Scheduling of Missions with Constrained Tasks
for Heterogeneous Robot Systems

Gricel Vázquez   Radu Calinescu   Javier Cámara

Department of Computer Science   Department of Computer Science
University of York   University of Málaga
York, UK   Málaga, Spain

`gnvf500@york.ac.uk`   `radu.calinescu@york.ac.uk`   `javier.camaramoreno@york.ac.uk`

We present a formal task allocation and scheduling approach for multi-robot missions (KANOA). KANOA supports two important types of task constraints: task ordering, which requires the execution of several tasks in a specified order; and joint tasks, which indicates tasks that must be performed by more than one robot. To mitigate the complexity of robotic mission planning, KANOA handles the allocation of the mission tasks to robots, and the scheduling of the allocated tasks separately. To that end, the task allocation problem is formalised in first-order logic and resolved using the Alloy model analyzer, and the task scheduling problem is encoded as a Markov decision process and resolved using the PRISM probabilistic model checker. We illustrate the application of KANOA through a case study in which a heterogeneous robotic team is assigned a hospital maintenance mission.

1 Introduction

Multi-robot systems (MRS) are increasingly used in domains ranging from medical and emergency assistance [29] to inspection of critical infrastructures [12]. Nevertheless, the specification of complex missions with constrained tasks, the allocation of these tasks to the robots, and the scheduling of allocated tasks for each robot continues to pose significant challenges to the use of MRS in these applications [28, 32, 40].

Our paper introduces KANOA, a new MRS task allocation and scheduling approach that uses a combination of formal methods to solve what Tran et al. [32] describe as "*the joint problem of deciding what tasks to perform, when, and with what resources, to achieve a set of goals*". Task allocation and scheduling can be tackled in two ways: as a single, monolithic problem [14], or separately, as two interrelated sub-problems [18]. KANOA does the latter. As in [18], we posit that this enables a separation of concerns that not only increases scalability, but also simplifies the enforcement of a broad range of mission constraints and the achievement of optimization objectives.

Task allocation and scheduling is among the problems that can be approached using formal methods [37], which offer underlying logics for the unambiguous specification of missions, and behavioural models that are able to capture important MRS mission characteristics such as task interdependence [13]. However, the adoption of formal methods in real MRS scenarios has been limited due to the complexity of formalising MRS missions and their requirements into the required modelling paradigms and logic formulae, respectively [39, 38]. To address this limitation, KANOA (i) supports the specification of MRS missions with their tasks, task constraints and requirements in a domain-specific language, and (ii) automates both the allocation of tasks to robots (by using the model analyzer Alloy [24]) and the scheduling of the allocated tasks (by using the probabilistic model checker PRISM [26]). PRISM schedules the tasks allocated to each robot by solving the ordering of the tasks through synthesising a Markov decision process policy that minimises the time needed to complete the tasks.

The rest of the paper is organised as follows. After providing a motivating example in Section 2, we present our KANOA approach in Section 3, and its implementation and evaluation in Sections 4 and 5, respectively. Section 6 discusses related work, and Section 7 a short summary.

2   Motivating Scenario

We will illustrate our approach in the context of a hospital environment, in which patient recovery rooms must be kept clean, and medical equipment has to be moved around to perform daily operations (Fig. 1). We introduce two types of tasks: atomic tasks, AT; and compound tasks, CT. The MRS mission consists of six tasks: moving medical equipment in Room1 (task M1) and Room6 (task M2); and cleaning patient rooms 2–5 (tasks M3 to M6). Moving medical equipment is a type of task (AT1) that must be performed by two robots. Cleaning a patient's room (CT2) requires notifying the patient (AT4) and cleaning the room (CT1), in that specific order. Finally, cleaning a room comprises floor cleaning (AT2) and sanitising (AT3). Five robots are available: two cleaner robots (r1 and r2) and three pick-and-place robots (r3-r5).

In addition to the constraints imposed by the ordering of tasks and the tasks requiring more than one robot, the mission must be completed within 100 time units, and the robots that are used to perform the mission cannot spend more than 30 time units idle. Furthermore, the tasks must be allocated and scheduled such that the total robot idle time and travelling time are minimised, and the probability of succeeding with the mission is maximised. An example of a synthesized plan (discussed later in the paper) is shown by the red arrows from Fig. 1.

3   KANOA Approach

KANOA employs the four-stage process from Fig. 2. In stage 1, the KANOA domain-specific language (DSL) is used to specify the world model, robots model, task model and mission that together form the problem that KANOA needs to solve. In stage 2, KANOA solves the allocation of tasks to robots, producing multiple allocation models, all of
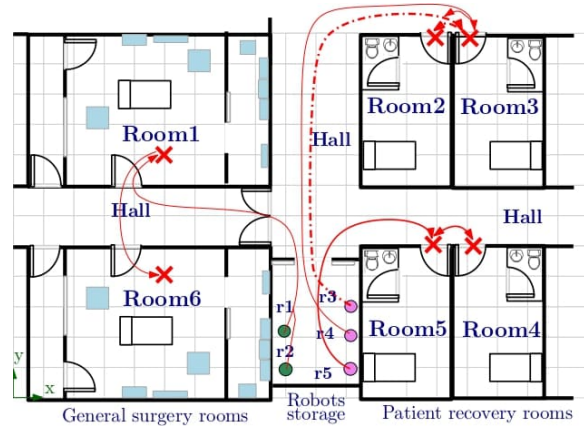

Figure 1: Floor plan of hospital area with six rooms serviced by robots r1-r5. Crosses mark the locations that robots must reach to start a task in each room, and blue rectangles denote medical equipment (inspired by [30]). The red lines shows a possible robots' schedule to clean rooms 2 to 4, and to move equipment in rooms 1 and 6.
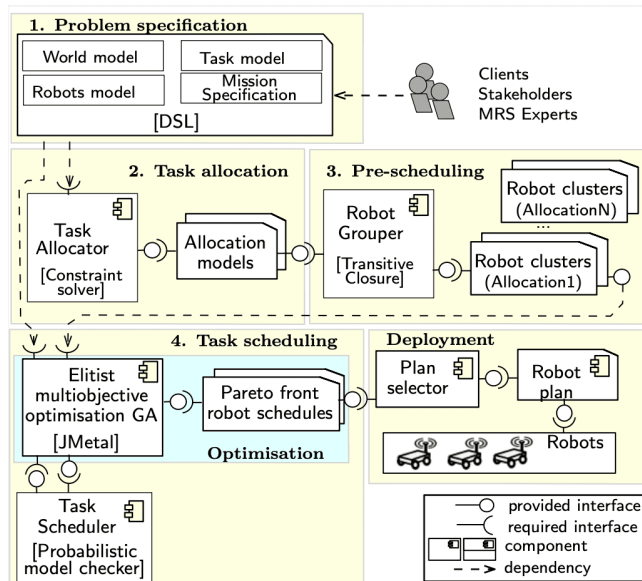

Figure 2: The four-stage KANOA approach

Name	Term	Syntax
problem spec	problem	::= `ProblemSpecification { WorldModel:` world `TaskModel:` tasks `RobotsModel:` robots `Mission:` miss `}`
world model	world	::= $(\texttt{loc}\|\texttt{dist})^+$
location	loc	::= locID `:(x` v_1 `y` v_2 `)`
path	dist	::= locID `to` locID `, distance` v
task model	tasks	::= $(\texttt{a\_task}\|\texttt{c\_task})^+$
atomic task	a_task	::= atID `:` v `robots needed`
compound task	c_task	::= ctID `:` `subtasks [` $(\text{atID} \mid \text{ctID})^+$ `] ordered:` v_{bool}
robot model	robots	::= $\texttt{robot}^+$
robot	robot	::= robID`:at initial location` locID `has velocity` v `can do (` $\texttt{capab}^+$ `)`
capability	capab	::= atID `- time required:` v`, success probability:`$v_{[0,1]}$
mission	miss	::= $(\texttt{m\_task}\|\texttt{constraint})^+$
mission task	m_task	::= mtID`: (compound task` ctID `| atomic task` atID `) at` locID
constraint	constraint	::= space|time
space	space	::= setXY|betweenXY
set coordinate	setXY	::= `(robot` robID`|all robots) work in (x | y) (lower than | greater than | geq | leq)` v
space constraint	betweenXY	::= `robot` robID `work between (x | y) [`v_1`,`v_2`]`
deadline	time	::= `time available:` v

* v, v_1, v_2 are nonnegative integers; $v_{[0,1]}$ is a value between 0 and 1; v_{bool} is a Boolean value; and locID, robID, atID, ctID, mtID are identifiers (i.e., strings starting with a letter) for locations, robots, and atomic/compound/mission tasks, respectively

