.

[37] G. S. Rodrigues, F. P. Guimarães, G. N. Rodrigues, A. Knauss, J. P. C. de Araújo, H. Andrade, and R. Ali, "GoalD: A Goal-Driven deployment framework for dynamic and heterogeneous computing environments," *Information and Software Technology*, vol. 111, pp. 159–176, 2019.

[38] S. Sariel and T. Balch, "Real time auction based allocation of tasks for multi-robot exploration problem in dynamic environments," in *Proceedings of the AAAI Workshop on Integrating Planning into Scheduling.* Citeseer, 2005, pp. 27–33.

[39] B. P. Gerkey and M. J. Matarić, "A formal analysis and taxonomy of task allocation in multi-robot systems," *The International journal of robotics research*, vol. 23, no. 9, pp. 939–954, 2004.

[40] H. Chakraa, F. Guérin, E. Leclercq, and D. Lefebvre, "Optimization techniques for multi-robot task allocation problems: Review on the state-of-the-art," *Robotics and Autonomous Systems*, 2023.

[41] M. Otte, M. J. Kuhlman, and D. Sofge, "Auctions for multi-robot task allocation in communication limited environments," *Autonomous Robots*, vol. 44, pp. 547–584, 2020.

[42] J. Blazewicz, K. Ecker, E. Pesch, G. Schmidt, and J. Weglarz, *Handbook on scheduling.* Springer, 2019.

[43] A. Ham and M.-J. Park, "Human–robot task allocation and scheduling: Boeing 777 case study," *Robotics and Automation Letters*, vol. 6, no. 2, pp. 1256–1263, 2021.

[44] P. U Lima and L. M Custódio, "Multi-robot systems," *Innovations in Robot Mobility and Control*, pp. 1–64, 2005.

[45] B. Hayes and B. Scassellati, "Autonomously constructing hierarchical task networks for planning and human-robot collaboration," *International Conference on Robotics and Automation*, pp. 5469–5476, 2016.

[46] A. Muluk, H. Akpolat, and J. Xu, "Scheduling problems—an overview," *Journal of Systems Science and Systems Engineering*, vol. 12, no. 4, pp. 481–492, 2003.

[47] S. Alirezazadeh and L. A. Alexandre, "Dynamic task scheduling for human-robot collaboration," *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 8699–8704, 2022.

[48] R. Bezerra, K. Ohno, S. Kojima, H. A. Aryadi, K. Gunji, M. Kuwahara, Y. Okada, M. Konyo, and S. Tadokoro, "Heterogeneous multi-robot task scheduling heuristics for garment mass customization," *International Conference on Automation Science and Engineering*, pp. 439–446, 2022.

[49] G. Notomista, S. Mayya, Y. Emam, C. Kroninger, A. Bohannon, S. Hutchinson, and M. Egerstedt, "A resilient and energy-aware task allocation framework for heterogeneous multirobot systems," *IEEE Transactions on Robotics*, vol. 38, no. 1, pp. 159–179, 2021.

[50] S. Gerasimou, G. Tamburrelli, and R. Calinescu, "Search-based synthesis of probabilistic models for quality-of-service software engineering (t)," in *IEEE/ACM International Conference on Automated Software Engineering*, 2015, pp. 319–330.

[51] G. A. Korsah, A. Stentz, and M. B. Dias, "A comprehensive taxonomy for multi-robot task allocation," *The International Journal of Robotics Research*, vol. 32, no. 12, pp. 1495–1512, 2013.

[52] R. Luna and K. E. Bekris, "Efficient and complete centralized multi-robot path planning," in *International Conference on Intelligent Robots and Systems*. IEEE, 2011, pp. 3268–3275.

[53] J. Woodcock and J. Davies, "Using Z: Specification, refinement and proof," *Prentice Hall International*, 1996.

[54] J. M. Spivey, *The fuzz manual*. Spivey, 1993.

[55] D. Jackson, "Alloy: a lightweight object modelling notation," *ACM Transactions on Software Engineering and Methodology*, vol. 11, no. 2, pp. 256–290, 2002.

[56] E. Torlak and D. Jackson, "Kodkod: a relational model finder," *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pp. 632–647, 2007.

[57] Alloy. (2023) *Alloy Quickguide* [Online]. Available: http://alloytools.org/quickguide/gui.html. [Accessed: 08 November 2023].

[58] W. Wang, K. Wang, M. Zhang, and S. Khurshid, "Learning to optimize the alloy analyzer," in *IEEE Conference on Software Testing, Validation and Verification*, 2019, pp. 228–239.

[59] Alloy. (2023) *Which SAT solver should I tell Alloy to use?* [Online]. Available: http://alloytools.org/faq/which_sat_solver_should_i_tell_alloy_to_use.html. [Accessed: 08 November 2023].

[60] V. Forejt, M. Kwiatkowska, G. Norman, and D. Parker, "Automated verification techniques for probabilistic systems," *International School on Formal Methods for the Design of Computer, Communication and Software Systems, SFM*, no. 11, pp. 53–113, 2011.

[61] M. Kwiatkowska and D. Parker, "Automated verification and strategy synthesis for probabilistic systems," *Automated Technology for Verification and Analysis*, no. 11, pp. 5–22, 2013.

[62] M. Kwiatkowska, G. Norman, and D. Parker, "Probabilistic model checking and autonomy," *Annual review of control, robotics, and autonomous systems*, vol. 5, no. 1, pp. 385–410, 2022.

[63] H. Hansson and B. Jonsson, "A logic for reasoning about time and reliability," *Formal aspects of computing*, vol. 6, no. 5, pp. 512–535, 1994.

[64] F. Faruq, "Verified multi-robot planning under uncertainty," Ph.D. dissertation, University of Birmingham, 2022.

[65] M. Y. Vardi, "Automatic verification of probabilistic concurrent finite state programs," in *Symposium on Foundations of Computer Science*, no. 26. IEEE, 1985, pp. 327–338.

[66] R. Alur and D. Dill, "Automata for modeling real-time systems," in *International Colloquium on Automata, Languages, and Programming*. Springer, 1990, pp. 322–335.

[67] R. Alur, C. Courcoubetis, and D. Dill, "Verifying automata specifications of probabilistic real-time systems," in *Research and Education in Concurrent Systems*. Springer, 1991, pp. 28–44.

[68] A. Bianco and L. De Alfaro, "Model checking of probabilistic and nondeterministic systems," in *International Conference on Foundations of Software Technology and Theoretical Computer Science*. Springer, 1995, pp. 499–513.

[69] C. Daws, "Symbolic and parametric model checking of discrete-time Markov chains," in *International Colloquium on Theoretical Aspects of Computing*. Springer, 2004, pp. 280–294.

[70] E. M. Hahn, A. Hartmanns, C. Hensel, M. Klauck, J. Klein, J. Křetínskỳ, D. Parker, T. Quatmann, E. Ruijters, and M. Steinmetz, "The 2019 comparison of tools for the analysis of quantitative formal models," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems.* Springer, 2019, pp. 69–92.

[71] M. Kwiatkowska, G. Norman, and D. Parker, "Prism 4.0: Verification of probabilistic real-time systems," in *International Conference on Computer Aided Verification.* Springer, 2011, pp. 585–591.

[72] M. Kwiatkowska, D. Parker, and G. Norman, "PRISM: probabilistic symbolic model checker," in *International Conference on Modelling Techniques and Tools for Computer Performance Evaluation.* Springer, 2002, pp. 200–204.

[73] PRISM probabilistic model checker web site [Online]. Available: https://www.prismmodelchecker.org. [Accessed: 2023-01-08].

[74] S. P. Boyd and L. Vandenberghe, *Convex optimization.* Cambridge university press, 2004.

[75] J. R. Koza, "Genetic programming as a means for programming computers by natural selection," *Statistics and computing*, vol. 4, no. 2, pp. 87–112, 1994.

[76] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[77] S. Gerasimou, R. Calinescu, and G. Tamburrelli, "Synthesis of probabilistic models for quality-of-service software engineering," *Automated Software Engineering*, vol. 25, no. 4, pp. 785–831, 2018.

[78] S. Gerasimou, J. Cámara, R. Calinescu, N. Alasmari, F. Alhwikem, and X. Fang, "Evolutionary-guided synthesis of verified Pareto-optimal MDP policies," in *IEEE/ACM International Conference on Automated Software Engineering.* IEEE, 2021, pp. 842–853.

[79] J. Biolchini, P. G. Mian, A. C. C. Natali, and G. H. Travassos, "Systematic review in software engineering," *System engineering and computer science department COPPE/UFRJ, Technical Report ES*, vol. 679, no. 05, p. 45, 2005.

[80] S. Keele *et al.*, "Guidelines for performing systematic literature reviews in software engineering," Technical report, ver. 2.3 ebse technical report. ebse, Tech. Rep., 2007.

[81] G. C. Silva, L. M. Rose, and R. Calinescu, "A systematic review of cloud lock-in solutions," in *2013 IEEE 5th International Conference on Cloud Computing Technology and Science*, vol. 2. IEEE, 2013, pp. 363–368.

[82] DBLP. (2022) *DBLP: Computer Science Bibliography* [Online]. Available: https://dblp.org. [Accessed: 27 November 2022].

[83] S. Garcia, P. Pelliccione, C. Menghi, T. Berger, and T. Bures, "Promise: high-level mission specification for multiple robots," in *International Conference on Software Engineering: Companion Proceedings*, 2020, pp. 5–8.

[84] S. García, P. Pelliccione, C. Menghi, T. Berger, and T. Bures, "High-level mission specification for multiple robots," in *ACM International Conference on Software Language Engineering*, 2019, pp. 127–140.

[85] S. Mayya, R. K. Ramachandran, L. Zhou, V. Senthil, D. Thakur, G. S. Sukhatme, and V. Kumar, "Adaptive and risk-aware target tracking for robot teams with heterogeneous sensors," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 5615–5622, 2022.

[86] A. Fang and H. Kress-Gazit, "Automated task updates of temporal logic specifications for heterogeneous robots," in *International Conference on Robotics and Automation*. IEEE, 2022, pp. 4363–4369.

[87] S. Wang, Y. Liu, Y. Qiu, and J. Zhou, "Consensus-based decentralized task allocation for multi-agent systems and simultaneous multi-agent tasks," *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 12 593–12 600, 2022.

[88] B. A. Ferreira, T. Petrović, and S. Bogdan, "Distributed mission planning of complex tasks for heterogeneous multi-robot systems," in *International Conference on Automation Science and Engineering*. IEEE, 2022, pp. 1224–1231.

[89] D. Gundana and H. Kress-Gazit, "Event-based signal temporal logic tasks: execution and feedback in complex environments," *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 10 001–10 008, 2022.

[90] A. Messing, G. Neville, S. Chernova, S. Hutchinson, and H. Ravichandar, "GRSTAPS: Graphically Recursive Simultaneous Task Allocation, Planning, and Scheduling," *The International Journal of Robotics Research*, vol. 41, no. 2, pp. 232–256, 2022.

[91] S. Paull, P. Ghassemi, and S. Chowdhury, "Learning scalable policies over graphs for multi-robot task allocation using capsule attention networks," in *IEEE International Conference on Robotics and Automation*, 2022, pp. 8815–8822.

[92] K. Leahy, Z. Serlin, C.-I. Vasile, A. Schoer, A. M. Jones, R. Tron, and C. Belta, “Scalable and robust algorithms for task-based coordination from high-level specifications (ScRATCHeS),” *IEEE Transactions on Robotics*, vol. 38, no. 4, pp. 2516–2535, 2021.

[93] G. Vázquez, R. Calinescu, and J. Cámara, “Scheduling of missions with constrained tasks for heterogeneous robot systems,” in *Fourth Workshop on Formal Methods for Autonomous Systems*, 2022.

[94] E. Suslova and P. Fazli, “Multi-robot task allocation with time window and ordering constraints,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2020, pp. 6909–6916.

[95] J. Salvado, M. Mansouri, and F. Pecora, “A network-flow reduction for the multi-robot goal allocation and motion planning problem,” in *IEEE International Conference on Automation Science and Engineering*, 2021, pp. 2194–2201.

[96] W. Smith and Y. Zhang, “Achieving multitasking robots in multi-robot tasks,” in *IEEE International Conference on Robotics and Automation*, 2021, pp. 8948–8954.

[97] R. F. dos Santos, E. R. Nascimento, and D. G. Macharet, “Anytime fault-tolerant adaptive routing for multi-robot teams,” in *International Conference on Robotics and Automation.* IEEE, 2021, pp. 7936–7942.

[98] Y. Chen, U. Rosolia, and A. D. Ames, “Decentralized task and path planning for multi-robot systems,” *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 4337–4344, 2021.

[99] J. Salvado, M. Mansouri, and F. Pecora, “Combining multi-robot motion planning and goal allocation using roadmaps,” in *International Conference on Robotics and Automation.* IEEE, 2021, pp. 10 016–10 022.

[100] D. Gundana and H. Kress-Gazit, “Event-based signal temporal logic synthesis for single and multi-robot tasks,” *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 3687–3694, 2021.

[101] Z. Liu, H. Wei, H. Wang, H. Li, and H. Wang, “Integrated task allocation and path coordination for large-scale robot networks with uncertainties,” *IEEE Transactions on Automation Science and Engineering*, vol. 19, no. 4, pp. 2750–2761, 2021.

[102] Z. Chen, J. Alonso-Mora, X. Bai, D. D. Harabor, and P. J. Stuckey, “Integrated task assignment and path planning for capacitated multi-agent pickup and delivery,” *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 5816–5823, 2021.

[103] S. Park, Y. D. Zhong, and N. E. Leonard, “Multi-robot task allocation games in dynamically changing environments,” in *IEEE International Conference on Robotics and Automation*, 2021, pp. 8678–8684.

[104] R. Bai, R. Zheng, M. Liu, and S. Zhang, “Multi-robot task planning under individual and collaborative temporal logic specifications,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2021, pp. 6382–6389.

[105] P. Forte, A. Mannucci, H. Andreasson, and F. Pecora, “Online task assignment and coordination in multi-robot fleets,” *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 4584–4591, 2021.