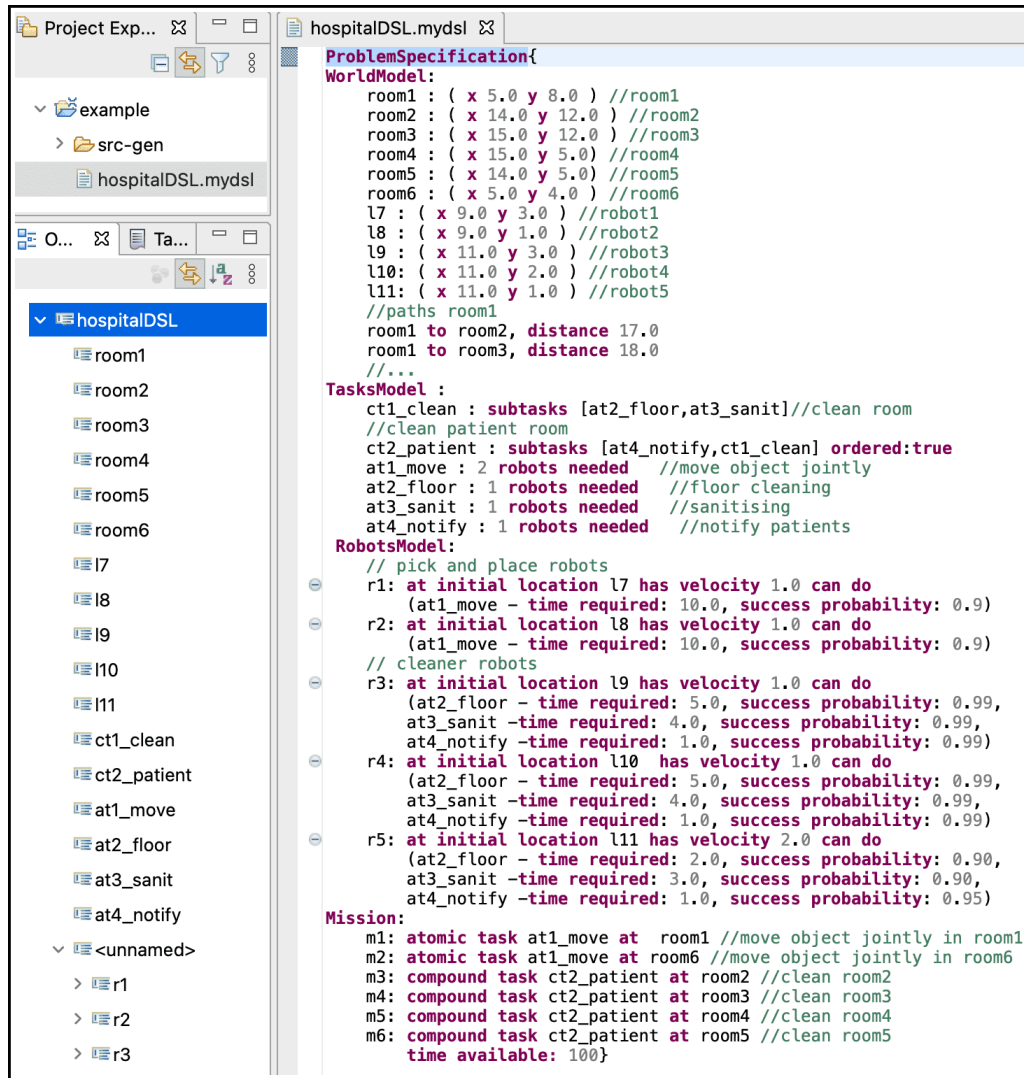Figure 3: KANOA problem specification DSL. Based on the EBNF metasyntax notation [1].

which comply with the robot capabilities. Stage 3, pre-scheduling, groups robots that share constrained tasks, so that their tasks are scheduled together. Finally, the task scheduling stage generates a set of feasible MRS plans that satisfy all task constraints and are Pareto optimal with respect to the optimisation objectives from the mission specification. The user can select one of the synthesized plans to be deployed to the physical robots, as shown in the deployment stage. KANOA focuses on the four KANOA stages as detailed below.

3.1 Stage 1: Problem specification

We use the user-friendly DSL with the syntax from Fig. 3 for the unambiguous specification of the KANOA task allocation and scheduling problem. In line with established conventions, alternative items are divided by | and may be enclosed in brackets (). Items that can appear one or more times are followed by a plus +.

A KANOA **problem** specification comprises a **world** model, a **task** model, a **robot** model, and a **mission** specification:

- The **world** model consists of a sequence of locations **loc** and path (distances) **dist**, where each location is defined by an identifier **locID** and its **x** and **y** coordinates, and each path **dist** specifies the all of the distance between two locations.

- The **tasks** model defines a set of *atomic tasks* and *compound tasks*. An atomic task (**a_task**) needs to be performed in one go, by a predefined number of **robots needed**. In contrast, a compound task (**c_task**) comprises a set of **subtasks**, each of which is an atomic or a compound task. The **subtasks** of a compound task may need to be performed in the specified order.

Figure 4: KANOA tool screenshot showing DSL encoding of the hospital MRS mission from Section 2

- The **robots** model specifies, for each robot available, its **initial position**, its (mean) **velocity** and its list of *capabilities*, i.e., the identifiers **atID** of the atomic tasks that the robot **can do**, and the **required time** and **success probability** associated with the task execution by the robot[1].

- The mission specification (**miss**) comprises a sequence of mission tasks, **constraint**s. A mission task (**m_task**) specifies a **compound task** or **atomic task** (from the **task** model), which needs to be performed **at** a location specified by **locID**. **Constraint**s define boundaries within which a specific **robot robID** or **all robots** must reside at all times, or the **time available** to complete the mission.

Example 3.1. Figure 4 shows the KANOA problem specification of the hospital scenario from our motivating example. As detailed in Section 2, this scenario comprises: (a) six rooms with their (entry

---

[1]The success probabilities differ from robot to robot based on age, manufacturing differences, etc.; and calculated in advance after N runs of the physical or simulated robots

point) coordinates, initial locations for the five robots, and all distances between these; (b) two compound tasks and four atomic tasks; (c) two pick-and-place robots (r1 and r2) and three cleaner robots (r3–r5); (d) the mission that the robots are required to carry out (within 100 time units), consisting of moving medical equipment in rooms 1 and 6, and cleaning the patient rooms 2–5. Pick-and-place robots r1 and r2 require 10 time units to reorganise the medical equipment in a surgical room. Cleaner robots r3–r5 have a higher probability of success compared to r5, but r5 travels twice as fast as r3 and r4.

3.2   Stage 2: Task allocation

The multi-robot task allocation problem involves the optimal partitioning of the tasks of a mission among the available robots. This problem is very challenging due to the heterogeneity of robots, the unreliability of sensors and actuators, temporal and spatial task constraints, the complexity of task dependencies, and the search for an optimal allocation [2]. The KANOA Task Allocator (see Fig. 2) performs the allocation of tasks to robots by using the Alloy Analyzer constraint solver [24] to generate feasible allocations through reasoning about the capabilities of each robot, and the spatial constraints from the mission specification.

3.2.1   Background

Alloy is a formal specification language (supported by the *Alloy analyzer* tool) that has its roots in the Z specification language [24]. It supports the formal description of the structural properties of systems. Alloy is defined in terms of simple relational semantics, using constraint solver notation, and employing constructs that are common in object-oriented notations. We can indicate the existence of a set (of atoms) for robots, which in Alloy is specified using *signatures*:

> **abstract sig** Robot {hascapability: **some** Capability}

In this signature, Robot.hascapability denotes a relation with a non-empty set of capabilities that a robot possesses. The keyword 'abstract' indicates that new signatures can *extend* this signature, inheriting all of its 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 { ... }

Alloy models can incorporate relational logic sentences in the form of predicates and *facts* (i.e., expressions that always have to be satisfied). These constructs place explicit constraints on the model. Hence, when the Alloy analyzer searches for instantiations 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 makes sure 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 $\Leftrightarrow$ c in r.hascapability }

The Alloy analyzer can look for examples of structures that satisfy all of the relational constraints in the model within a finite scope that explicitly sets the maximum number of atoms of each signature to be considered by a solution.

3.2.2   Task allocation problem

We define the task allocation problem in the Alloy declarative language with abstract signatures for atomic tasks, robot capabilities and robots. There are four facts associated with these signatures. First, a
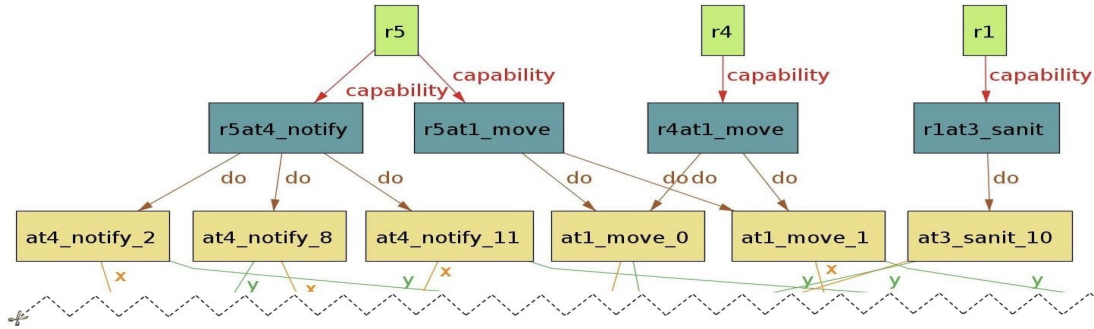
Figure 5: Allocation generated by Alloy Analyzer.

robot has its own set of capabilities, meaning that each capability belongs to only one robot. Second, if a robot appears in the allocation (see Figure 5) it must have assigned tasks, so that robots that are not used are ignored. Third, only capabilities that have allocated tasks are shown. Finally, each robot appears once or, if not selected by the allocator, does not appear at all.

For the declaration of atomic tasks, we model a second abstraction extending the *AtomicTask* abstract signature. These are atomic tasks specified in the problem specification. The instances of these tasks are atomic tasks required by the mission (recursively fetched from the *tree-like* structure of the tasks model). We explicitly define the maximum number of robots and capabilities, the number of atomic tasks and each atomic task, to generate $N > 0$ possible allocations, where N is a KANOA configuration parameter.

```
1: // robot r1
2: sig r1 extends Robot { }
3:      {disj [ capability, Capability - r1at2_-
   floor -
4:      r1at3_sanit - r1at4_notify ]}
5: fact {#r1<=1}
6: // robot capability
7: sig r1at2_floor extends Capability { }
8:      {all d:do | d in at2_floor}
9: // all capabilities instances appear max. once
10: fact { #r1at2_floor < =1
11:      #r1at3_sanit<=1
12:      #r1at4_notify<=1 ...
```

Figure 6: Path from the initial location of robot r2 to room 6 found using PRM

Figure 5 shows part of an allocation found by Alloy Analyser. At the top are three of the robots, $r5, r4$ and $r1$. Only capabilities needed from the robots for the allocated tasks are shown, for instance, $r1$ only shows one capability $r1at3\_sanit$. Notice that robots $r4$ and $r5$ have the same allocated task $at1\_move\_0$ as it is a joint task. Links (x,y) at the bottom of the figure point to the locations where each atomic task must be done.

Example 3.2 An example of a robot instance based on our motivating example is shown in Figure 6, lines 1-4. In this case, robot $r1$ has capabilities $r1at2\_floor, r1at3\_sanit$ and $r1at4\_notify$, modelled as the disjoint set of these tasks robot capabilities and the set of all capabilities. The number of robots required is written as a fact (line 5). A robot capability instance has the constraint of only performing atomic tasks in a specific scope. For example, robot $r1$ with capability $r1at2\_floor$ can only perform $at2\_floor$ type of tasks, as specified in Figure 6 lines 6-8. We add a constraint to show a capability instance once, or 0 if it has no allocated tasks (lines 9-13). Due to space constraints, we refer the user our GitHub repository for the complete Alloy code for the motivating example [33].

3.3   Stage 3: Pre-scheduling

For each allocation found by KANOA, we reduce (where possible) the complexity of the scheduling models, by creating separate models for each *group* of robots that share task dependencies and do not
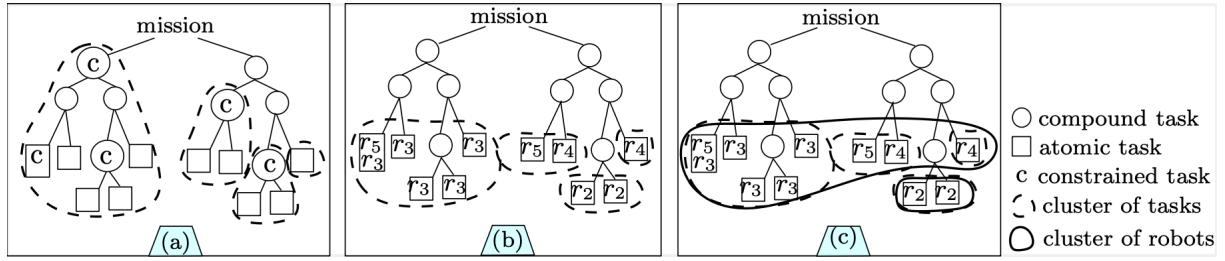
Figure 7: Transitive closure of robot interdependence relation: (a) atomic tasks clustered by breath-first until a constrained task is found; (b) for each subtree, the robots in the current allocation are obtained; (c) the transitive closure returns clusters of robots—in this example, there are two robot clusters, i.e., $\{r3, r4, r5\}$ and $\{r2\}$.

share any robot with the other groups. We call this stage *pre-scheduling*, as shown in Figure 2.

Let T be the tree of tasks from the mission specification to the atomic tasks. First, KANOA computes the Breadth-first search BFS algorithm pruning the tree every time that a constraint (joint or consecutive task) is found, or a leaf node (atomic task) is reached, returning multiple subtrees, as shown in Fig. 7a.

$$BFS(T) = \{subtree_1, subtree_2, ...\} \tag{1}$$

Let A_i be the ith allocation of tasks to robots found by Alloy, and Rob_i the set of all robots in A_i; for example, the task allocation in Figure 7.b has robots $Rob_i = [r2, r3, r4, r5]$. For each $subtree_j \in BFS(T)$, we define the set of robots associated with its leaf nodes by,

$$Rob_{ij}(A_i, subtree_j) \subseteq Rob_i \tag{2}$$

as shown in Fig. 7b. We define a reflexive binary relation R between robot r_a and r_b as,

$$r_a R r_b = \begin{cases} 1, & \text{if } \exists subtree \in BFS(T) \cdot r_a, r_b \in subtree \\ 0, & \text{otherwise} \end{cases}$$

Finally, we define the *transitive closure* of R as a reflexive relation R' such that $r_a R' r_k$ if there is a sequence $(r_a, r_b, ..., r_j, r_k)$ of the members of the domain of R such that $r_a R r_b, ..., r_j R r_k$, as shown in Fig. 7c. The relation R may be represented as a Boolean matrix M (where $M(r_a, r_b) = r_a R r_b$), and R' as matrix M' computed from M using *Warshall*'s algorithm or variations [31].

3.4 Stage 4: Task scheduling

KANOA uses probabilistic model checking for the scheduling of individual robot plans. In short, it models the scheduling problem as a Markov decision process (MDP), finding a policy that optimises a set of objectives under a set of constraints. A policy is a solution to the non-determinism actions, assessing if a robot should do the next task or wait. Hence, finding a policy means finding a feasible schedule.

3.4.1 Background

Markov decision processes (MDPs) support the modelling of probabilistic and nondeterministic behaviour [34]. An MDP is defined as a tuple $\mathcal{M} = (S, s_i, A, \delta_{\mathcal{M}}, L)$, where S is the finite set of states and

$s_i \in S$ an initial state; A is the set of actions and $\delta_{\mathcal{M}} : S \times A \to Dist(s)$ is a partial probabilistic function returning a distribution over S when an action $a \in A$ is taken from $s \in S$; and, $L : S \to 2^{AP}$ is a labelling function assigning a set of atomic prepositions from a set AP to each state $s \in S$. *Reward* structures that associate a quantity to each action choice can be used, for example, to track the energy consumption, or the total travelling distance of a robot.

An MDP **policy** resolves the non-deterministic choice of an action in the states of an MDP [34]. KANOA uses *deterministic − memoryless* policies, which involves the selection of a fixed action whenever the same state is reached [16, 34].