[106] S. Mayya, D. S. D’antonio, D. Saldaña, and V. Kumar, “Resilient task allocation in heterogeneous multi-robot systems,” *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 1327–1334, 2021.

[107] O. Al-Buraiki and P. Payeur, “Task allocation in multi-robot systems based on the suitability level of the individual agents,” in *IEEE International Conference on Automation Science and Engineering*, 2021, pp. 209–214.

[108] Y. Emam, S. Mayya, G. Notomista, A. Bohannon, and M. Egerstedt, “Adaptive task allocation for heterogeneous multi-robot teams with evolving and unknown robot capabilities,” in *IEEE International Conference on Robotics and Automation*, 2020, pp. 7719–7725.

[109] Z.-Y. Zhang, Y.-F. Chen, D. Xu, S.-Y. Gong, and Y.-L. Meng, “Research on multi-robot task allocation algorithm based on hadtql,” in *IEEE International Workshop on Electronic Communication and Artificial Intelligence*, 2020, pp. 164–169.

[110] C. Nam and D. A. Shell, “Robots in the huddle: Upfront computation to reduce global communication at run time in multirobot task allocation,” *IEEE Transactions on Robotics*, vol. 36, no. 1, pp. 125–141, 2019.

[111] J. K. Behrens, K. Stepanova, and R. Babuska, “Simultaneous task allocation and motion scheduling for complex tasks executed by multiple robots,” in *IEEE International Conference on Robotics and Automation*, 2020, pp. 11 443–11 449.

[112] H. Hong, W. Kang, and S. Ha, “Software development framework for cooperating robots with high-level mission specification,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2020, pp. 11 615–11 622.

[113] M. J. Schuster, M. G. Müller, S. G. Brunner, H. Lehner, P. Lehner, R. Sakagami, A. Dömel, L. Meyer, B. Vodermayer, R. Giubilato *et al.*, “The arches space-analogue demonstration mission: Towards heterogeneous teams of autonomous robots for

collaborative scientific sampling in planetary exploration," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 5315–5322, 2020.

[114] D. S. Losvik and A. Rutle, "A domain-specific language for the development of heterogeneous multi-robot systems," in *ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion*, no. 22, 2019, pp. 549–558.

[115] J. Bae, J. Lee, and W. Chung, "A heuristic for task allocation and routing of heterogeneous robots while minimizing maximum travel cost," in *IEEE International Conference on Robotics and Automation*, 2019, pp. 4531–4537.

[116] Y. Wang, W. Tang, and S. Xiong, "A multi-task scheduling algorithm for cloud robots," in *IEEE International Conference on Service-Oriented System Engineering*, 2019, pp. 344–3445.

[117] V. Tereshchuk, J. Stewart, N. Bykov, S. Pedigo, S. Devasia, and A. G. Banerjee, "An efficient scheduling algorithm for multi-robot task allocation in assembling aircraft structures," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3844–3851, 2019.

[118] N. Tsiogkas and D. M. Lane, "An evolutionary algorithm for online, resource-constrained, multivehicle sensing mission planning," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 1199–1206, 2018.

[119] I. Jang, H.-S. Shin, and A. Tsourdos, "Anonymous hedonic game for task allocation in a large-scale multiple agent system," *IEEE Transactions on Robotics*, vol. 34, no. 6, pp. 1534–1548, 2018.

[120] A. Rutle, J. Backer, K. Foldøy, and R. T. Bye, "CommonLang: a DSL for defining robot tasks," *Technical University of Aachen*, 2018.

[121] M. C. Gombolay, R. J. Wilcox, and J. A. Shah, "Fast scheduling of robot teams performing tasks with temporospatial constraints," *IEEE Transactions on Robotics*, vol. 34, no. 1, pp. 220–239, 2018.

[122] G. Vazquez, "Automated scheduling of multi-robot system missions: An architectural perspective," *European Conference on Software Architecture*, 2021.

[123] A. W. Palmer, A. J. Hill, and S. J. Scheding, "Modelling resource contention in multi-robot task allocation problems with uncertain timing," in *IEEE International Conference on Robotics and Automation*, 2018, pp. 3693–3700.

[124] A. Ham, "Transfer-robot task scheduling in job shop," *International Journal of Production Research*, vol. 59, no. 3, pp. 813–823, 2021.

[125] D. Li, Q. Wang, W. Zou, H. Su, X. Wang, and X. Xu, “An efficient approach for solving robotic task sequencing problems considering spatial constraint,” in *IEEE International Conference on Automation Science and Engineering*, no. 18, 2022, pp. 60–66.

[126] M. Mattamala, N. Chebrolu, and M. Fallon, “An efficient locally reactive controller for safe navigation in visual teach and repeat missions,” *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 2353–2360, 2022.

[127] F. Reister, M. Grotz, and T. Asfour, “Combining navigation and manipulation costs for time-efficient robot placement in mobile manipulation tasks,” *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 9913–9920, 2022.

[128] E. Scheide, G. Best, and G. A. Hollinger, “Behavior tree learning for robotic task planning through Monte Carlo dag search over a formal grammar,” in *IEEE International Conference on Robotics and Automation*, 2021, pp. 4837–4843.

[129] S. Kim, K. Jang, S. Park, Y. Lee, S. Y. Lee, and J. Park, “Continuous task transition approach for robot controller based on hierarchical quadratic programming,” *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 1603–1610, 2019.

[130] N. T. Dantam, Z. K. Kingston, S. Chaudhuri, and L. E. Kavraki, “An incremental constraint-based framework for task and motion planning,” *The International Journal of Robotics Research*, vol. 37, no. 10, pp. 1134–1151, 2018.

[131] C. R. Garrett, T. Lozano-Perez, and L. P. Kaelbling, “Ffrob: Leveraging symbolic planning for efficient task and motion planning,” *The International Journal of Robotics Research*, vol. 37, no. 1, pp. 104–136, 2018.

[132] C. Heinzemann and R. Lange, “vTSL-A Formally Verifiable DSL for Specifying Robot Tasks,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2018, pp. 8308–8314.

[133] J. Cámara, B. Schmerl, and D. Garlan, “Software architecture and task plan co-adaptation for mobile service robots,” *IEEE/ACM International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, no. 15, pp. 125–136, 2020.

[134] W. Lian, T. Kelch, D. Holz, A. Norton, and S. Schaal, “Benchmarking off-the-shelf solutions to robotic assembly tasks,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2021, pp. 1046–1053.

[135] C. Messeri, A. Bicchi, A. M. Zanchettin, and P. Rocco, “A dynamic task allocation strategy to mitigate the human physical fatigue in collaborative robotics,” *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 2178–2185, 2022.

[136] W. Si, Y. Guan, and N. Wang, “Adaptive compliant skill learning for contact-rich manipulation with human in the loop,” *IEEE Robotics and Automation Letters*, vol. 7, no. 3, pp. 5834–5841, 2022.

[137] A. Pupa and C. Secchi, “A safety-aware architecture for task scheduling and execution for human-robot collaboration,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2021, pp. 1895–1902.

[138] S. Bustamante, G. Quere, D. Leidner, J. Vogel, and F. Stulp, “CATs: task planning for shared control of assistive robots with variable autonomy,” in *IEEE International Conference on Robotics and Automation*, 2022, pp. 3775–3782.

[139] N. Gjeldum, A. Aljinovic, M. Crnjac Zizic, and M. Mladineo, “Collaborative robot task allocation on an assembly line using the decision support system,” *International Journal of Computer Integrated Manufacturing*, vol. 35, no. 4-5, pp. 510–526, 2022.

[140] N. Nikolakis, K. Sipsas, P. Tsarouchi, and S. Makris, “On a shared human-robot task scheduling and online re-scheduling,” *Procedia CIRP*, vol. 78, pp. 237–242, 2018.

[141] Wordcloud. (2023) *Word clouds web site for the creation of word clouds* [Online]. Available: https://www.wordclouds.com. [Accessed: 08 January 2023].

[142] G. F. Solano, R. D. Caldas, G. N. Rodrigues, T. Vogel, and P. Pelliccione, “Taming uncertainty in the assurance process of self-adaptive systems: a goal-oriented approach,” in *IEEE/ACM International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, no. 14, 2019, pp. 89–99.

[143] G. M. Skaltsis, H.-S. Shin, and A. Tsourdos, “A survey of task allocation techniques in mas,” in *2021 International Conference on Unmanned Aircraft Systems*. IEEE, 2021, pp. 488–497.

[144] P. Tsarouchi, J. Spiliotopoulos, G. Michalos, S. Koukas, A. Athanasatos, S. Makris, and G. Chryssolouris, “A decision making framework for human robot collaborative workplace generation,” *Procedia Cirp*, vol. 44, pp. 228–232, 2016.

[145] S. J. Russell and P. Norvig, *Artificial intelligence a modern approach.* London, 2010.

[146] C. Menghi, C. Tsigkanos, M. Askarpour, P. Pelliccione, G. Vazquez, R. Calinescu, and S. Garcia. (2022) *Robotic Patterns Tool and Replication Package* [Online]. Available: roboticpatterns.com/quantitative. [Accessed: 08 January 2023].

[147] E. F. Inc. (2022) *Xtext For the specification of languages* [Online]. Available: http://www.eclipse.org/Xtext/. [Accessed: 08 January 2023].

[148] D. B. Licea, M. Bonilla, M. Ghogho, S. Lasaulce, and V. S. Varma, "Communication-aware energy efficient trajectory planning with limited channel knowledge," *IEEE Transactions on Robotics*, vol. 36, no. 2, pp. 431–442, 2019.

[149] PRISM case studies web site [Online]. Available: https://www.prismmodelchecker.org/casestudies/index.php. [Accessed: 2022-01-08].

[150] I. Cizelj, X. C. D. Ding, M. Lahijanian, A. Pinto, and C. Belta, "Probabilistically safe vehicle control in a hostile environment," *IFAC Proceedings Volumes*, vol. 44, no. 1, pp. 11 803–11 808, 2011.

[151] M. Kwiatkowska, D. Parker, and C. Wiltsche, "Prism-games: verification and strategy synthesis for stochastic multi-player games with multiple objectives," *International Journal on Software Tools for Technology Transfer*, vol. 20, pp. 195–210, 2018.

[152] R. P. de Araújo, A. C. Mota, and S. de Carvalho Nogueira, "Probabilistic analysis applied to cleaning robots," in *2017 IEEE International Conference on Information Reuse and Integration (IRI)*. IEEE, 2017, pp. 275–282.

[153] K. Ye, A. Cavalcanti, S. Foster, A. Miyazawa, and J. Woodcock, "Probabilistic modelling and verification using robochart and prism," *Software and Systems Modeling*, pp. 1–50, 2022.

[154] D. Applegate, R. Bixby, V. Chvátal, and W. Cook, "Tsp cuts which do not conform to the template paradigm," *Computational Combinatorial Optimization: Optimal or Provably Near-Optimal Solutions*, pp. 261–303, 2001.

[155] G. Nollert and S. Wich, "Planning a cardiovascular hybrid operating room: the technical point of view," in *The heart surgery forum*, 2008, p. 1033.

[156] E. B. Gil, G. N. Rodrigues, P. Pelliccione, and R. Calinescu, "Mission specification and decomposition for multi-robot systems," *Robotics and Autonomous Systems*, vol. 163, p. 104386, 2023.

[157] H. Bourbouh *et al.*, "Integrating formal verification and assurance: an inspection rover case study," in *NASA Formal Methods*. Springer, 2021, pp. 53–71.

[158] M. Farrell *et al.*, "Formal modelling and runtime verification of autonomous grasping for active debris removal," *Frontiers in Robotics and AI*, vol. 8, p. 639282, 2022.

[159] L. Grunske, "Specification patterns for probabilistic quality properties," in *International Conference Software Engineering*, 2008, pp. 31–40.

[160] M. B. Dwyer *et al.*, “Patterns in property specifications for finite-state verification,” in *Software engineering*, 1999, pp. 411–420.

[161] C. Bolton, “Using the alloy analyzer to verify data refinement in z,” *Electronic Notes in Theoretical Computer Science*, vol. 137, no. 2, pp. 23–44, 2005.

[162] D. Jackson, *Software Abstractions: logic, language, and analysis.* MIT press, 2012.

[163] N. Atay and B. Bayazit, “Mixed-integer linear programming solution to multi-robot task allocation problem,” *Louis: St. Louis, MI, USA*, 2006.

[164] M. B. Dias, R. Zlot, N. Kalra, and A. Stentz, “Market-based multirobot coordination: A survey and analysis,” *Proceedings of the IEEE*, vol. 94, no. 7, pp. 1257–1270, 2006.

[165] H.-L. Choi, L. Brunet, and J. P. How, “Consensus-based decentralized auctions for robust task allocation,” *IEEE transactions on robotics*, vol. 25, no. 4, pp. 912–926, 2009.

[166] T. Bektas, “The multiple traveling salesman problem: an overview of formulations and solution procedures,” *omega*, vol. 34, no. 3, pp. 209–219, 2006.

[167] S. Giordani, M. Lujak, and F. Martinelli, “A distributed algorithm for the multi-robot task allocation problem,” in *International Conference on Industrial Engineering and Other Applications of Applied Intelligent Systems.* Springer, 2010, pp. 721–730.

[168] H. W. Kuhn, “The Hungarian method for the assignment problem,” *Naval research logistics quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.

[169] N. T. Dantam, Z. K. Kingston, S. Chaudhuri, and L. E. Kavraki, “Incremental task and motion planning: a constraint-based approach.” in *Robotics: Science and systems*, vol. 12, 2016, p. 00052.

[170] L. De Moura and N. Bjørner, “Z3: an efficient smt solver,” in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems.* Springer, 2008, pp. 337–340.

[171] D. H. Lehmer, “Teaching combinatorial tricks to a computer,” *Combinatorial Analysis*, pp. 179–193, 1960.

[172] H. S. Warren Jr, “A modification of Warshall’s algorithm for the transitive closure of binary relations,” *Communications of the ACM*, vol. 18, no. 4, pp. 218–220, 1975.

[173] M. Kwiatkowska and C. Thachuk, “Probabilistic model checking for biology,” in *Software Systems Safety.* IOS Press, 2014, pp. 165–189.