Probabilistic computational tree logic (PCTL) augmented with rewards is used to define qualitative and quantitative formulae related to MDP rewards and probabilities that can be assessed via probabilistic model checking. A *state PCTL formula* Φ and a *path PCTL* formula Ψ are defined over a set of atomic prepositions AP by the grammar:

$$\Phi := true|\alpha|\neg\Phi|\Phi \wedge \Phi|\mathscr{P}_{\bowtie p}[\Psi]|\mathscr{R}_{\bowtie r}[F\Phi]$$
$$\Psi := X\Phi|\Phi U\Phi|\Phi U^{\leq k}\Phi \tag{3}$$

with atomic preposition $\alpha \in AP$, probability bound $p \in [0,1]$, reward bound $r \in \mathbb{R}_0^+$, time step bound $k \in \mathbb{N}_{>0}$, and $\bowtie \in \{\geq, >, <, \leq\}$. PCTL *semantics* is defined by the satisfaction relation $\models$. For a state $s \in S$ of an MDP $\mathcal{M}$, we describe a relation $\mathcal{M}, s \models \Phi$ as "*the formula Φ holds on s*". Hence, the following relations are valid: always $s \models true$; $s \in \neg\Phi$ iff $\neg(s \models \Phi)$; $s \models \alpha$ iff $\alpha \in L(s)$; and $s \models \Phi_1 \wedge \Phi_2$ iff $s \models \Phi_1$ and $s \models \Phi_2$. The *until* formula $\Phi_1 U \Phi_2$ holds for a path iff Φ_1 holds in the first i states and Φ_2 holds in the state $i + 1$. The *next* formula $X\Phi$ holds if Φ is satisfied in the following state. Formulae with the probability symbol $\mathscr{P}$, have their semantics defined over all policies σ of $\mathcal{M}$, such that $\mathscr{P}_{\bowtie p}[\Psi]$ is the probability that paths starting at a defined initial state satisfy a path property Ψ with probability $\bowtie p$ for all policies. Replacing $\bowtie p$ with *min =?* or *max =?* specifies the calculation of the minimum or maximum probability over all the MDP policies. Similarly, for reward formulae, $\mathscr{R}_{\bowtie r}[F\Phi]$ (where F represents a reachable state in the *future*) holds if the expected reward accumulated before reaching a state satisfying Φ is $\bowtie p$ for all policies. Replacing '$\bowtie r$' with '*min =?*' or '*max =?*' specifies the calculation of the minimum or maximum reward over all the MDP policies. We refer the reader to [34, 4, 22] for further details.

PRISM modelling language KANOA MDPs are defined in the high-level modelling language of the probabilistic model checker PRISM [26]. A model specified in this language comprises several interacting reactive modules. Each such module consists of a set of finite-valued state variables, and a number of 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 ($\to$) with a probability of *prob*. In an MDP, non-deterministic actions are taken when two transitions in the same or different modules have overlapping guards.

3.4.2 Multi-robot task scheduling problem

We use MDPs encoded in the PRISM language to capture the KANOA multi-robot task scheduling problem. Informally, we define this problem as finding a sequence of possible *robot actions* (execute
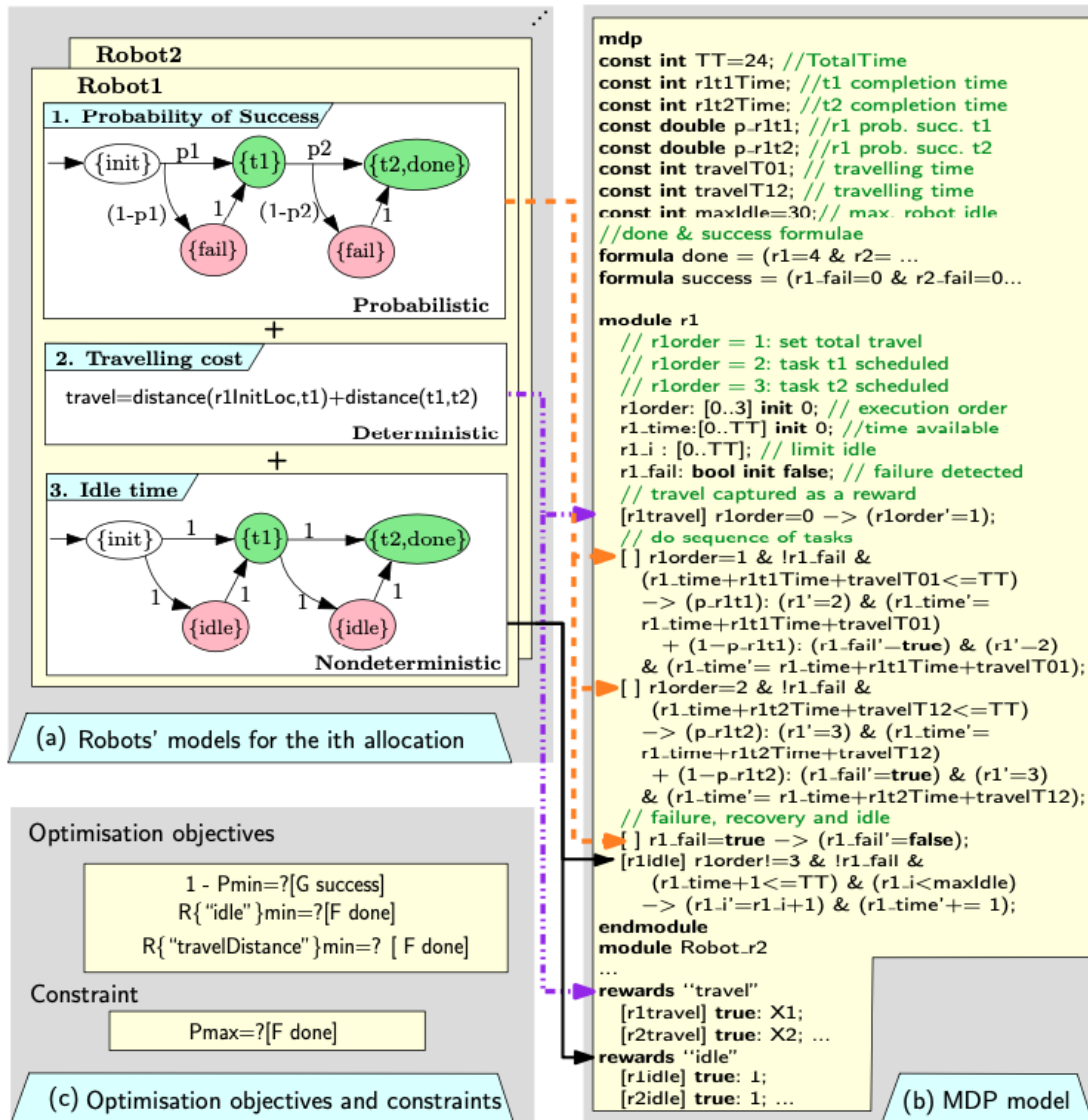
Figure 8: Model assembly for scheduling the tasks of interdependent robots under task constraints and multiple optimisation objectives: (a) representation of the models—(1) maximisation of probability of success, (2) minimisation of the travelling distance, and (3) minimisation of the idle time; (b) MDP assembly with the actions and state variables from (a); and (c) the three optimisation objectives. The bottom left shows the definitions for A and B for the MDP in (b).

a task, travel, or stay idle) such that, for each robot, all of its allocated tasks from Section 3.2.2 are completed with an optimal level (minimising the completion time, travelling cost and probability of success), preserving any required task ordering and joint execution of tasks across the schedules of all robots.

To this end, we first generate feasible permutations of tasks that satisfy the tasks constraints. This is done by randomly selecting a task that is not part of any constrained task, or the first task to be done in a task tree of ordered tasks. We use MDPs to model the robots' behaviour [26]. For an allocation *a*

with robot r_i, a permutation of its tasks is given by $p = a.modules[r_i]$. Each permutation is modelled as a PRISM module with state variables s_{r_i} defined by the tuple,

$$s_{r_i} = (r_i order, r_i time, r_i idle, r_i fail, r_i tasks) \tag{4}$$

where $r_i order$ tracks the order in which tasks are executed and the travelling action (further explained later); $r_i time$ tracks the current time updating every time a robot travels to another location or a task is completed and $r_i dle$ the time that the robot has spent idling; $r_i travel$ contains the information of the robots travelling time; $r_i fail$ is a boolean flag triggered when the robot fails performing a task; and $r_i tasks$ is a set of variables associated to task constraints (see Section 3.4.3). An MDP state is given by the composition of each robot module variables $s = (s_{r1}, s_{r2}, ..., s_{rn})$.

The MDP modules can be explained as the composition of three sub-models, as shown in Figure 8a. In the first model, a probabilistic choice is made between succeeding with a task or failing with a small probability. From the robot's initial position, where atomic proposition *init* is true, a robot travels and completes its first task with probability $p1$ (state with atomic preposition $t1$), and with probability $1 - p1$ fails. If a robot fails, the system recovers and continues with the next task[2] until all tasks are completed (label *done*).

Second, as shown in Section 2 from Figure 8a, the travelling cost is computed as the sum of the distances from the initial location of the robot to the location of its last task, passing through the locations of all the tasks of the current permutation[3].

Third, a nondeterministic action is available at any stage as the robot progresses with its tasks. This action allows the robot to stop for *one time unit*. This *idle* option allows robots to synchronise in the case they need to perform *join* tasks, or wait until previous tasks are completed in case of *ordered* tasks (see Figure 8a.3).