[174] PRISM. PRISM model checker multi-objective properties web site [Online]. Available: https://www.pris.mmodelchecker.org/manual/PropertySpecification/Multi-objectiveProperties. [Accessed: 10 February 2023].

[175] M. Frasheri, B. Miloradović, L. Esterle, and A. V. Papadopoulos, "Glocal: A hybrid approach to the multi-agent mission re-planning problem," in *IEEE Symposium Series on Computational Intelligence*, 2023, pp. 1696–1703.

[176] R. Giaquinta, R. Hoffmann, M. Ireland, A. Miller, and G. Norman, "Strategy synthesis for autonomous agents using prism," in *NASA Formal Methods*, no. 10. Springer, 2018, pp. 220–236.

[177] T. Zhang, W. Gruver, and M. H. Smith, "Team scheduling by genetic search," in *IEEE International Conference on Intelligent Processing and Manufacturing of Materials*, vol. 2, 1999, pp. 839–844.

[178] Z. Yu, L. Jinhai, G. Guochang, Z. Rubo, and Y. Haiyan, "An implementation of evolutionary computation for path planning of cooperative mobile robots," in *IEEE World Congress on Intelligent Control and Automation*, vol. 3, 2002, pp. 1798–1802.

[179] J. Li, M. Zhou, Q. Sun, X. Dai, and X. Yu, "Colored traveling salesman problem," *IEEE transactions on cybernetics*, vol. 45, no. 11, pp. 2390–2401, 2014.

[180] L. Bettini, *Implementing domain-specific languages with Xtext and Xtend.* Packt Publishing Ltd, 2016.

[181] J. J. Durillo and A. J. Nebro, "jMetal: a Java framework for multi-objective optimization," *Advances in Engineering Software*, vol. 42, no. 10, pp. 760–771, 2011.

[182] A. Arcuri and L. Briand, "A practical guide for using statistical tests to assess randomized algorithms in software engineering," in *Proceedings of the 33rd international conference on software engineering*, 2011, pp. 1–10.

[183] M. Harman, S. A. Mansouri, and Y. Zhang, "Search-based software engineering: Trends, techniques and applications," *ACM Computing Surveys (CSUR)*, vol. 45, no. 1, pp. 1–61, 2012.

[184] P. Runeson, M. Host, A. Rainer, and B. Regnell, *Case study research in software engineering: Guidelines and examples.* John Wiley & Sons, 2012.

[185] S. Easterbrook, J. Singer, M.-A. Storey, and D. Damian, "Selecting empirical methods for software engineering research," *Guide to advanced empirical software engineering*, pp. 285–311, 2008.

[186] J. Harbin, S. Gerasimou, N. Matragkas, A. Zolotas, and R. Calinescu, "Model-driven simulation-based analysis for multi-robot systems," in *ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, no. 24, 2021, pp. 331–341.

[187] Ocean AI. *Bo-Alpha simulating obstacles mission* [Online]. Available: https://oceanai.mit.edu/ivpman/pmwiki/pmwiki.php?n=IvPTools.UFldObstacleSim. [Accessed: 20 March 2024].

[188] S. Wang, S. Ali, T. Yue, Y. Li, and M. Liaaen, "A practical guide to select quality indicators for assessing Pareto-based search algorithms in search-based software engineering," in *International Conference on Software Engineering*, 2016, pp. 631–642.

[189] G. Rodrigues, R. Caldas, G. Araujo, V. de Moraes, G. Rodrigues, and P. Pelliccione, "An architecture for mission coordination of heterogeneous robots," *Journal of Systems and Software*, vol. 191, p. 111363, 2022.

[190] M. Li, T. Chen, and X. Yao, "How to evaluate solutions in pareto-based search-based software engineering: A critical review and methodological guidance," *IEEE Transactions on Software Engineering*, vol. 48, no. 5, pp. 1771–1799, 2020.

[191] D. A. Van Veldhuizen, *Multiobjective evolutionary algorithms: classifications, analyses, and new innovations.* Air Force Institute of Technology, 1999.

[192] R. Gu, E. Baranov, A. Ameri *et al.* (2022) *Mission planning for multiple autonomous agents under complex road conditions: model-checking-based synthesis and verification* [Online]. Available: http://www.es.mdh.se/pdf_publications/6427.pdf. [Accessed: 11 September 2022].

[193] I. Gavran, R. Majumdar, and I. Saha, "Antlab: a multi-robot task server," *ACM Transactions on Embedded Computing Systems*, vol. 16, no. 5s, pp. 1–19, 2017.

[194] A. Ulusoy, S. L. Smith, X. C. Ding, C. Belta, and D. Rus, "Optimal multi-robot path planning with temporal logic constraints," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2011, pp. 3087–3092.

[195] A. Ulusoy, S. L. Smith, and C. Belta, "Optimal multi-robot path planning with LTL constraints: guaranteeing correctness through synchronization," in *Distributed Autonomous Robotic Systems*, no. 11. Springer, 2014, pp. 337–351.

[196] B. Miloradović, B. Çürüklü, M. Ekström, and A. V. Papadopoulos, "Optimizing parallel task execution for multi-agent mission planning," *IEEE Access*, vol. 11, pp. 24 367–24 381, 2023.

[197] T. Chen, V. Forejt, M. Kwiatkowska, D. Parker, and A. Simaitis, “Automatic verification of competitive stochastic systems,” *Formal Methods in System Design*, vol. 43, pp. 61–92, 2013.

[198] S. G. Yaman, C. Burholt, M. Jones, R. Calinescu, and A. Cavalcanti, “Specification and validation of normative rules for autonomous agents,” in *International Conference on Fundamental Approaches to Software Engineering.* Springer Nature Switzerland Cham, 2023, pp. 241–248.

[199] I. Georgievski and M. Aiello, “HTN planning: overview, comparison, and beyond,” *Artificial Intelligence*, vol. 222, pp. 124–156, 2015.

[200] N. Hawes, C. Burbridge, F. Jovan, L. Kunze, B. Lacerda, L. Mudrova, J. Young, J. Wyatt, D. Hebesberger, T. Kortner *et al.*, “The strands project: Long-term autonomy in everyday environments,” *IEEE Robotics & Automation Magazine*, vol. 24, no. 3, pp. 146–156, 2017.

[201] L. Kunze, N. Hawes, T. Duckett, M. Hanheide, and T. Krajník, “Artificial intelligence for long-term robot autonomy: a survey,” *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 4023–4030, 2018.

[202] C. Street, B. Lacerda, M. Mühlig, and N. Hawes, “Right place, tight time: proactive multi-robot task allocation under spatiotemporal uncertainty,” *Journal of Artificial Intelligence Research*, vol. 79, pp. 137–171, 2024.

[203] M. Carwehl, C. Imrie, T. Vogel, G. Rodrigues, R. Calinescu, and L. Grunske, “Formal synthesis of uncertainty reduction controllers,” *Software Engineering for Self-Adaptive Systems*, 2024.

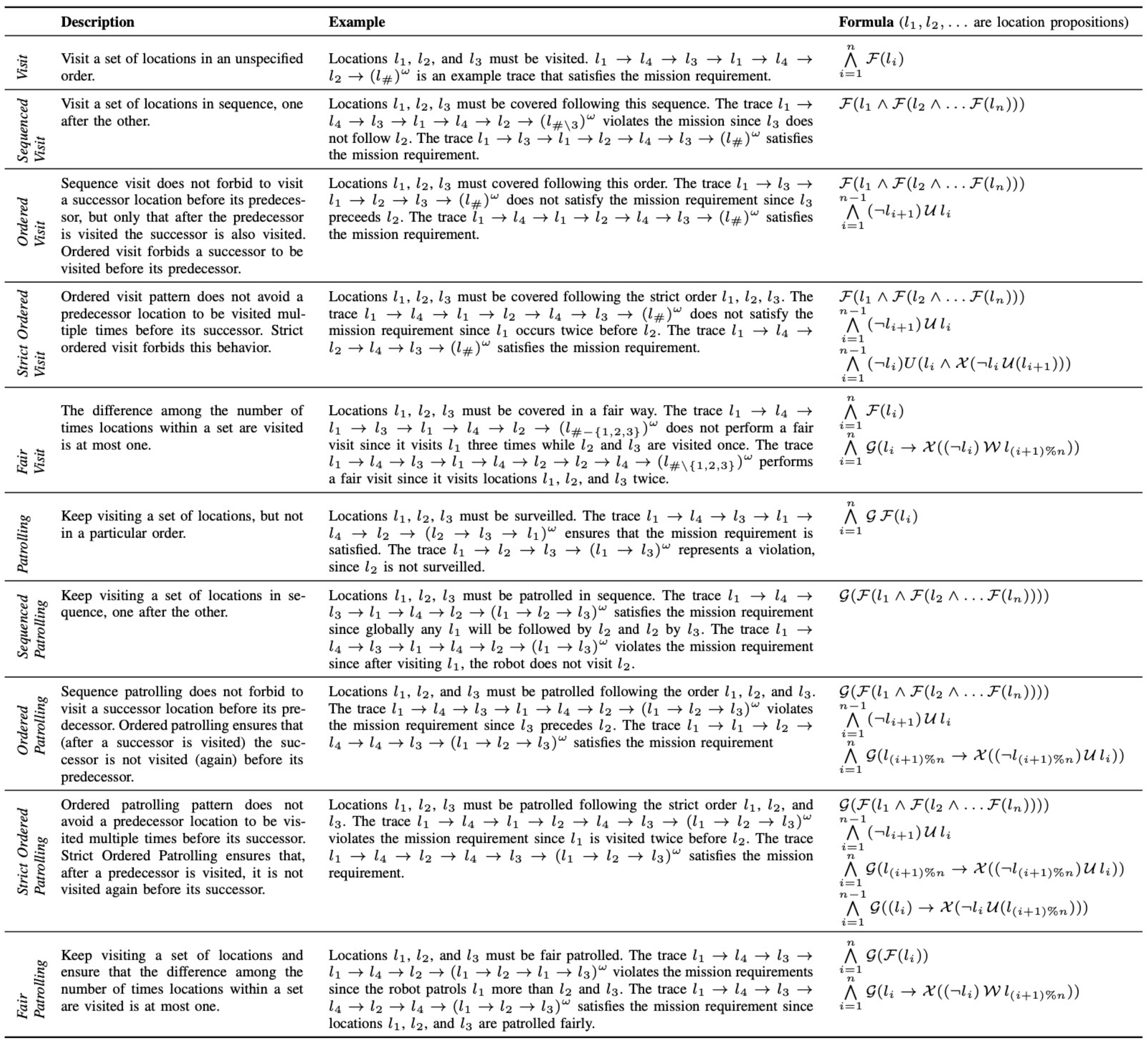
[204] D. G. D. L. Iglesia and D. Weyns, “MAPE-K formal templates to rigorously design behaviors for self-adaptive systems,” *ACM Transactions on Autonomous and Adaptive Systems*, vol. 10, no. 3, pp. 1–31, 2015.

[205] B. Yao, W. Xu, T. Shen, X. Ye, and S. Tian, “Digital twin-based multi-level task rescheduling for robotic assembly line,” *Scientific Reports*, vol. 13, no. 1, p. 1769, 2023.

APPENDIX

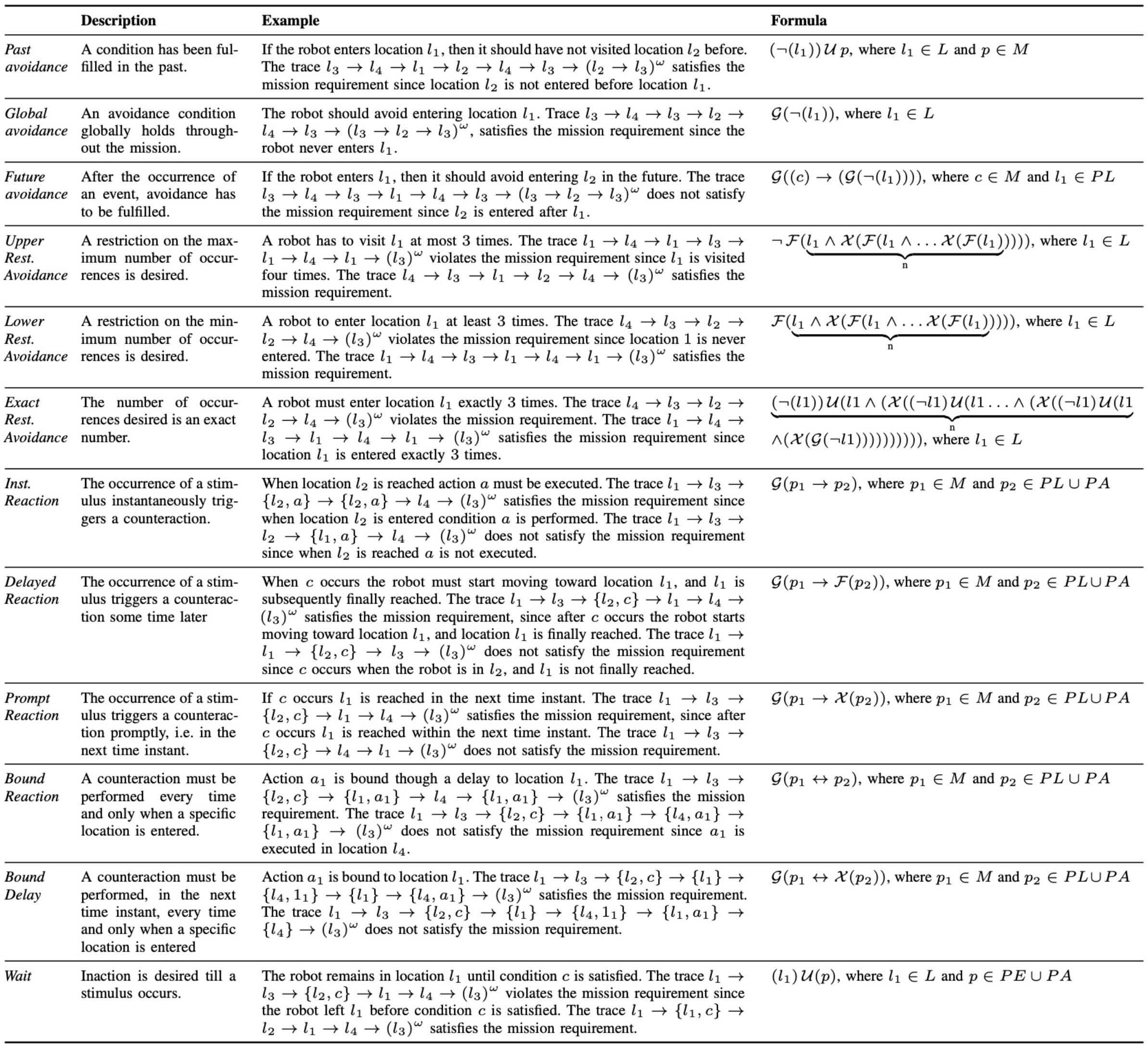
A Non-quantitative specification patterns

Menghi et al. [2] proposed a set of 22 non-quantitative patterns presented in Figure 3.8a. The translation into LTL and CTL formulae for each of these patterns is presented in the self-explanatory Figures 8.1 and 8.2. For example, the first row depicts the *visit* pattern. It describes a pattern where a set of locations in any order are visited by a robot. An example of this pattern is to visit locations $l1, l2$ and $l3$, and an example of a path that fulfills this pattern is given by $l1 \rightarrow l4 \rightarrow l3 \rightarrow l1 \rightarrow l4 \rightarrow l2 \rightarrow (l_\#)^w$, where $l_\#$ is any location and $(l_\#)^w$ indicates an infinite sequence of locations. For the complete notation of symbols in the translated formulae we refer the reader to [2].