MDP model. Figure 8b shows the composition of an MDP model in the PRISM language. In the first line, **mdp** declares the type of model. Second, all necessary **constants** *const* that are used within the robot modules are declared[4]. Third, the boolean formula **done** is true when all robots have completed their tasks and the formula *success* is true when no robot fails with their tasks. Fourth, the robot modules ri contains the variables: a) $r_i order$, from 0 to $last(r_i order)$; b) $r_i time$, from 0 to the maximum time available (TT); and C) $r_i fail$ as a Boolean (changing to true when the robot fails a task). For each transition, the guard checks: the order of transitions, that the time does not exceed the available time and that the robot has not failed. When a transition is taken, with a probability of succeeding with the task, the time and order is updated, meaning that the task has been completed; and with a smaller probability, the robot fails and $r_i fail' = true$. Continuing, there is a transition for recovery after failure, where $r_i fail$ becomes false again; and a non-deterministic action $r_i idle$ (feasible if the robot hasn't finished, if it has not failed and if time allows) that adds +1 to the robot timer. Finally, a *travel* reward structure computes the cumulative travelling cost among robots; and an *idle* reward the cumulative time that the robots spend in stand-by (idling).

For an MDP, we use probabilistic model checking to assess if the selected permutation of tasks is

---

[2]We consider the probability of recovery of 1. However, it can be modelled as multiple recovery modes, for instance, with a probability $p1$ the robot can retry and succeed, with probability $p2$ it asks for help from a human, and with probability $p3$, the robot sends an alert that the task failed and needs to be rescheduled.

[3]The travelling cost changes among permutations, as the distance covered by robot $r1$ travelling between two tasks $t1, t2$, is different if following the order $\langle initialPosition(r), location(t1), location(t2)\rangle$ or $\langle initialPosition(r), location(t2), location(t1)\rangle$

[4]For the description of all constants *const*, we refer the reader directly to Figure 8b.

```
mdp
const bool ti_subtask_1 =false; //TotalTime
const bool ti_subtask_2 =false; //TotalTime
...
module Robot_r1
    // r1order = j: do task ti_subtask_2
    ...
  [ ] r1order=j & !r1_fail &
      (r1_time+r1tiTime+travelTi<=TT) &
      (r1_time >= r5_time) &
      (ti_subtask_1 =True)
          -> (p_r1ti): (r1order'=j+1) &
      (r1_time'+=r1tiTime+travelTi) &
      (ti_subtask_2'=true);
          + (1-p_r1ti): (r1_fail'=true) &
      (r1order'=j+1) &
      (r1_time'+=r1tiTime+travelTi) &
      (ti_subtask_2'=true);
...
endmodule                    (a)
```

```
mdp
...
module Robot_r1
    // r1order = i: travel to task JT1
    // r1order = i+1: do task JT1
    ...
  [ ] r1order=i & !r1_fail &
      (r1_time+travelTi<=TT) -> (p_r1t1):
          (r1order'=i+1) &
          (r1_time'+=travelTi) +
          (1-p_r1t1): (r1_fail'=true) &
          (r1order'=i+1) &
          (r1_time'+=travelTi) ;
  [JT1] r1order=i+1 & !r1_fail
      & r1_time=r2_time ->
          (r1_time'+= r1JT1);
...
endmodule                    (b)
```
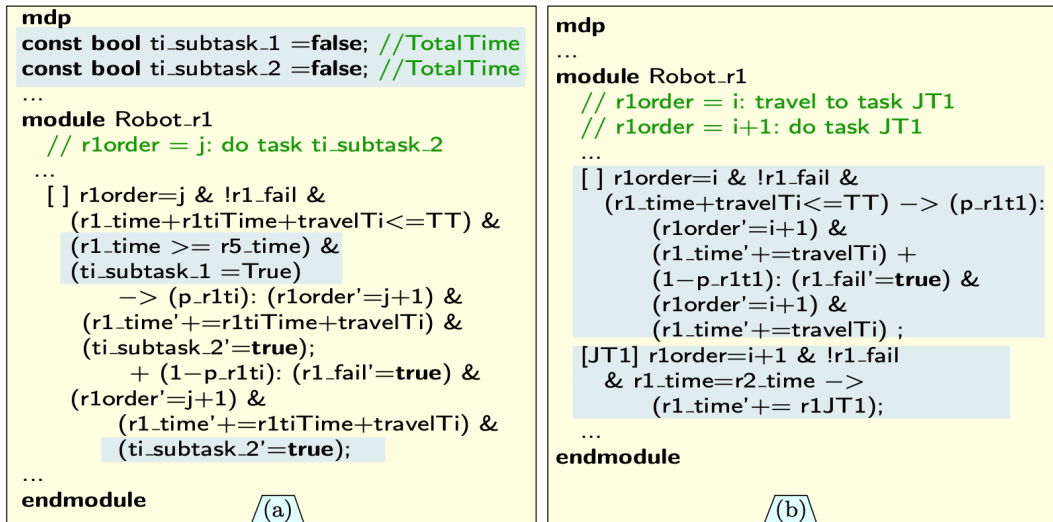
Figure 9: (a) Joint task JT1 in robot r1 written in the PRISM language as two transitions: first travel, then synchronise with other robots via the action label [*JT*1]. All transitions in other modules, with the same action label, "transition" at the same time. (b) Robot r1 is assigned with the second subtask of the ordered task *ti*. In its *jth* transition, r1 checks if robot r5 has already completed the first subtask in the past, by checking if r1 time is greater than or equal to r5 time, and *ti_subtask*_1 is true. Every constraint subtask is declared as a Boolean constant. Changes compared to unconstrained tasks are highlighted in blue.

feasible[5], meaning that the non-determinism can be solved to create a *plan* for which each robot knows exactly what to do at every time step and that the plan fulfils all constraints. We check for *feasibility* by assessing:

$$P_{max=?}[F \; done] \tag{5}$$

If there is a path that reaches label *done*, then this formula evaluates to 1, and 0 otherwise. For feasible solutions, we can compute the optimisation objectives depicted in Figure 8c: finding the minimal probability of success, minimising the idle time and the travelling cost. Note that the first formula evaluates the probability of completing a mission without a failure, which is equivalent to the multiplication of the probabilities of each robot completing its tasks; and the last formula evaluates the travelling cost, which can be computed in advance as describe in Figure 8a.2.

3.4.3   Modelling task constraints

Joint tasks. *Joint tasks* require two or more robots to meet at the same space and time. We model joint tasks using two PRISM transitions. In each module, the robots travel to the location of the joint task, JTi. If the robot is to synchronize in time with the other robots ($r_1$time=$r_2$time=...), synchronous transitions among their modules are possible via a common *action* so that all robots perform the task. Figure 9a shows robot *r*1 with the action label *JT*1.

Ordered tasks. We model *ordered tasks* in PRISM as single transitions, in the same way as unconstrained tasks. Then, we modify the guard of the *ith* subtask of the ordered task ($\forall i > 1$), to check if its

---

[5]The feasibility of a permutation of tasks does not necessarily mean that the robot schedule is feasible. For example, assuming that the permutation of tasks assigned to robot r1 is correct, if the available time is set to 5 and the robot requires 6 time units travelling, the schedule is unfeasible.

time is greater than or equal to the robot's time assigned with the subtask $i - 1$; as well as checking if the subtask $i - 1$ itself has been completed. This solution is illustrate in Figure 9b.

3.5 Optimisation of feasible solutions

In the previous section, we described the assembly of an MDP model for a group of robots, each of which with a pre-defined sequence of allocated tasks, we used probabilistic model checking to obtain policies satisfying three optimisation objectives: maximisation of the probability of success, and minimisation of the idle time and travelling cost, and we described the assessment of the feasibility of a schedule. We solve the problem of obtaining the Pareto-optimal solutions of this constrained multi-objective optimization problem using a elitist genetic algorithm. Each a chromosome represents an allocation and task permutation as a single gene. The Pareto front of a such a problem comprises all *non-dominated solution*s, i.e., all solutions for which there is no other feasible solution that can improve one of the optimisation objectives without worsening one or more of the others [10].

Genetic algorithms (GA) provide an effective approach for solving complex optimisation problems. They operate by encoding a number of solutions (population) whose elements (chromosomes) consist of one or more values, called *genes*, that are genetically bred through the application of the Darwinian principles of survival and reproduction of the fittest solutions, and their recombination (croosover) to create new populations (offspring) that *evolve* towards an optimal solution [25]. For KANOA, we adopt the widely used NSGA-II (nondominated sorting genetic algorithm II) with chromosomes that encode both the selected allocation and the selected permutation of robot tasks. Furthermore, we use the EvoChecker [17, 15] framework (which integrates NSGA-II with the PRISM probabilistic model checker) to obtain the Pareto front of task scheduling solutions.

4   Implementation

We developed a prototype tool that implements KANOA. The input is a .mydsl file with the *problem specification* written in our DSL. For the paths in the world model, we use the established probabilistic roadmaps (PRM) path planner to find the distances between locations in the world model, rounding the distances up to the closest integer (see Figure 10) [11].

We used Ecore and Xtend [27] to check the DSL syntax and to automatically generate the *Alloy model*. As described in Section 3.2.1, Alloy is a declarative language for the specification of systems under a set of constraints supported by the Alloy Analyzer solver tool. Hence, Alloy Analyzer gen-

Figure 10: Path from the initial location of robot r2 to room 6 found using PRM

erates up to N allocation files. For each allocation, the *transitive closure* algorithm finds clusters of robots sharing task constraints.

The JMetal 5 [23] framework was used to solve the optimisation of the permutation of tasks using the NSGA-II algorithm. We use the PRISM model checker [26] in the NSGA-II *evaluation* stage, to compute the values of the three objective variables: probability of success, idle time and travelling cost; as well as assessing if a schedule is feasible under the time and space constraints. The KANOA open-source code is available from our GitHub repository at `https://github.com/Gricel-lee/Kanoa`.
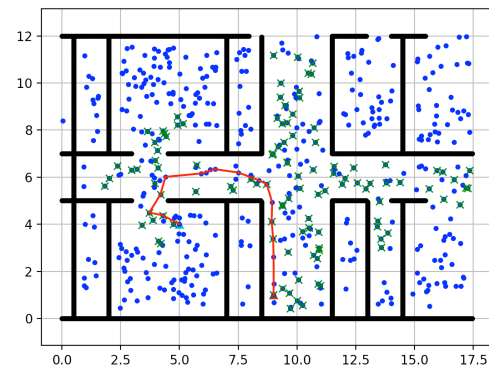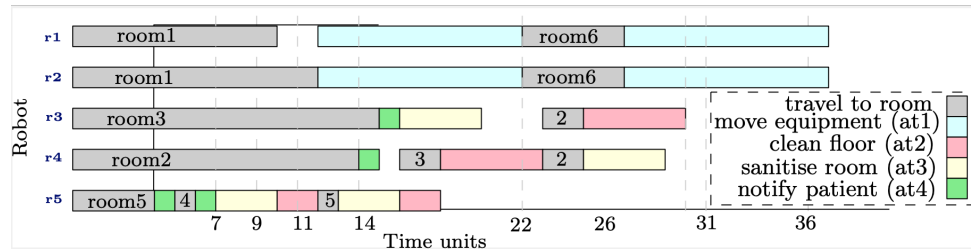
Figure 12: Example of a scheduled plan synthesized by KANOA. All results were obtained on a Mac-Book Air computer with Apple M1 processor, macOS Monterey 12.3.1, and 8 GB of memory.

5   Evaluation

In this section, we evaluate KANOA applicability using the hospital motivating scenario, and the efficiency of the approach varying the number of tasks, robots and complexity of the tasks.
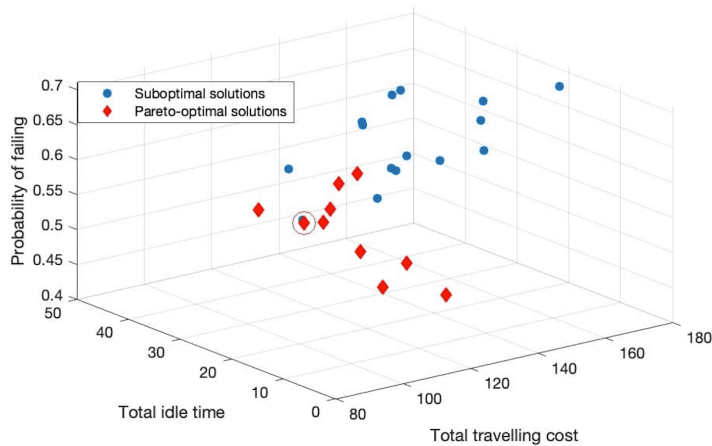

Figure 11: Solutions found by KANOA for the hospital case study. Pareto-front solutions are coloured in red. The schedule of the solution circled in red, is depicted in Figure 12, with values corresponding to the first row of Table 1.

Application to the motivating example. To evaluate KANOA, we applied it to the hospital maintenance mission from our motivating example. The DSL-encoded specification of this mission is provided in Figure 4 earlier in the paper. This specification was supplied as input to KANOA tool. Internally, the Alloy analyzer is configured to generate 30 allocations models. One of these models is depicted in Figure 1 earlier in the paper.

Pre-scheduling was then applied to each of these task allocations, yielding between two and three subsets of robots with one to three robots in each. The KANOA genetic algorithm was configured to run for 5 iterations with a population of 50 chromosomes. Each of the 30 allocation was allowed to compute 20 permutations, giving a total of 600 possible chromosomes. The idle time, travelling cost and probability of failing to execute the mission for the Pareto-optimal solutions evaluated across the task allocations are shown in Figure 11, and one of these solutions is detailed along the timeline in Figure 12. We note that: (i) the failure probability was used instead of the success probability because multiobjective optimisation genetic algorithms operate by minimising all objectives; and (ii) that the values obtained for the failure probability are unacceptably high, which is due to using randomly selected failure probabilities for the individual robots in our experiments. Nevertheless, neither of these issues affects the validity of the experimental results, which were only intended to assess the ability of KANOA to produce Pareto-optimal plans for MRS missions.

From the Pareto-optimal solutions detailed in Table 1, four solutions (rows 1, 3, 6 and 10) achieve very low travelling cost and the idling time at the expense of high failure probabilities. The rest of the Pareto-optimal solutions correspond to solutions that achieve much lower probabilities of failure (with

values below 0.55) in exchange for completing the mission with additional travelling cost and idling time.

Table 1: Pareto-optimal solutions

	Alloc ation	Permu tation	Probability of failure	Idling	Travel
1	17	9	0.63423811	6	80
2	5	5	0.51937429	39	117
3	17	5	0.63423811	5	84
4	16	5	0.53284079	31	126
5	11	1	0.41844289	28	137
6	21	1	0.69771745	2	84
7	11	7	0.41844289	21	145
8	9	8	0.47131171	29	132
9	9	1	0.47131171	22	135
10	12	8	0.70993088	1	88

Table 1 shows the number of allocations and permutations that the Pareto solutions belong to. It is interesting that there is not a single allocation of tasks to robots that seem to perform better than the rest. For instance, allocations 9, 11 and 17 each occur twice. Also notice that the probability of success is the same among the same allocations, however, the idling time and travelling costs change as the permutation changes.

Figure 12 shows a schedule from the Pareto front; in this schedule, all robots were deployed. Robots r1 and r2 first visit room1, perform task at1 (moving medical equipment), travel to room2, and again perform at1. In this case, KANOA found that deploying robots r3 and r4 to clean rooms 2 and 3, and robot r5 to clean rooms 4 and 5 is a Pareto-optimal schedule. Notice that these three cleaner robots travel between the rooms back and forth (see also Figure 1). This is because the distance between rooms 2 to 3 and 4 to 5, is relatively small that it represents a better solution that travelling between other rooms. Although a better solution, where robots avoid repeating rooms, was not found (given the number of permutations and allocations, population and evaluation size), the synthesized solution complies with all requirements and, besides GA, no heuristic guidance was used in the searching process.

Efficiency. We also carried out preliminary experiments to assess the execution time of KANOA. We tested five variants of the hospital scenario in order to assess the impact that (a) adding new tasks and (b) adding more robots have on the time required to find and assess a solution; and (c) the impact that the complexity of the task constraints have on the overall performance. The results obtained for all five variants and for the original case study are shown in Figure 13. The experiments were run until 30 feasible solutions were found or until the time surpassed 300 seconds, whichever occurred first.

Variants v1 and v2 involved using the two pick-and-place robots to move surgical equipment in multiple rooms. For variant v1, they must visit two rooms, and for variant v2 there were six rooms to service. The time to get one feasible solution was 9ms for both variants. However, the time for variant v3 increases to a mean of 4.7s per solution, while for variant v4 KANOA finds a solution every 1.33s. This is an increase of approximately 3.53 times the time to assess a solution when the number of tasks tripled. As v1 and v2 plots show a linear increase, this means that we expect the time to increase approximately 1.17 times for each task added to this hospital scenario.