	Description	**Example**	**Formula** ($l_1, l_2, \ldots$ are location propositions)
Visit	Visit a set of locations in an unspecified order.	Locations l_1, l_2, and l_3 must be visited. $l_1 \rightarrow l_4 \rightarrow l_3 \rightarrow l_1 \rightarrow l_4 \rightarrow l_2 \rightarrow (l_\#)^\omega$ is an example trace that satisfies the mission requirement.	$\bigwedge_{i=1}^{n} \mathcal{F}(l_i)$
Sequenced Visit	Visit a set of locations in sequence, one after the other.	Locations l_1, l_2, l_3 must be covered following this sequence. The trace $l_1 \rightarrow l_4 \rightarrow l_3 \rightarrow l_1 \rightarrow l_4 \rightarrow l_2 \rightarrow (l_{\#\backslash 3})^\omega$ violates the mission since l_3 does not follow l_2. The trace $l_1 \rightarrow l_3 \rightarrow l_1 \rightarrow l_2 \rightarrow l_4 \rightarrow l_3 \rightarrow (l_\#)^\omega$ satisfies the mission requirement.	$\mathcal{F}(l_1 \wedge \mathcal{F}(l_2 \wedge \ldots \mathcal{F}(l_n)))$
Ordered Visit	Sequence visit does not forbid to visit a successor location before its predecessor, but only that after the predecessor is visited the successor is also visited. Ordered visit forbids a successor to be visited before its predecessor.	Locations l_1, l_2, l_3 must covered following this order. The trace $l_1 \rightarrow l_3 \rightarrow l_1 \rightarrow l_2 \rightarrow l_3 \rightarrow (l_\#)^\omega$ does not satisfy the mission requirement since l_3 preceeds l_2. The trace $l_1 \rightarrow l_4 \rightarrow l_1 \rightarrow l_2 \rightarrow l_4 \rightarrow l_3 \rightarrow (l_\#)^\omega$ satisfies the mission requirement.	$\mathcal{F}(l_1 \wedge \mathcal{F}(l_2 \wedge \ldots \mathcal{F}(l_n)))$ $\bigwedge_{i=1}^{n-1} (\neg l_{i+1})\,\mathcal{U}\,l_i$
Strict Ordered Visit	Ordered visit pattern does not avoid a predecessor location to be visited multiple times before its successor. Strict ordered visit forbids this behavior.	Locations l_1, l_2, l_3 must be covered following the strict order l_1, l_2, l_3. The trace $l_1 \rightarrow l_4 \rightarrow l_1 \rightarrow l_2 \rightarrow l_4 \rightarrow l_3 \rightarrow (l_\#)^\omega$ does not satisfy the mission requirement since l_1 occurs twice before l_2. The trace $l_1 \rightarrow l_4 \rightarrow l_2 \rightarrow l_4 \rightarrow l_3 \rightarrow (l_\#)^\omega$ satisfies the mission requirement.	$\mathcal{F}(l_1 \wedge \mathcal{F}(l_2 \wedge \ldots \mathcal{F}(l_n)))$ $\bigwedge_{i=1}^{n-1} (\neg l_{i+1})\,\mathcal{U}\,l_i$ $\bigwedge_{i=1}^{n-1} (\neg l_i)U(l_i \wedge \mathcal{X}(\neg l_i\,\mathcal{U}(l_{i+1})))$
Fair Visit	The difference among the number of times locations within a set are visited is at most one.	Locations l_1, l_2, l_3 must be covered in a fair way. The trace $l_1 \rightarrow l_4 \rightarrow l_1 \rightarrow l_3 \rightarrow l_1 \rightarrow l_4 \rightarrow l_2 \rightarrow (l_{\#-\{1,2,3\}})^\omega$ does not perform a fair visit since it visits l_1 three times while l_2 and l_3 are visited once. The trace $l_1 \rightarrow l_4 \rightarrow l_3 \rightarrow l_1 \rightarrow l_4 \rightarrow l_2 \rightarrow l_2 \rightarrow l_4 \rightarrow (l_{\#\backslash\{1,2,3\}})^\omega$ performs a fair visit since it visits locations l_1, l_2, and l_3 twice.	$\bigwedge_{i=1}^{n} \mathcal{F}(l_i)$ $\bigwedge_{i=1}^{n} \mathcal{G}(l_i \rightarrow \mathcal{X}((\neg l_i)\,\mathcal{W}\,l_{(i+1)\%n}))$
Patrolling	Keep visiting a set of locations, but not in a particular order.	Locations l_1, l_2, l_3 must be surveilled. The trace $l_1 \rightarrow l_4 \rightarrow l_3 \rightarrow l_1 \rightarrow l_4 \rightarrow l_2 \rightarrow (l_2 \rightarrow l_3 \rightarrow l_1)^\omega$ ensures that the mission requirement is satisfied. The trace $l_1 \rightarrow l_2 \rightarrow l_3 \rightarrow (l_1 \rightarrow l_3)^\omega$ represents a violation, since l_2 is not surveilled.	$\bigwedge_{i=1}^{n} \mathcal{G}\,\mathcal{F}(l_i)$
Sequenced Patrolling	Keep visiting a set of locations in sequence, one after the other.	Locations l_1, l_2, l_3 must be patrolled in sequence. The trace $l_1 \rightarrow l_4 \rightarrow l_3 \rightarrow l_1 \rightarrow l_4 \rightarrow l_2 \rightarrow (l_1 \rightarrow l_2 \rightarrow l_3)^\omega$ satisfies the mission requirement since globally any l_1 will be followed by l_2 and l_2 by l_3. The trace $l_1 \rightarrow l_4 \rightarrow l_3 \rightarrow l_1 \rightarrow l_4 \rightarrow l_2 \rightarrow (l_1 \rightarrow l_3)^\omega$ violates the mission requirement since after visiting l_1, the robot does not visit l_2.	$\mathcal{G}(\mathcal{F}(l_1 \wedge \mathcal{F}(l_2 \wedge \ldots \mathcal{F}(l_n))))$
Ordered Patrolling	Sequence patrolling does not forbid to visit a successor location before its predecessor. Ordered patrolling ensures that (after a successor is visited) the successor is not visited (again) before its predecessor.	Locations l_1, l_2, and l_3 must be patrolled following the order l_1, l_2, and l_3. The trace $l_1 \rightarrow l_4 \rightarrow l_3 \rightarrow l_1 \rightarrow l_4 \rightarrow l_2 \rightarrow (l_1 \rightarrow l_2 \rightarrow l_3)^\omega$ violates the mission requirement since l_3 precedes l_2. The trace $l_1 \rightarrow l_1 \rightarrow l_2 \rightarrow l_4 \rightarrow l_4 \rightarrow l_3 \rightarrow (l_1 \rightarrow l_2 \rightarrow l_3)^\omega$ satisfies the mission requirement	$\mathcal{G}(\mathcal{F}(l_1 \wedge \mathcal{F}(l_2 \wedge \ldots \mathcal{F}(l_n))))$ $\bigwedge_{i=1}^{n-1} (\neg l_{i+1})\,\mathcal{U}\,l_i$ $\bigwedge_{i=1}^{n} \mathcal{G}(l_{(i+1)\%n} \rightarrow \mathcal{X}((\neg l_{(i+1)\%n})\,\mathcal{U}\,l_i))$
Strict Ordered Patrolling	Ordered patrolling pattern does not avoid a predecessor location to be visited multiple times before its successor. Strict Ordered Patrolling ensures that, after a predecessor is visited, it is not visited again before its successor.	Locations l_1, l_2, l_3 must be patrolled following the strict order l_1, l_2, and l_3. The trace $l_1 \rightarrow l_4 \rightarrow l_1 \rightarrow l_2 \rightarrow l_4 \rightarrow l_3 \rightarrow (l_1 \rightarrow l_2 \rightarrow l_3)^\omega$ violates the mission requirement since l_1 is visited twice before l_2. The trace $l_1 \rightarrow l_4 \rightarrow l_2 \rightarrow l_4 \rightarrow l_3 \rightarrow (l_1 \rightarrow l_2 \rightarrow l_3)^\omega$ satisfies the mission requirement.	$\mathcal{G}(\mathcal{F}(l_1 \wedge \mathcal{F}(l_2 \wedge \ldots \mathcal{F}(l_n))))$ $\bigwedge_{i=1}^{n-1} (\neg l_{i+1})\,\mathcal{U}\,l_i$ $\bigwedge_{i=1}^{n} \mathcal{G}(l_{(i+1)\%n} \rightarrow \mathcal{X}((\neg l_{(i+1)\%n})\,\mathcal{U}\,l_i))$ $\bigwedge_{i=1}^{n-1} \mathcal{G}((l_i) \rightarrow \mathcal{X}(\neg l_i\,\mathcal{U}(l_{(i+1)\%n})))$
Fair Patrolling	Keep visiting a set of locations and ensure that the difference among the number of times locations within a set are visited is at most one.	Locations l_1, l_2, and l_3 must be fair patrolled. The trace $l_1 \rightarrow l_4 \rightarrow l_3 \rightarrow l_1 \rightarrow l_4 \rightarrow l_2 \rightarrow (l_1 \rightarrow l_2 \rightarrow l_1 \rightarrow l_3)^\omega$ violates the mission requirements since the robot patrols l_1 more than l_2 and l_3. The trace $l_1 \rightarrow l_4 \rightarrow l_3 \rightarrow l_4 \rightarrow l_2 \rightarrow l_4 \rightarrow (l_1 \rightarrow l_2 \rightarrow l_3)^\omega$ satisfies the mission requirement since locations l_1, l_2, and l_3 are patrolled fairly.	$\bigwedge_{i=1}^{n} \mathcal{G}(\mathcal{F}(l_i))$ $\bigwedge_{i=1}^{n} \mathcal{G}(l_i \rightarrow \mathcal{X}((\neg l_i)\,\mathcal{W}\,l_{(i+1)\%n}))$

Figure 8.1: Core movement robot patterns, from [2].



	Description	Example	Formula
Past avoidance	A condition has been fulfilled in the past.	If the robot enters location l_1, then it should have not visited location l_2 before. The trace $l_3 \rightarrow l_4 \rightarrow l_1 \rightarrow l_2 \rightarrow l_4 \rightarrow l_3 \rightarrow (l_2 \rightarrow l_3)^\omega$ satisfies the mission requirement since location l_2 is not entered before location l_1.	$(\neg(l_1))\,\mathcal{U}\,p$, where $l_1 \in L$ and $p \in M$
Global avoidance	An avoidance condition globally holds throughout the mission.	The robot should avoid entering location l_1. Trace $l_3 \rightarrow l_4 \rightarrow l_3 \rightarrow l_2 \rightarrow l_4 \rightarrow l_3 \rightarrow (l_3 \rightarrow l_2 \rightarrow l_3)^\omega$, satisfies the mission requirement since the robot never enters l_1.	$\mathcal{G}(\neg(l_1))$, where $l_1 \in L$
Future avoidance	After the occurrence of an event, avoidance has to be fulfilled.	If the robot enters l_1, then it should avoid entering l_2 in the future. The trace $l_3 \rightarrow l_4 \rightarrow l_3 \rightarrow l_1 \rightarrow l_4 \rightarrow l_3 \rightarrow (l_3 \rightarrow l_2 \rightarrow l_3)^\omega$ does not satisfy the mission requirement since l_2 is entered after l_1.	$\mathcal{G}((c) \rightarrow (\mathcal{G}(\neg(l_1))))$, where $c \in M$ and $l_1 \in PL$
Upper Rest. Avoidance	A restriction on the maximum number of occurrences is desired.	A robot has to visit l_1 at most 3 times. The trace $l_1 \rightarrow l_4 \rightarrow l_1 \rightarrow l_3 \rightarrow l_1 \rightarrow l_4 \rightarrow l_1 \rightarrow (l_3)^\omega$ violates the mission requirement since l_1 is visited four times. The trace $l_4 \rightarrow l_3 \rightarrow l_1 \rightarrow l_2 \rightarrow l_4 \rightarrow (l_3)^\omega$ satisfies the mission requirement.	$\neg\,\mathcal{F}(\underbrace{l_1 \wedge \mathcal{X}(\mathcal{F}(l_1 \wedge \ldots \mathcal{X}(\mathcal{F}(l_1)}_{\text{n}})))))$, where $l_1 \in L$
Lower Rest. Avoidance	A restriction on the minimum number of occurrences is desired.	A robot to enter location l_1 at least 3 times. The trace $l_4 \rightarrow l_3 \rightarrow l_2 \rightarrow l_2 \rightarrow l_4 \rightarrow (l_3)^\omega$ violates the mission requirement since location 1 is never entered. The trace $l_1 \rightarrow l_4 \rightarrow l_3 \rightarrow l_1 \rightarrow l_4 \rightarrow l_1 \rightarrow (l_3)^\omega$ satisfies the mission requirement.	$\mathcal{F}(\underbrace{l_1 \wedge \mathcal{X}(\mathcal{F}(l_1 \wedge \ldots \mathcal{X}(\mathcal{F}(l_1)}_{\text{n}})))))$, where $l_1 \in L$
Exact Rest. Avoidance	The number of occurrences desired is an exact number.	A robot must enter location l_1 exactly 3 times. The trace $l_4 \rightarrow l_3 \rightarrow l_2 \rightarrow l_2 \rightarrow l_4 \rightarrow (l_3)^\omega$ violates the mission requirement. The trace $l_1 \rightarrow l_4 \rightarrow l_3 \rightarrow l_1 \rightarrow l_4 \rightarrow l_1 \rightarrow (l_3)^\omega$ satisfies the mission requirement since location l_1 is entered exactly 3 times.	$\underbrace{(\neg(l1))\,\mathcal{U}(l1 \wedge (\mathcal{X}((\neg l1)\,\mathcal{U}(l1 \ldots \wedge (\mathcal{X}((\neg l1)\,\mathcal{U}(l1}_{\text{n}}$ $\wedge(\mathcal{X}(\mathcal{G}(\neg l1))))))))))$, where $l_1 \in L$
Inst. Reaction	The occurrence of a stimulus instantaneously triggers a counteraction.	When location l_2 is reached action a must be executed. The trace $l_1 \rightarrow l_3 \rightarrow \{l_2, a\} \rightarrow \{l_2, a\} \rightarrow l_4 \rightarrow (l_3)^\omega$ satisfies the mission requirement since when location l_2 is entered condition a is performed. The trace $l_1 \rightarrow l_3 \rightarrow l_2 \rightarrow \{l_1, a\} \rightarrow l_4 \rightarrow (l_3)^\omega$ does not satisfy the mission requirement since when l_2 is reached a is not executed.	$\mathcal{G}(p_1 \rightarrow p_2)$, where $p_1 \in M$ and $p_2 \in PL \cup PA$
Delayed Reaction	The occurrence of a stimulus triggers a counteraction some time later	When c occurs the robot must start moving toward location l_1, and l_1 is subsequently finally reached. The trace $l_1 \rightarrow l_3 \rightarrow \{l_2, c\} \rightarrow l_1 \rightarrow l_4 \rightarrow (l_3)^\omega$ satisfies the mission requirement, since after c occurs the robot starts moving toward location l_1, and location l_1 is finally reached. The trace $l_1 \rightarrow l_1 \rightarrow \{l_2, c\} \rightarrow l_3 \rightarrow (l_3)^\omega$ does not satisfy the mission requirement since c occurs when the robot is in l_2, and l_1 is not finally reached.	$\mathcal{G}(p_1 \rightarrow \mathcal{F}(p_2))$, where $p_1 \in M$ and $p_2 \in PL \cup PA$
Prompt Reaction	The occurrence of a stimulus triggers a counteraction promptly, i.e. in the next time instant.	If c occurs l_1 is reached in the next time instant. The trace $l_1 \rightarrow l_3 \rightarrow \{l_2, c\} \rightarrow l_1 \rightarrow l_4 \rightarrow (l_3)^\omega$ satisfies the mission requirement, since after c occurs l_1 is reached within the next time instant. The trace $l_1 \rightarrow l_3 \rightarrow \{l_2, c\} \rightarrow l_4 \rightarrow l_1 \rightarrow (l_3)^\omega$ does not satisfy the mission requirement.	$\mathcal{G}(p_1 \rightarrow \mathcal{X}(p_2))$, where $p_1 \in M$ and $p_2 \in PL \cup PA$
Bound Reaction	A counteraction must be performed every time and only when a specific location is entered.	Action a_1 is bound though a delay to location l_1. The trace $l_1 \rightarrow l_3 \rightarrow \{l_2, c\} \rightarrow \{l_1, a_1\} \rightarrow l_4 \rightarrow \{l_1, a_1\} \rightarrow (l_3)^\omega$ satisfies the mission requirement. The trace $l_1 \rightarrow l_3 \rightarrow \{l_2, c\} \rightarrow \{l_1, a_1\} \rightarrow \{l_4, a_1\} \rightarrow \{l_1, a_1\} \rightarrow (l_3)^\omega$ does not satisfy the mission requirement since a_1 is executed in location l_4.	$\mathcal{G}(p_1 \leftrightarrow p_2)$, where $p_1 \in M$ and $p_2 \in PL \cup PA$
Bound Delay	A counteraction must be performed, in the next time instant, every time and only when a specific location is entered	Action a_1 is bound to location l_1. The trace $l_1 \rightarrow l_3 \rightarrow \{l_2, c\} \rightarrow \{l_1\} \rightarrow \{l_4, 1_1\} \rightarrow \{l_1\} \rightarrow \{l_4, a_1\} \rightarrow (l_3)^\omega$ satisfies the mission requirement. The trace $l_1 \rightarrow l_3 \rightarrow \{l_2, c\} \rightarrow \{l_1\} \rightarrow \{l_4, 1_1\} \rightarrow \{l_1, a_1\} \rightarrow \{l_4\} \rightarrow (l_3)^\omega$ does not satisfy the mission requirement.	$\mathcal{G}(p_1 \leftrightarrow \mathcal{X}(p_2))$, where $p_1 \in M$ and $p_2 \in PL \cup PA$
Wait	Inaction is desired till a stimulus occurs.	The robot remains in location l_1 until condition c is satisfied. The trace $l_1 \rightarrow l_3 \rightarrow \{l_2, c\} \rightarrow l_1 \rightarrow l_4 \rightarrow (l_3)^\omega$ violates the mission requirement since the robot left l_1 before condition c is satisfied. The trace $l_1 \rightarrow \{l_1, c\} \rightarrow l_2 \rightarrow l_1 \rightarrow l_4 \rightarrow (l_3)^\omega$ satisfies the mission requirement.	$(l_1)\,\mathcal{U}(p)$, where $l_1 \in L$ and $p \in PE \cup PA$

Figure 8.2: Avoidance and Trigger patterns, from [2].

B Example of the task model

The following example shows the Task Model consisting of 4 atomic tasks and 7 compound tasks; and the Mission *m*1 specified in the KANOA DSL for the example presented in Section 6.2.

```
TaskModel:
at1 : 2 robots needed at location l1,
at2 : 1 robot needed at location l1,
at3: 1 robot needed at location l1,
at4: 1 robot needed at location l1,
ct1: subtasks [ct2,ct3],
ct2: subtasks [ct4,ct5] constraint:ordered,
ct4: subtasks [at1,at2],
ct5: subtasks [ct7,at1],
ct7: subtasks [at3,at4] constraint:ordered,
ct3: subtasks [ct6,ct5],
ct6: subtasks [at1,at4] constraint:ordered
RobotsModel: ...
Mission :
m1 : compound task ct1
```

C Alloy facts in the task allocation model

This section is a continuation of Section 5.1. This section shows the effects of removing any of the *facts* in the Alloy model related to the capabilities. The example consists of three robots, $r1, r2$ and $r3$. Robot $r1$ has two capabilities, $c1, c11$, robot $r2$ capabilities $c2$, $c22$, and robot $r3$ capabilities $c3, c33$. Atomic task $at1$ can be done by capabilities $c1$ or $c3$. Atomic task $at2$ can be done by capabilities $c11$ or $c33$. Atomic task $at3$ can be done by capability $c2$ and atomic task $at4$ by $c22$.

The complete Alloy model is depicted in Figure 8.4, which shows an example of an allocation problem, with the three *facts* related to capabilities inside a red rectangle. The effects of removing (b),(c) or (d) are depicted in Figure 8.3 matching these labels. Figure 8.3.a shows a possible allocation solution without removing any fact.

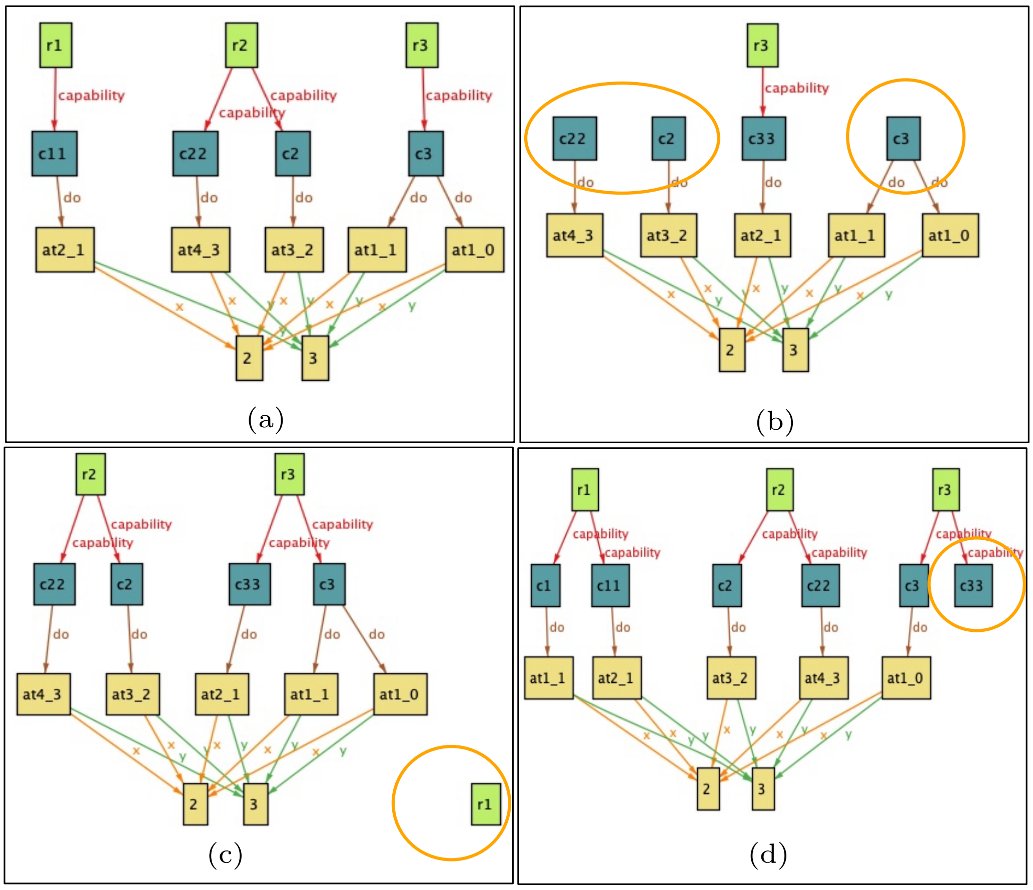
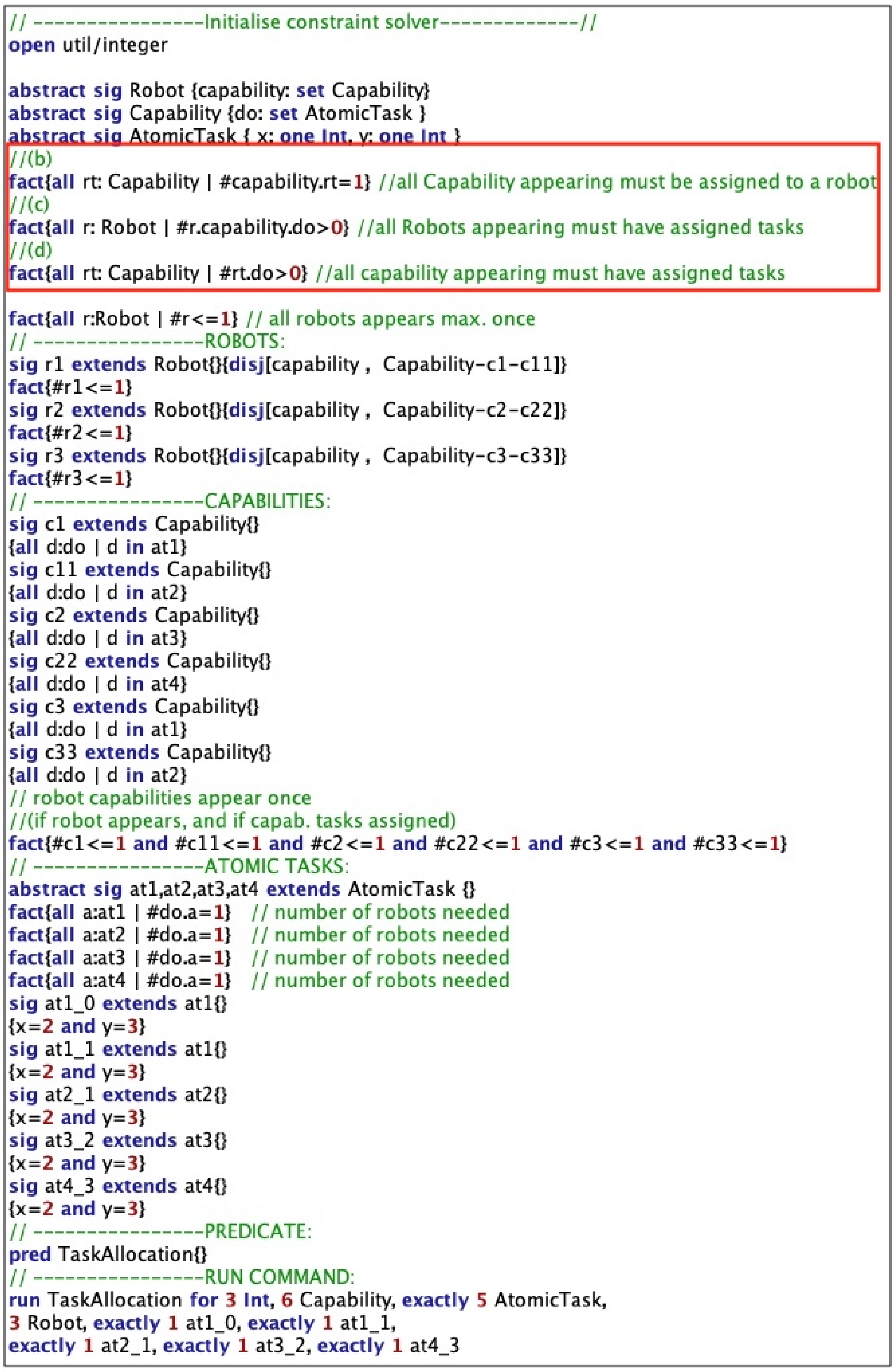