The third, fourth and fifth variants (v3, v4, and v5) correspond to one, two and three robots available, respectively, with the mission to clean patient rooms 2 to 5. For variant v3, we only found 3 solutions as only one robot has all tasks assigned, making it difficult to synthesize plans that comply with the time limit. The time to find the solutions is 14.86 seconds, this is understandable as, for most permutations, PRISM is called only to check for feasibility and discarded as they do not comply with the requirements. Variants v4 and v5, increasing v3 by one and two robots, show a considerable increase in the time to find a solution. This is understandable as the complexity of the PRISM models increases, taking more time for the PMC to check for feasibility and to return the probability of failure, idling time and travelling cost.
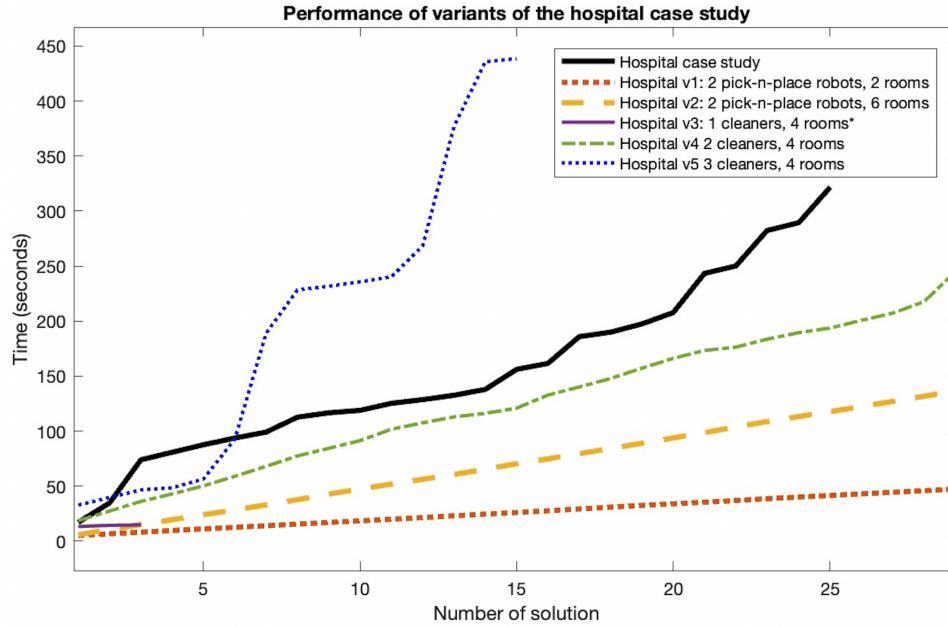
Figure 13: Scheduled plan synthesized by KANOA; all solutions were computed with 30 allocations, 20 permutation of tasks, with population size 50 and 5 evaluations (except variants v3, which was tested with 500 permutations)

6   Related Work

Work in other domains, such as manufacturing [36] and distributed computer systems [35] has influenced the algorithms used to solve allocation and scheduling problems in robotics. Some of the most broadly researched categories are mixed-integer linear programming (MILP), and hybrid methods such as MILP constraint programming and MILP depth-first search heuristics [7]. These approaches exhibit desirable traits such as scalability, but often require crafting application-dependent heuristics by domain-experts to become scalable, simplify task complexity to concurrent and temporal constraints of simple tasks, and do not consider uncertainty [40, 18, 6].

On the other hand, model checking has successfully been applied in MRS to provide guarantees for the synthesised robot schedules. To solve the allocation and scheduling problems, these can be modelled together or separately. There is not a clear preference adopted by the research community. For instance, some research only focuses on one of these problems. Menghi et al. [8] used model checking for scheduling when only incomplete knowledge is available; and Yu et al. [39] consider constraints due to the robots sharing a room or hall, and limited local information for re-planning online. When allocation and scheduling are modelled together, we notice the introduction of optimisation metrics. For instance, Antlab [14] ranks the schedules by travelling cost, and Ulusoy et al. [3] optimise the time for the completion of all tasks. In terms of probabilistic model checking, Lacerda et al. [5] presents a single-robot planning problem to perform a series of tasks. However, they consider the tasks ordered in advance, and PMC only used to select a path amount pre-define paths. Others use PMC for the synthesis of MRS controllers without considering [9], inferring the allocation and scheduling as a prerequisite.

KANOA extends our preliminary work from [19] (where we advocate the use of this type of approach

in a doctoral symposium paper) and [20] (where we assess the feasibility of applying Alloy analyzer and PRISM), and tackles the allocation and scheduling of tasks separately. The information about the robot capabilities and mission tasks is used for both the allocation and the scheduling of tasks, allowing us to modify how the schedules are done by grouping robots from the allocations found, thus reducing the complexity of the models used in the KANOA's scheduling stage. We also assume that the path planning is performed in advance, as adding the planning into the model (e.g., as in [14]) requires adding each intermediate point, increasing the states and transitions of the model and reducing scalability. In further work, we will reduce the state explosion problem by adding middle points where necessary, for instance, in the hospital halls to avoid two robots entering the same location at the same time. Other interesting MRS characteristics were also found in the state-of-the-art. Antlab [14] captures dynamic collision avoidance, and MALTA [21] models road conditions that may slow down robots. MRS must deal with a large range of uncertainty sources.

Although the ultimate goal for *multi-robot planning, scheduling and allocation* may be the creation of a single tool that is applicable to all the variants of MRS applications, this will be very challenging to achieve (and may not be feasible) because there are too many applications with different critical characteristics to model. For example, for search-and-rescue missions, the robots must be able to work in unknown environments; while in a factory, robots may have a map in advance, although they must be able to avoid staff and cooperate with operators. In further work, we would like to incorporate further task constraints, such as time windows (for example, serving meals to the patients between 13:00 and 14:00), and consecutive tasks (tasks that must be perform one after the other).

7   Conclusion

We introduced a new end-to-end approach for the task allocation and scheduling of multi-robot missions comprising joint and ordered tasks that need to be executed by teams of robots with different capabilities. Three different optimisation objectives and a feasibility constraint were use to find Pareto-optimal solutions. We support our end-to-end approach (from the high-level problem definition to the synthesis of robot plans) with an open-source tool and a preliminary evaluation of a hospital case study. We presented five different variants of the MRS mission from this case study, assessing the impact of adding new tasks and robots to the MRS mission.

In future work, we will explore options for improving the scalability of KANOA, and for gracefully degrading the constraints defined in the KANOA problem specifications when no plan can be found when the initial constraints are taken into account. We also aim to enhance KANOA with self-adaptation capabilities, enabling the dynamic evolution of the generated plans after disturbances such as the failure of a robot, the failure of a robot's sensors/actuators, or a delay in the execution of a task by one of the robots. Moreover, longer runs will be used in the evaluation of our future work to establish whether a longer search produces significantly better solutions. Additionally, we will explore the inclusion of spatial restrictions, such as "only one robot can pass through a hall at a time" or "no more than two robots are allowed in a room at the same time". Moreover, as we deal with autonomous mobile robots, we need to introduce an event when the battery is low so that the robot can reach a charging station.

Understanding the computational complexity of our approach is another area of future work for the project. Analysing the complexity of the approach is nontrivial, as it depends on the Alloy Analyser solver, PRISM engine and JMetal configuration, and, as described in Section 5, it also depends on the number of robots, number of tasks and task dependencies.

References

[1] EBNF: ISO/IEC 14977 (1996): *Standard EBNF Syntaxt Specification*. Available at `http://www.cl.cam.ac.uk/mgk25/iso-14977.pdf`.

[2] Khamis Alaa, Hussein Ahmed & Elmogy Ahmed (2015): *Multi-robot task allocation: A review of the state-of-the-art*. Cooperative robots and sensor networks, pp. 31–51, doi:10.1007/978-3-319-18299-5_2.

[3] Ulusoy Alphan, Smith Stephen L, Ding Xu Chu, Belta Calin & Rus Daniela (2011): *Optimal multi-robot path planning with temporal logic constraints*. In: *IEEE/RSJ IROS*, IEEE, p. 3087–3092, doi:10.48550/arXiv.1107.0062.

[4] Andrea Bianco & Luca de Alfaro (1995): *Model checking of probabilistic and nondeterministic systems*. In: *International Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS*, Springer, pp. 499–513, doi:10.1007/3-540-60692-0_70.

[5] Lacerda Bruno, Faruq Fatma, Parker David & Hawes Nick (2019): *Probabilistic planning with formal performance guarantees for mobile service robots*. The International JOURNAL of Robotics Research 38(9), doi:10.1177/0278364919856695.

[6] Radu Calinescu (2013): *Emerging techniques for the engineering of self-adaptive high-integrity software*. In: *Assurances for Self-Adaptive Systems*, Springer, pp. 297–310, doi:10.1007/978-3-642-36249-1_11.