Figure 8.3: Effect of removing Alloy *facts* on the solutions to the allocation problem. This visualisation of solutions in the Alloy analyzer is explained in Section 5.3.2)

Figure 8.3.b shows that by removing *fact*{*all rt:Capability*| *#capability.rt=1* }, some of the capabilities may appear without a relation to a robot. Figure 8.3.c shows that by removing fact (c), some robots may appear without any capability or task assigned. Figure 8.3.c shows that by removing fact (d), some capabilities may appear without any allocated task.



```
// ----------------Initialise constraint solver-------------//
open util/integer

abstract sig Robot {capability: set Capability}
abstract sig Capability {do: set AtomicTask }
abstract sig AtomicTask { x: one Int, y: one Int }
//(b)
fact{all rt: Capability | #capability.rt=1} //all Capability appearing must be assigned to a robot
//(c)
fact{all r: Robot | #r.capability.do>0} //all Robots appearing must have assigned tasks
//(d)
fact{all rt: Capability | #rt.do>0} //all capability appearing must have assigned tasks

fact{all r:Robot | #r<=1} // all robots appears max. once
// ----------------ROBOTS:
sig r1 extends Robot{}{disj[capability , Capability-c1-c11]}
fact{#r1<=1}
sig r2 extends Robot{}{disj[capability , Capability-c2-c22]}
fact{#r2<=1}
sig r3 extends Robot{}{disj[capability , Capability-c3-c33]}
fact{#r3<=1}
// ----------------CAPABILITIES:
sig c1 extends Capability{}
{all d:do | d in at1}
sig c11 extends Capability{}
{all d:do | d in at2}
sig c2 extends Capability{}
{all d:do | d in at3}
sig c22 extends Capability{}
{all d:do | d in at4}
sig c3 extends Capability{}
{all d:do | d in at1}
sig c33 extends Capability{}
{all d:do | d in at2}
// robot capabilities appear once
//(if robot appears, and if capab. tasks assigned)
fact{#c1<=1 and #c11<=1 and #c2<=1 and #c22<=1 and #c3<=1 and #c33<=1}
// ----------------ATOMIC TASKS:
abstract sig at1,at2,at3,at4 extends AtomicTask {}
fact{all a:at1 | #do.a=1}  // number of robots needed
fact{all a:at2 | #do.a=1}  // number of robots needed
fact{all a:at3 | #do.a=1}  // number of robots needed
fact{all a:at4 | #do.a=1}  // number of robots needed
sig at1_0 extends at1{}
{x=2 and y=3}
sig at1_1 extends at1{}
{x=2 and y=3}
sig at2_1 extends at2{}
{x=2 and y=3}
sig at3_2 extends at3{}
{x=2 and y=3}
sig at4_3 extends at4{}
{x=2 and y=3}
// ----------------PREDICATE:
pred TaskAllocation{}
// ----------------RUN COMMAND:
run TaskAllocation for 3 Int, 6 Capability, exactly 5 AtomicTask,
3 Robot, exactly 1 at1_0, exactly 1 at1_1,
exactly 1 at2_1, exactly 1 at3_2, exactly 1 at4_3
```

Figure 8.4: Example of an allocation problem in Alloy.

D Compound Task Constraints to Atomic Tasks

This appendix complements the information provided in Section 6.2.2. Here, we remind the reader what an ordered and consecutive compound task constraint means, intuitively. **Ordered tasks.** When we talk about a **constraint** of a compound task, the constraint is applied to its subtasks. For an *ordered* compound task with subtasks ($t1, t2, t3$,...), intuitively, we allow other actions to happen between the subtasks but their order must be preserved[1],

$$\text{ordered tasks}[t1, t2, t3, ...] : t1 + \underbrace{\quad ... \quad}_{\text{idle and other tasks}} + t2 + \underbrace{\quad ... \quad}_{\text{idle and other tasks}} + t3 + ...$$

The solution of applying an order constraint to the list of subtasks of a compound task ct_i is a set of action sequences, where each sequence captures the ordering of the subtasks, other atomic tasks, and idle time slots. For example, a possible set of solutions S_{ord} for ordering task t1 to t4:

$$\begin{aligned} S_{ord}([t1, t2, t3, t4]) = \{&[t1, t2, t3, t4],\\ &[t1, idle, t2, t3, t4],\\ &[idle, t1, idle, t2, t3, idle, idle, t4]...\} \end{aligned}$$

Consecutive tasks. For *consecutive* compound tasks, every subtask must start *just* after the previous one is finished,

$$\text{consecutive tasks [t1,t2,t3,...]} : t1 \quad + \quad t2 \quad + \quad t3 \quad + \quad ...$$

Hence, there is only **one** solution s_{con} to the consecutive task constraint[2],

$$s_{consec} = [t1, t2, t3, t4]$$

as $[t1, t2, t3, t4] \in \{[t1, t2, t3, t4], [t1, idle, t2, t3, t4],...\}$, we propose the next condition.

Well-Formedness Condition. *Let $S_{ord}(T)$ be the set of solutions to ordering the sequence of tasks T=t1,t2,...,tn. Let $s_{con}(T)$ be the solution to consecutively order T, then $s_{con}(T) \in S_{ord}(T)$. In other words, "if an atomic task at_i has to be done* ***consecutively*** *after at_{i-1} and* ***ordered*** *after at_{i-1}, we remove the ordering constraint and apply only the consecutive constraint".*

[1]We are only reasoning about the start and end time of tasks. If t1 and t2 are done in different locations, two robots must be needed so that the robot at t2 is ready to start immediately after the robot at t1 finishes this task.

[2]Although it may seem trivial to consider consecutive tasks, robots require time to travel between locations. Hence, a consecutive task may require multiple robots synchronising at different locations. In other words, solving the sequencing of consecutive tasks is trivial, but not the robot allocation and scheduling of consecutive tasks.

This conclusion helped us solve the conflicts arising for atomic tasks that share multiple compound task constraints: a consecutive constraint cancels an ordering constraint.

Observation 10. **Constraint conflicts**. Consider an consecutive task ct1=[at1 at2 ct2], where ct2=[at3,at4] and at1,...at4 are atomic tasks. As designers, we must decide what does "consecutiveness" of ct1 subtasks (at1, at2, and ct2) mean. a) We may decide that all reachable atomic tasks are ordered, i.e., they must be performed as at1→at2→at3→at4. b) Another design solution is to constrain all atomic tasks of subtasks that are compound to be done immediately, i.e., at1→at2→{at3 and at4 start at the same time}. c) A third design may be to start with any task of ct2 after at1 and at2, leading to two solutions: 1) at1→at2→at3→at4 and 2) at1→at2→at4→at3. A fourth solution may be to start any task of ct2, then allow other tasks from the whole mission to happen and, at some point, complete ct2. We opt for solution a) for three reasons. First, in a multi-robot system, we expect to have consecutive tasks such as, "first clean room 1 and immediately after go to room 2 and immediately after clean room 2" (see, for example, the robotic mission patterns [25]). Second, it avoids confusion when the plans of the robots are synthesised and the user may expect tasks under a "*consecutive*" constraint to be done "*consecutively*"; and third, it is a tradeoff between scalability and solving the consecutiveness of atomic tasks, as avoiding permutations arising from solution (c) increases the size of the model.
.

We force atomic tasks to be performed consecutively when they are constrained by both (order and consecutive) constraints. In Section 6.2.2, we define new atomic constraints in terms of two additional functions, *doneBefore*(at_i) and *justDone*(at_i). These describe the tasks that have to be done before and the ones that must finished at the same as the new one starts, respectively (see Section 6.2.2).

We force atomic tasks to be performed consecutively when they are constrained by both (order and consecutive) constraints. In Section 6.2.2, we define new atomic constraints in terms of two additional functions, *doneBefore*(at_i) and *justDone*(at_i). These describe the tasks that have to be done before and the ones that must finished at the same as the new one starts, respectively (see Section 6.2.2).

E MDP model with idling-time limit assessment

In this Section, we further justify the decision to add a limit to the time robots can spend idling and demonstrate the reduction in the size of the model's transitions and states with an example.

Consider a PRISM model with a time limit of `TT` to complete four tasks allocated to two robots, r1 and r2. Robot r1 has tasks r1t1 and r1t2. Robot r2 has tasks r2t3 and r2t4. The travelling time for robot r_i from an initial location l0 to the location where t_j has to be done is denoted as $\mathrm{travelr}_i\mathrm{l0t}_j$. The travelling time between tasks' locations t_j and t_k is denoted as $\mathrm{travelr}_i\mathrm{t}_j\mathrm{t}_k$. The time to complete task t_j is denoted as $\mathrm{r}_i\mathrm{t}_j\mathrm{Time}$. The robots have the option to perform a task or idle for a time unit. Intuitively, if we limit the number of times that the robot can enter an idling state, the number of transitions and states of the MDP are reduced, which increases the scalability of our model. Hence, we decided to explicitly add to the model the maximum number of times that a robot can idle. For each robot, we limit the idle time to be the total time available, `TT`, minus the time spent travelling between locations and performing tasks. For example, for robot r1,

```
maxIdler1=TT-r1t1Time-r1t2Time-travelr1l0t1-travelr1t1t2;
```
(8.1)

where `r1t1Time` and `r1t2Time` are the time that r1 needs to perform t1 and t2; and `travelr1l0t1` and travelr1t1t2 the time travelling between locations.

We add a state variable for each robot (e.g., for robot r1, `r1idleTime`) to track the number of times that a robot has entered idle. We strengthen the idle transition by allowing the robot to idle only `maxIdler1` times,

```
[r1idle] r1order!=2 & (r1time+1<=TT) & (r1idleTime+1<=maxIdler1)
            -> (r1time'=r1time+1) & (r1idleTime'=r1idleTime+1);
```
(8.2)

An **special case** happens when the total time `TT` is equal to the sum of the time spend in travelling and doing tasks. In this case, we had the issue of a 0 range, which is marked as an error by Prism,

```
r1idleTime:[0..0];
```
(8.3)

Intuitively, this means that the robot only has time to travel and complete the tasks with the time available, and no time to idle. Hence, we remove the idling transition (equation 8.2) and the idle time modelled in the "idle" reward structure, for any robot r_i that has an $\texttt{maxIdler}_i$=0.

Observation 11. If a robot has an $\texttt{maxIdler}_i$<0, there is not enough time to complete all tasks (and trips between tasks) within the time available. This is computed before creating the MDP model so the schedule (aka. permutation of tasks) can be discarded.

We emphasize that the reduction of the model when a constraint in the idling time is added to the original model is possible as the idling transitions are reduced by `maxIdler1<=TT`.

Evaluation. We created a PRISM model without limiting the idling time (see Listing 8.1) and a second where we limit each robot's number of times that a robot can enter idling (see Listing 8.2). We assume a travelling time between any two locations and a task completion time equal to `time4TasknTravel`, to simplify the comparison between tests. Table 8.1 shows the number of states and transitions varying the values of `TT` and `time4TasknTravel`.

In the case of `time4TasknTravel=1`, this means that each robot travels from its initial location to the next task in 1-time step. It spends 1-time step travelling to its first allocated task, 1-time step performing its first task, another time step travelling to the second task, and one more to perform the second task. In total, this requires *at least* 4 time units. If `time4TasknTravel=10` the mission would require 40 time units, and so on. We should the results with `time4TasknTravel=1` and 10 in Table 8.1. Table 8.1 shows the reduction of transitions and states when the idle time is limited, as expected.

If `maxIdler1` and `maxIdler2` result in negative numbers, this means that there is no possible plan where robots can complete all tasks in the time available and the permutation is discarded.

Table 8.1: MDP model with two robots and four tasks

		No idle time limit		**Idle time limit**	
Time to do task or travel (time4TasknTravel)	Total time (TT)	States	Transitions	States	Transitions
1	4	81	189	9	13
1	50	21609	59437	19881	54661
1	100	88209	243837	84681	234061
1	200	356409	987637	349281	967861
10	40	3969	10341	25	54
10	50	8649	22861	1089	2893
10	100	59049	160461	33489	92293
10	200	294849	810661	110889	306693

```
mdp

const int TT=200;
const int time4TasknTravel=10;

const int r1t1Time =time4TasknTravel;
const int r1t2Time= time4TasknTravel;
const int r2t3Time= time4TasknTravel;
const int r2t4Time= time4TasknTravel;
const int travelr1l0t1= time4TasknTravel;
const int travelr1t1t2= time4TasknTravel;
const int travelr2l0t3= time4TasknTravel;
const int travelr2t3t4= time4TasknTravel;

formula done =(r1order=2 &r2order=2);

module r1 //robot 1
 r1order:[0..2];
 r1time:[0..TT];
 []r1order=0 &(r1time+r1t1Time+travelr1l0t1<=TT) →(r1order'=1) &(r1time'=r1time+r1t1Time+travelr1l0t1);
 []r1order=1 &(r1time+r1t2Time+travelr1t1t2<=TT) →(r1order'=2) &(r1time'=r1time+r1t2Time+travelr1t1t2);
 [r1idle] r1order!=2 &(r1time+1<=TT) →(r1time'=r1time+1);
endmodule

module r2 //robot 2
 r2order:[0..2];
 r2time:[0..TT];
 []r2order=0 &(r2time+r2t3Time+travelr2l0t3<=TT) →(r2order'=1) &(r2time'=r2time+r2t3Time+travelr2l0t3);
 []r2order=1 &(r2time+r2t4Time+travelr2t3t4<=TT) →(r2order'=2) &(r2time'=r2time+r2t4Time+travelr2t3t4);
 [r2idle] r2order!=2 &(r2time+1<=TT) →(r2time'=r2time+1);
endmodule

rewards "idle"
 [r1idle] true: 1;
 [r2idle] true: 1;

endrewards
```

Listing 8.1: MDP model for two robots with two tasks

```
1  mdp
2
3  const int TT=200;
4  const int time4TasknTravel=10;
5  const int r1t1Time =time4TasknTravel;
6  const int r1t2Time= time4TasknTravel;
7  const int r2t3Time= time4TasknTravel;
8  const int r2t4Time= time4TasknTravel;
9  const int travelr1l0t1= time4TasknTravel;
10 const int travelr1t1t2= time4TasknTravel;
11 const int travelr2l0t3= time4TasknTravel;
12 const int travelr2t3t4= time4TasknTravel;
13
14 const int maxIdler1_v =TT-r1t1Time-r1t2Time-travelr1l0t1-travelr1t1t2;
15 const int maxIdler2_v =TT-r2t3Time-r2t4Time-travelr2l0t3-travelr2t3t4;
16
17 const int maxIdler1 =maxIdler1_v=0 ? 1 :maxIdler1_v;
18 const int maxIdler2 =maxIdler2_v=0 ? 1 :maxIdler2_v;
19
20 formula done =(r1order=2 &r2order=2);
21
22 module r1 //robot 1
23  r1order:[0..2];
24  r1time:[0..TT];
25  r1idleTime:[0..maxIdler1];
26  []r1order=0 &(r1time+r1t1Time+travelr1l0t1<=TT) →(r1order'=1) &(r1time'=r1time+r1t1Time+travelr1l0t1);
27  []r1order=1 &(r1time+r1t2Time+travelr1t1t2<=TT) →(r1order'=2) &(r1time'=r1time+r1t2Time+travelr1t1t2);
28  [r1idle] r1order!=2 &(r1time+1<=TT) &(r1idleTime+1<=maxIdler1) →(r1time'=r1time+1) & . . .
        (r1idleTime'=r1idleTime+1);
29 endmodule
30
31 module r2 /robot 2
32  r2order:[0..2];
33  r2time:[0..TT];
34  r2idleTime:[0..maxIdler2];
35  []r2order=0 &(r2time+r2t3Time+travelr2l0t3<=TT) →(r2order'=1) &(r2time'=r2time+r2t3Time+travelr2l0t3);
36  []r2order=1 &(r2time+r2t4Time+travelr2t3t4<=TT) →(r2order'=2) &(r2time'=r2time+r2t4Time+travelr2t3t4);
37  [r2idle] r2order!=2 &(r2time+1<=TT) &(r2idleTime+1<=maxIdler1) →(r2time'=r2time+1) & . . .
        (r2idleTime'=r2idleTime+1);
38 endmodule
39
40 rewards “idle“
41  [r1idle] true: 1;
42  [r2idle] true: 1;
43
44 endrewards
```

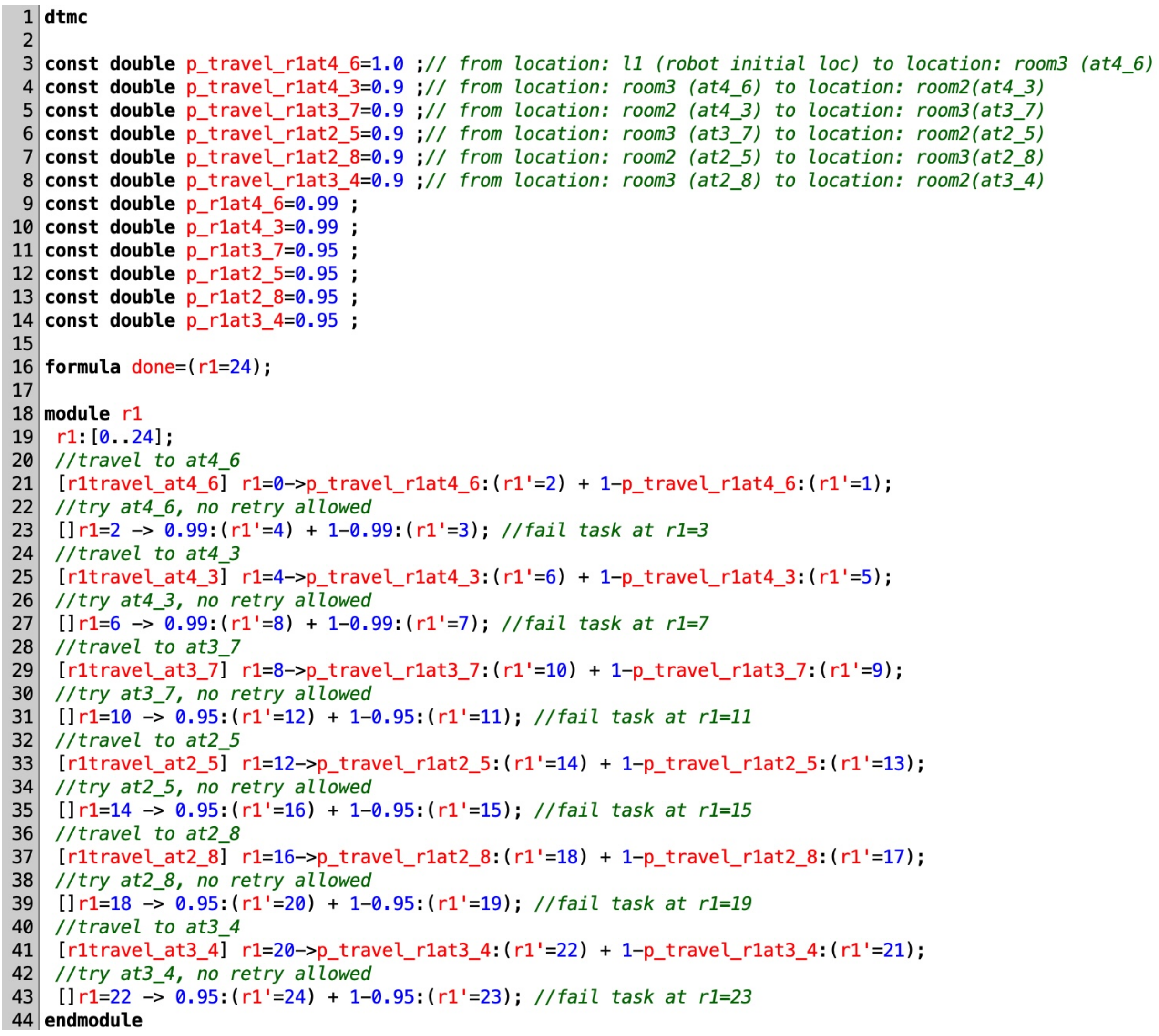
Listing 8.2: MDP model with limits on the idle time for each robot (lines 17-18).

```
mdp
const int TT=120;//total time available
//r1 velocity:1.0
const int travelr1at4_6=3 ;// from location: l1 (robot initial loc) to location: room3 (at4_6)    distance:3
const int travelr1at4_3=3 ;// from location: room3 (at4_6) to location: room2(at4_3)    distance:3
const int travelr1at3_7=3 ;// from location: room2 (at4_3) to location: room3(at3_7)    distance:3
const int travelr1at2_5=3 ;// from location: room3 (at3_7) to location: room2(at2_5)    distance:3
const int travelr1at2_8=3 ;// from location: room2 (at2_5) to location: room3(at2_8)    distance:3
const int travelr1at3_4=3 ;// from location: room3 (at2_8) to location: room2(at3_4)    distance:3
const int r1at4_6Time=1;
const int r1at4_3Time=1;
const int r1at3_7Time=5;
const int r1at2_5Time=5;
const int r1at2_8Time=5;
const int r1at3_4Time=5;
const int maxIdler1=20;

formula done=(r1order=6);

//formulae for ordered tasks
formula r1at4_6Done = r1order>=1 ;
formula r1at4_3Done = r1order>=2 ;

module r1
 r1order:[0..6];
 r1time:[0..120];
 r1idleTime:[0..maxIdler1];
 [r1at4_6] r1order=0 & (r1time+r1at4_6Time+travelr1at4_6<=TT) ->
        (r1order'=1) & (r1time'=r1time+r1at4_6Time+travelr1at4_6);
 [r1at4_3] r1order=1 & (r1time+r1at4_3Time+travelr1at4_3<=TT) ->
        (r1order'=2) & (r1time'=r1time+r1at4_3Time+travelr1at4_3);
 [r1at3_7] r1order=2 & (r1time+r1at3_7Time+travelr1at3_7<=TT) & (r1time+travelr1at3_7 >= r1time) &
        (r1at4_6Done) -> (r1order'=3) & (r1time'=r1time+r1at3_7Time+travelr1at3_7);
 [r1at2_5] r1order=3 & (r1time+r1at2_5Time+travelr1at2_5<=TT) & (r1time+travelr1at2_5 >= r1time) &
        (r1at4_3Done) -> (r1order'=4) & (r1time'=r1time+r1at2_5Time+travelr1at2_5);
 [r1at2_8] r1order=4 & (r1time+r1at2_8Time+travelr1at2_8<=TT) & (r1time+travelr1at2_8 >= r1time) &
        (r1at4_6Done) -> (r1order'=5) & (r1time'=r1time+r1at2_8Time+travelr1at2_8);
 [r1at3_4] r1order=5 & (r1time+r1at3_4Time+travelr1at3_4<=TT) & (r1time+travelr1at3_4 >= r1time) &
        (r1at4_3Done) -> (r1order'=6) & (r1time'=r1time+r1at3_4Time+travelr1at3_4);
 [r1idle] r1order!=6 & (r1time+1<=TT) & (r1idleTime+1<=maxIdler1) ->
        (r1time'=r1time+1) & (r1idleTime'=r1idleTime+1);
endmodule

rewards "idle"
 //Note- there is no idle option for robot ri if maxIdleri==0 (computed beforehand)
 [r1idle] true: 1;
endrewards
```

Figure 8.5: MDP model for group 3.

F MDP and DTMC model from task scheduling

This is a continuation of Section 6.5. Figures 8.5 and 8.6 show the MDP and DTMC models, respectively, of group 3 consisting of a single robot, $r1$. Figure 8.7 shows the DTMC model of group 1 consisting of robot $r3$.



```
dtmc

const double p_travel_r1at4_6=1.0 ;// from location: l1 (robot initial loc) to location: room3 (at4_6)
const double p_travel_r1at4_3=0.9 ;// from location: room3 (at4_6) to location: room2(at4_3)
const double p_travel_r1at3_7=0.9 ;// from location: room2 (at4_3) to location: room3(at3_7)
const double p_travel_r1at2_5=0.9 ;// from location: room3 (at3_7) to location: room2(at2_5)
const double p_travel_r1at2_8=0.9 ;// from location: room2 (at2_5) to location: room3(at2_8)
const double p_travel_r1at3_4=0.9 ;// from location: room3 (at2_8) to location: room2(at3_4)
const double p_r1at4_6=0.99 ;
const double p_r1at4_3=0.99 ;
const double p_r1at3_7=0.95 ;
const double p_r1at2_5=0.95 ;
const double p_r1at2_8=0.95 ;
const double p_r1at3_4=0.95 ;

formula done=(r1=24);

module r1
 r1:[0..24];
 //travel to at4_6
 [r1travel_at4_6] r1=0->p_travel_r1at4_6:(r1'=2) + 1-p_travel_r1at4_6:(r1'=1);
 //try at4_6, no retry allowed
 []r1=2 -> 0.99:(r1'=4) + 1-0.99:(r1'=3); //fail task at r1=3
 //travel to at4_3
 [r1travel_at4_3] r1=4->p_travel_r1at4_3:(r1'=6) + 1-p_travel_r1at4_3:(r1'=5);
 //try at4_3, no retry allowed
 []r1=6 -> 0.99:(r1'=8) + 1-0.99:(r1'=7); //fail task at r1=7
 //travel to at3_7
 [r1travel_at3_7] r1=8->p_travel_r1at3_7:(r1'=10) + 1-p_travel_r1at3_7:(r1'=9);
 //try at3_7, no retry allowed
 []r1=10 -> 0.95:(r1'=12) + 1-0.95:(r1'=11); //fail task at r1=11
 //travel to at2_5
 [r1travel_at2_5] r1=12->p_travel_r1at2_5:(r1'=14) + 1-p_travel_r1at2_5:(r1'=13);
 //try at2_5, no retry allowed
 []r1=14 -> 0.95:(r1'=16) + 1-0.95:(r1'=15); //fail task at r1=15
 //travel to at2_8
 [r1travel_at2_8] r1=16->p_travel_r1at2_8:(r1'=18) + 1-p_travel_r1at2_8:(r1'=17);
 //try at2_8, no retry allowed
 []r1=18 -> 0.95:(r1'=20) + 1-0.95:(r1'=19); //fail task at r1=19
 //travel to at3_4
 [r1travel_at3_4] r1=20->p_travel_r1at3_4:(r1'=22) + 1-p_travel_r1at3_4:(r1'=21);
 //try at3_4, no retry allowed
 []r1=22 -> 0.95:(r1'=24) + 1-0.95:(r1'=23); //fail task at r1=23
endmodule
```