[7] Jiaqiong Chen & Ronald G Askin (2009): *Project selection, scheduling and resource allocation with time dependent returns*. European JOURNAL of Operational Research 193(1), pp. 23–34, doi:10.1016/j.ejor.2007.10.040.

[8] Menghi Claudio, Garcia Sergio, Pelliccione Patrizio & Tumova Jana (2018): *Multi-robot LTL planning under uncertainty*. In: *International Symposium on Formal Methods*, Springer, p. 399–417, doi:10.1007/978-3-319-95582-7_24.

[9] Fraser Douglas, Giaquinta Ruben, Hoffmann Ruth, Ireland Murray, Miller Alice & Norman Gethin (2020): *Collaborative models for autonomous systems controller synthesis*. Formal Aspects of Computing 32(2), doi:10.1007/s00165-020-00508-1.

[10] Juan J Durillo & Antonio J Nebro (2011): *jMetal: A Java framework for multi-objective optimization*. Advances in Engineering Software 42(10), pp. 760–771, doi:10.1016/j.advengsoft.2011.05.014.

[11] Kavraki Lydia E, Svestka Petr, Latombe J-C & Overmars Mark H (1996): *Probabilistic roadmaps for path planning in high-dimensional configuration spaces*. IEEE transactions on Robotics and Automation 12(4), doi:10.1109/ROBOT.1996.509171.

[12] Sukkar Fouad, Best Graeme, Yoo Chanyeol & Fitch Robert (2019): *Multi-robot region-of-interest reconstruction with Dec-MCTS*. In: *ICRA*, p. 9101–9107, doi:10.1109/ICRA.2019.8793560.

[13] Sergio Garcia, Patrizio Pelliccione, Claudio Menghi, Thorsten Berger & Tomas Bures (2020): *Promise: high-level mission specification for multiple robots*. In: *Proceedings of the ACM/IEEE 42nd ICSE: Companion Proceedings*, p. 5–8, doi:10.1145/3377812.3382143.

[14] Ivan Gavran, Rupak Majumdar & Indranil Saha (2017): *Antlab: A multi-robot task server*. ACM Transactions on Embedded Computing Systems (TECS) 16(5s), pp. 1–19, doi:10.1145/3126513.

[15] Simos Gerasimou, Radu Calinescu & Giordano Tamburrelli (2018): *Synthesis of probabilistic models for quality-of-service software engineering*. Automated Software Engineering 25(4), pp. 785–831, doi:10.1007/s10515-018-0235-8.

[16] Simos Gerasimou, Javier Cámara, Radu Calinescu, Naif Alasmari, Faisal Alhwikem & Xinwei Fang (2021): *Evolutionary-guided synthesis of verified Pareto-optimal MDP policies*. In: *36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, IEEE, pp. 842–853, doi:10.1109/ASE51524.2021.9678727.

[17] Simos Gerasimou, Giordano Tamburrelli & Radu Calinescu (2015): *Search-based synthesis of probabilistic models for quality-of-service software engineering (t)*. In: *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, IEEE, pp. 319–330, doi:10.1109/ASE.2015.22.

[18] Matthew Gombolay, Ronald Wilcox & Julie Shah (2013): *Fast scheduling of multi-robot teams with temporospatial constraints.* Robotics: Science and Systems Foundation, doi:10.1109/TRO.2018.2795034.

[19] Vazquez Gricel (2021): *Automated Scheduling of Multi-Robot System Missions: An Architectural Perspective.* In: *ECSA'21 Doctoral Symposium*, pp. 1–5.

[20] Vázquez Gricel, Calinescu Radu & Cámara Javier (2021): *Scheduling multi-robot missions with joint tasks and heterogeneous robot teams.* In: *TAROS*, Springer, p. 354–359, doi:10.1007/978-3-030-89177-0_36.

[21] Rong Gu, Eduard Baranov et al. (2021): *Mission Planning for Multiple Autonomous Agents under Complex Road Conditions: Model-Checking-Based Synthesis and Verification.* Mälardalen University Technical Report.

[22] Hans Hansson & Bengt Jonsson (1994): *A logic for reasoning about time and reliability.* Formal aspects of computing 6(5), pp. 512–535, doi:10.1007/BF01211866.

[23] Nebro Antonio J, Durillo Juan J & Vergne Matthieu (2015): *Redesigning the jMetal multi-objective optimization framework.* Proceedings of the companion publication of the annual conference on genetic and evolutionary computation, doi:10.1145/2739482.2768462.

[24] Daniel Jackson (2002): *Alloy: a lightweight object modelling notation.* ACM Transactions on Software Engineering and Methodology 11(2), pp. 256–290, doi:10.1145/505145.505149.

[25] John R Koza (1994): *Genetic programming as a means for programming computers by natural selection.* Statistics and computing 4(2), pp. 87–112, doi:10.1007/BF00175355.

[26] Marta Kwiatkowska, Gethin Norman & David Parker (2011): *PRISM 4.0. Verification of probabilistic real-time systems.* In: *International conference on computer aided verification*, Springer, pp. 585–591, doi:10.1007/978-3-642-22110-1_47.

[27] Bettini Lorenzo (2016): *Implementing domain-specific languages with Xtext and Xtend.* Packt Publishing Ltd.

[28] Fox Maria & Long Derek (2003): *PDDL2. 1: An extension to PDDL for expressing temporal planning domains.* JOURNAL of artificial intelligence research 20, doi:10.1613/jair.1129.

[29] Askarpour Mehrnoosh, Tsigkanos Christos, Menghi Claudio, Calinescu Radu, Pelliccione Patrizio, Garcia Sergio, Caldas Ricardo, von Oertzen Tim J, Wimmer Manuel, Berardinelli Luca et al. (2021): *RoboMAX: Robotic Mission Adaptation eXemplars.* In: *SEAMS*, IEEE, p. 245–251, doi:10.1109/SEAMS51251.2021.00040.

[30] Georg Nollert & Sabine Wich (2008): *Planning a Cardiovascular Hybrid Operating Room. the Technical Point of View.* In: *The heart surgery forum*, HSF, p. 1033, doi:10.1532/HSF98.2009103.

[31] Warren Jr Henry S (1975): *A modification of Warshall's algorithm for the transitive closure of binary relations.* Communications of the ACM 18(4), doi:10.1145/360715.360746.

[32] Tran Tony T, Tiago, Goldie Nejat & Beck J Christopher (2017): *Robots in retirement homes: Applying off-the-shelf planning and scheduling to a team of assistive robots.* JOURNAL of Artificial Intelligence Research 58, pp. ,523–590, doi:10.1613/jair.5306.

[33] Gricel Vázquez, Radu Calinescu & Javier Cámara (2022): *KANOA GitHub repository.* Available at `https://github.com/Gricel-lee/Kanoa`.

[34] Forejt Vojtěch, Kwiatkowska Marta, Norman Gethin & Parker David (2011): *Automated verification techniques for probabilistic systems.* In: *International school on formal methods for the design of computer, communication and software systems*, Springer, p. 53–113, doi:10.1007/978-3-642-21455-4_3.

[35] Joel Wolf, Nikhil Bansal, Kirsten Hildrum, Sujay Parekh, Deepak Rajan, Rohit Wagle, Kun-Lung Wu & Lisa Fleischer (2008): *SODA: An optimizing scheduler for large-scale stream-based distributed computer systems.* In: *ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing*, Springer, pp. 306–325, doi:10.1007/978-3-540-89856-6_16.

[36] TN Wong, CW Leung, Kai-Ling Mak & Richard YK Fung (2006): *Dynamic shopfloor scheduling in multi-agent manufacturing systems*. Expert Systems with Applications 31(3), pp. 486–494, doi:10.1016/j.eswa.2005.09.073.

[37] Jim Woodcock, Peter Gorm Larsen, Juan Bicarregui & John Fitzgerald (2009): *Formal methods: Practice and experience*. ACM computing surveys (CSUR) 41(4), doi:10.1145/1592434.1592436.

[38] Carreno Yaniel, Pairet Eric, Petillot Yvan & Petrick Ronald PA (2020): *Task allocation strategy for heterogeneous robot teams in offshore missions*. In: Proceedings of AAMAS'19, pp. 222–230, doi:10.5555/3398761.3398792.

[39] Pian Yu & Dimos V Dimarogonas (2021): *Distributed motion coordination for multirobot systems under LTL specifications*. IEEE Transactions on Robotics 38(2), pp. 1047–1062, doi:10.1109/TRO.2021.3088764.

[40] Wang Zheyuan & Matthew Gombolay (2020): *Learning scheduling policies for multi-robot coordination with graph attention networks*. IEEE Robotics and Automation Letters 5(3), pp. 4509–4516, doi:10.1109/LRA.2020.3002198.