Figure 8.6: DTMC model for group 3.

```
dtmc

const double p_travel_r3at4_9=1.0 ;// from location: l3 (robot initial loc) to location: room4 (at4_9)
const double p_travel_r3at2_11=1.0 ;// from location: room4 (at4_9) to location: room4(at2_11)
const double p_travel_r3at3_10=1.0 ;// from location: room4 (at2_11) to location: room4(at3_10)
const double p_travel_r3at4_12=1.0 ;// from location: room4 (at3_10) to location: room5(at4_12)
const double p_travel_r3at3_13=1.0 ;// from location: room5 (at4_12) to location: room5(at3_13)
const double p_travel_r3at2_14=1.0 ;// from location: room5 (at3_13) to location: room5(at2_14)
const double p_r3at4_9=0.99 ;
const double p_r3at2_11=0.99 ;
const double p_r3at3_10=0.99 ;
const double p_r3at4_12=0.99 ;
const double p_r3at3_13=0.99 ;
const double p_r3at2_14=0.99 ;

formula done=(r3=24);

module r3
 r3:[0..24];
 //travel to at4_9
 [r3travel_at4_9] r3=0->p_travel_r3at4_9:(r3'=2) + 1-p_travel_r3at4_9:(r3'=1);
 //try at4_9, no retry allowed
 []r3=2 -> 0.99:(r3'=4) + 1-0.99:(r3'=3); //fail task at r3=3
 //travel to at2_11
 [r3travel_at2_11] r3=4->p_travel_r3at2_11:(r3'=6) + 1-p_travel_r3at2_11:(r3'=5);
 //try at2_11, no retry allowed
 []r3=6 -> 0.99:(r3'=8) + 1-0.99:(r3'=7); //fail task at r3=7
 //travel to at3_10
 [r3travel_at3_10] r3=8->p_travel_r3at3_10:(r3'=10) + 1-p_travel_r3at3_10:(r3'=9);
 //try at3_10, no retry allowed
 []r3=10 -> 0.99:(r3'=12) + 1-0.99:(r3'=11); //fail task at r3=11
 //travel to at4_12
 [r3travel_at4_12] r3=12->p_travel_r3at4_12:(r3'=14) + 1-p_travel_r3at4_12:(r3'=13);
 //try at4_12, no retry allowed
 []r3=14 -> 0.99:(r3'=16) + 1-0.99:(r3'=15); //fail task at r3=15
 //travel to at3_13
 [r3travel_at3_13] r3=16->p_travel_r3at3_13:(r3'=18) + 1-p_travel_r3at3_13:(r3'=17);
 //try at3_13, no retry allowed
 []r3=18 -> 0.99:(r3'=20) + 1-0.99:(r3'=19); //fail task at r3=19
 //travel to at2_14
 [r3travel_at2_14] r3=20->p_travel_r3at2_14:(r3'=22) + 1-p_travel_r3at2_14:(r3'=21);
 //try at2_14, no retry allowed
 []r3=22 -> 0.99:(r3'=24) + 1-0.99:(r3'=23); //fail task at r3=23
endmodule
```

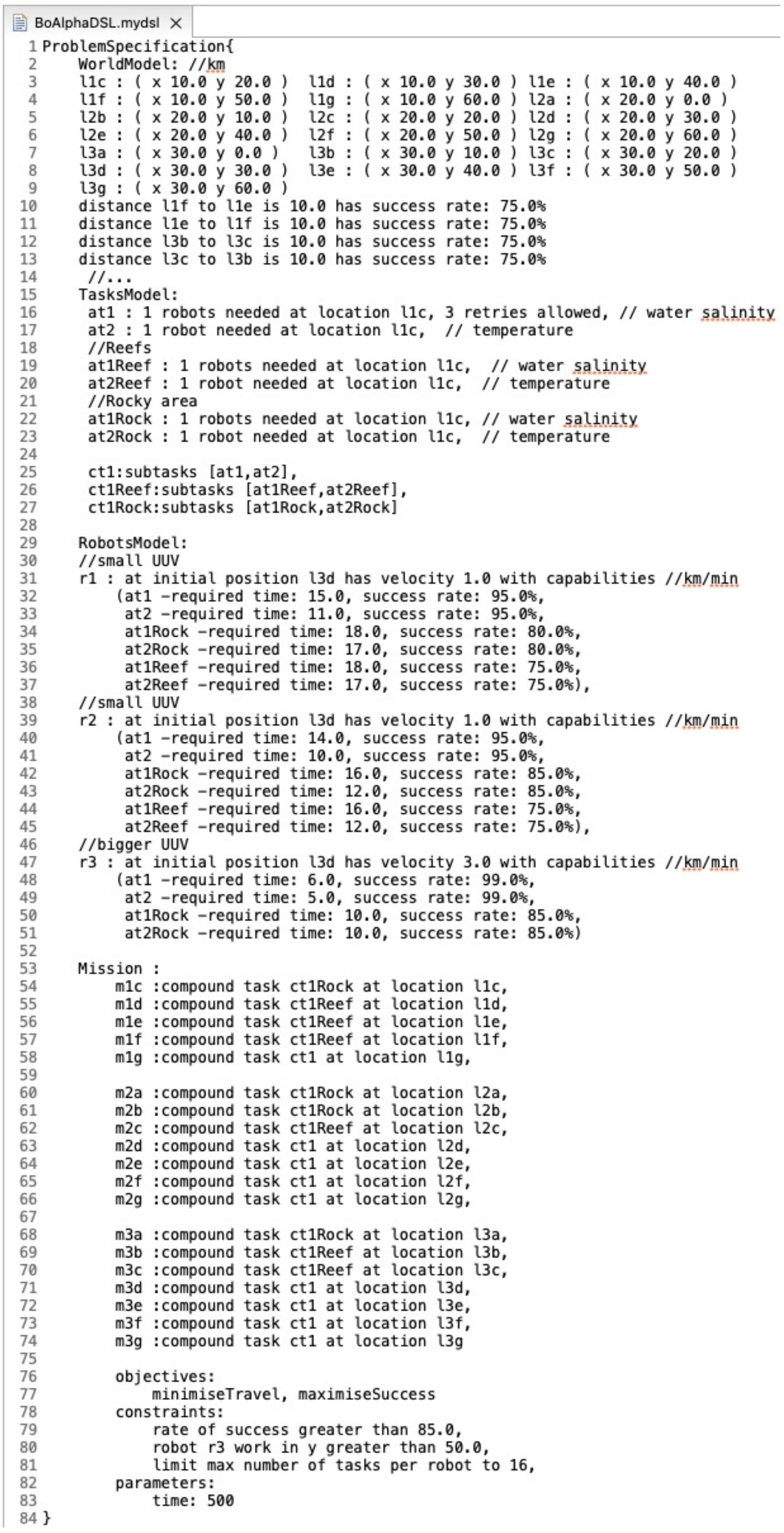
Figure 8.7: DTMC model for group 1.

Table 8.2: Robot parameter values for the Bo-Alpha case study: velocity (meters/minute), time to complete a task (minutes) and probability of completing a task. NA means not apply as the robot does not possess the capability to perform the task.

			Time of completion (min) [zone]		**Prob. of completion [zone]**	
Robot	**Initial Loc.**	**Velocity (km/m)**	**AT1 –salinity**	**AT2 –temp.**	**AT1**	**AT2**
r1	l3d	1	15 [deep water] 18 [rocky] 18 [reef]	11[deep water] 17 [rocky] 17 [reef]	0.95 [deep water] 0.80 [rocky] 0.75 [reef]	0.95 [deep water] 0.80 [rocky] 0.75 [reef]
r2	l3d	1	14 [deep water] 16 [rocky] 16 [reef]	10[deep water] 12 [rocky] 12 [reef]	0.95 [deep water] 0.85 [rocky] 0.75 [reef]	0.95 [deep water] 0.85 [rocky] 0.75 [reef]
r3	l3d	1	6 [deep water] 10 [rocky]	5[deep water] 10 [rocky]	0.99 [deep water] 0.85 [rocky]	0.99 [deep water] 0.85 [rocky]

G Bo-Alpha case study complementary material

This section contains complementary material to the Bo-Alpha study presented in Section 7.2. The robots' capabilities are depicted in Figure 8.2, while the KANOA's problem specification is in Figure 8.8.

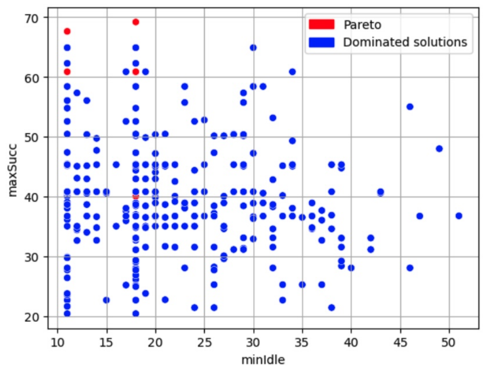


```
ProblemSpecification{
    WorldModel: //km
    l1c : ( x 10.0 y 20.0 )  l1d : ( x 10.0 y 30.0 ) l1e : ( x 10.0 y 40.0 )
    l1f : ( x 10.0 y 50.0 )  l1g : ( x 10.0 y 60.0 ) l2a : ( x 20.0 y 0.0 )
    l2b : ( x 20.0 y 10.0 )  l2c : ( x 20.0 y 20.0 ) l2d : ( x 20.0 y 30.0 )
    l2e : ( x 20.0 y 40.0 )  l2f : ( x 20.0 y 50.0 ) l2g : ( x 20.0 y 60.0 )
    l3a : ( x 30.0 y 0.0 )   l3b : ( x 30.0 y 10.0 ) l3c : ( x 30.0 y 20.0 )
    l3d : ( x 30.0 y 30.0 )  l3e : ( x 30.0 y 40.0 ) l3f : ( x 30.0 y 50.0 )
    l3g : ( x 30.0 y 60.0 )
    distance l1f to l1e is 10.0 has success rate: 75.0%
    distance l1e to l1f is 10.0 has success rate: 75.0%
    distance l3b to l3c is 10.0 has success rate: 75.0%
    distance l3c to l3b is 10.0 has success rate: 75.0%
     //...
    TasksModel:
     at1 : 1 robots needed at location l1c, 3 retries allowed, // water salinity
     at2 : 1 robot needed at location l1c,  // temperature
     //Reefs
     at1Reef : 1 robots needed at location l1c,  // water salinity
     at2Reef : 1 robot needed at location l1c,  // temperature
     //Rocky area
     at1Rock : 1 robots needed at location l1c, // water salinity
     at2Rock : 1 robot needed at location l1c,  // temperature

     ct1:subtasks [at1,at2],
     ct1Reef:subtasks [at1Reef,at2Reef],
     ct1Rock:subtasks [at1Rock,at2Rock]

    RobotsModel:
    //small UUV
    r1 : at initial position l3d has velocity 1.0 with capabilities //km/min
        (at1 -required time: 15.0, success rate: 95.0%,
         at2 -required time: 11.0, success rate: 95.0%,
         at1Rock -required time: 18.0, success rate: 80.0%,
         at2Rock -required time: 17.0, success rate: 80.0%,
         at1Reef -required time: 18.0, success rate: 75.0%,
         at2Reef -required time: 17.0, success rate: 75.0%),
    //small UUV
    r2 : at initial position l3d has velocity 1.0 with capabilities //km/min
        (at1 -required time: 14.0, success rate: 95.0%,
         at2 -required time: 10.0, success rate: 95.0%,
         at1Rock -required time: 16.0, success rate: 85.0%,
         at2Rock -required time: 12.0, success rate: 85.0%,
         at1Reef -required time: 16.0, success rate: 75.0%,
         at2Reef -required time: 12.0, success rate: 75.0%),
    //bigger UUV
    r3 : at initial position l3d has velocity 3.0 with capabilities //km/min
        (at1 -required time: 6.0, success rate: 99.0%,
         at2 -required time: 5.0, success rate: 99.0%,
         at1Rock -required time: 10.0, success rate: 85.0%,
         at2Rock -required time: 10.0, success rate: 85.0%)

    Mission :
        m1c :compound task ct1Rock at location l1c,
        m1d :compound task ct1Reef at location l1d,
        m1e :compound task ct1Reef at location l1e,
        m1f :compound task ct1Reef at location l1f,
        m1g :compound task ct1 at location l1g,

        m2a :compound task ct1Rock at location l2a,
        m2b :compound task ct1Rock at location l2b,
        m2c :compound task ct1Reef at location l2c,
        m2d :compound task ct1 at location l2d,
        m2e :compound task ct1 at location l2e,
        m2f :compound task ct1 at location l2f,
        m2g :compound task ct1 at location l2g,

        m3a :compound task ct1Rock at location l3a,
        m3b :compound task ct1Reef at location l3b,
        m3c :compound task ct1Reef at location l3c,
        m3d :compound task ct1 at location l3d,
        m3e :compound task ct1 at location l3e,
        m3f :compound task ct1 at location l3f,
        m3g :compound task ct1 at location l3g

        objectives:
            minimiseTravel, maximiseSuccess
        constraints:
            rate of success greater than 85.0,
            robot r3 work in y greater than 50.0,
            limit max number of tasks per robot to 16,
        parameters:
            time: 500
}
```

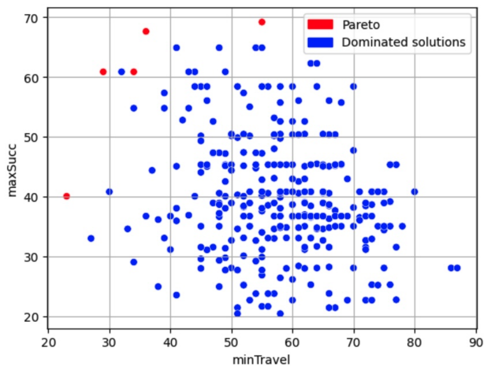
Figure 8.8: Bo-Alpha problem specification in KANOA's DSL.

H Hospital dominated solutions and reference Pareto front

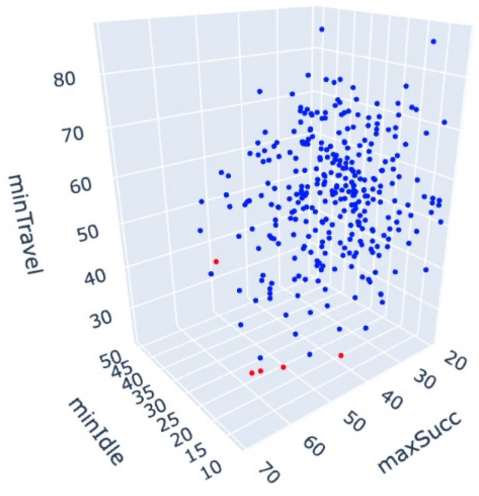
The evaluation section 7.2 contains the values of the reference Pareto for the hospital case study. A visualisation of these and the all dominated solutions is shown in this section.



(a)



(b)



(c)

Figure 8.9: Feasible (attribute values of) solutions found for the Hospital case study. Figures (a) and (b) show different 2D views of the attribute values of the solution space. Figure (c) is the combined 3D plot. Figure Pareto front of solution depicted in red.