A Framework for Simultaneous Task Allocation and Planning under Uncertainty

FATMA FARUQ, University of Birmingham, UK
BRUNO LACERDA, University of Oxford, UK
NICK HAWES, University of Oxford, UK
DAVID PARKER, University of Oxford, UK

We present novel techniques for simultaneous task allocation and planning in multi-robot systems operating under uncertainty. By performing task allocation and planning simultaneously, allocations are informed by individual robot behaviour, creating more efficient team behaviour. We go beyond existing work by planning for task reallocation across the team given a model of partial task satisfaction under potential robot failures and uncertain action outcomes. We model the problem using Markov decision processes, with tasks encoded in co-safe linear temporal logic, and optimise for the expected number of tasks completed by the team. To avoid the inherent complexity of joint models, we propose an alternative model that simultaneously considers task allocation and planning, but in a sequential fashion. We then build a joint policy from the sequential policy obtained from our model, thus allowing for concurrent policy execution. Furthermore, to enable adaptation in the case of robot failures, we consider replanning from failure states and propose an approach to preemptively replan in an anytime fashion, replanning for more probable failure states first. Our method also allows us to quantify the performance of the team by providing an analysis of properties such as the expected number of completed tasks under concurrent policy execution. We implement and extensively evaluate our approach on a range of scenarios. We compare its performance to a state-of-the-art baseline in decoupled task allocation and planning: sequential single-item auctions. Our approach outperforms the baseline in terms of computation time and the number of times replanning is required on robot failure.

CCS Concepts: • **Computing methodologies → Planning under uncertainty**; **Multi-agent planning**; **Robotic planning**.

Additional Key Words and Phrases: Markov decision processes, linear temporal logic, multi-robot systems

ACM Reference Format:
Fatma Faruq, Bruno Lacerda, Nick Hawes, and David Parker. 2024. A Framework for Simultaneous Task Allocation and Planning under Uncertainty. *ACM Trans. Autonom. Adapt. Syst.* 1, 1, Article 1 (January 2024), 29 pages. https://doi.org/10.1145/3665499

1 INTRODUCTION

In many service robot applications, such as intra-logistics, surveillance or stock monitoring, it is desirable for a collection of *tasks* to be allocated to a team of robots. The use of a team of robots allows for the tasks to be completed in a way that is robust against failures of individual robots, and allows the number of tasks requested to be dynamic as long as resources exist in the team to service them. In this paper, we address applications such as these where tasks are independent (i.e., there are no inter-task dependencies) and where each task only requires a single robot to complete it. Most existing approaches for solving this class of problems divide the problem into separate *task allocation* (TA) and *planning* processes. TA determines which robot should complete which tasks, and planning determines how each task,

Authors' Contact Information: Fatma Faruq, ffaruq@gmail.com, University of Birmingham, Birmingham, UK; Bruno Lacerda, University of Oxford, Oxford, UK, bruno@robots.ox.ac.uk; Nick Hawes, University of Oxford, Oxford, UK, nickh@robots.ox.ac.uk; David Parker, University of Oxford, Oxford, UK, david.parker@cs.ox.ac.uk.

Manuscript submitted to ACM

or conjunction of tasks, should be completed. This separation is made to reduce the computational complexity of the problem. It allows each robot to plan separately for its own task set, avoiding the need for a joint planning model which is typically exponential in the number of team members. It also allows specialised algorithms to be used for the TA and planning parts, increasing the efficiency with which the task-directed behaviour of the team can be generated. TA usually assumes a greatly simplified model of planning in order to be able to efficiently compute allocations. However, this also means that the TA process cannot be informed by the plans of the individual robots, which prevents it from exploiting opportunities, or avoiding hindrances, that are only evident once planning has been performed. For example, if the individual robots plan with time-based models, a task may be much quicker to complete at a particular time of day, but with TA separated from planning, this information cannot be exploited in the allocation process.

To address this limitation, recent work has considered the problem of *simultaneous task allocation and planning* (STAP) [52, 54], which solves the complete problem in a single process, and can therefore take the plans of each robot (and their costs etc.) into account during the allocation process. The STAP problem is challenging due to the need to search for solutions (task allocations and associated single-agent plans) over the joint space of possible allocations and multi-agent action choices. In [19], we introduced an extension of STAP called STAPU, which considers *uncertainty* in the action outcomes of the individual agents. In particular, we model the probability that individual robots suffer permanent *failures* meaning that they can no longer execute tasks. In this paper, we significantly extend this work by: (i) tackling an alternative problem formulation which allows the team to continue completing tasks even when some have become unachievable, improving robustness; (ii) proposing a novel anytime planning approach based on task reallocation in cases of robot failure, allowing the team to adapt to individual robot failures; and (iii) extending the modelling formalism to incorporate shared environmental state features.

To enable the specification of intricate goal behaviours more naturally, and in order to produce guarantees on the quality of the multi-robot plans that we generate, our formulation of the STAPU problem adopts languages and techniques from the field of formal verification. Specifically, we make use of linear temporal logic (LTL), a powerful and intuitive formalism that is widely used to specify a variety of robot tasks. Each task to be completed by the team is specified in the *co-safe* fragment of LTL, and a *safe* LTL formula is used to specify safety constraints to be obeyed by all robots as they achieve these tasks. To model how the robots can achieve these specifications, individual robots' capabilities and environments are described using Markov decision processes (MDPs). Building upon techniques and tools to synthesise provably correct policies from an MDP model of the system and an LTL task specification [33, 34], we propose a method that generates multi-robot policies which maximise the expected number of tasks completed before the safety constraint is violated.

We tackle the basic task allocation and planning problem using a *team MDP* that exploits the independence between tasks and adopts a sequential modelling approach to build policies for each robot. We then iteratively improve these policies by incorporating the possibility of reallocating tasks in the event of individual robot failures. To do so, we construct a joint model of the synchronised execution of the individual robot policies in order to identify states where robots might fail, and build new team policies from those states, thus providing *probabilistic guarantees* on the performance (and safe operation) of the team of robots, along with an efficient task reallocation mechanism. We implement our approach as an extension of the probabilistic model checker PRISM [33], and evaluate its performance on a variety of representative multi-robot problems.

In comparison to [19], we introduce a number of significant improvements. Firstly, the previous work considered the problem of maximising the probability of all tasks being completed by the team. Our new approach tackles the issue of *partial mission satisfaction*, i.e., tasks not being achievable with probability 1, in a more principled and flexible way by posing the objective as finding a team policy that, in expectation, achieves as many tasks as possible. This alternative formulation allows the team to continue to complete tasks even when some have become unachievable, a behaviour that is not attainable under the objective addressed in [19]. In addition, we further explore the coupling within the multi-robot system by extending our method to deal with *shared environmental state features*. These are global state features which may be observed by one robot during task execution (e.g., whether a door is open or closed), but can then be exploited by another robot subsequently (e.g., the second robot does not need to re-observe the door status). Moreover, we substantially improve the formalisation of the solution. In particular we provide a more precise formalisation of the underlying team model, introducing a *joint MDP* with shared state features. Building on this, we improve the description of the problem statement, the method for building joint policies, and the approach for task reallocation. The latter provides a novel *anytime planning approach* to deal with cases of robot failure. Finally, we extend the evaluation of our work, comparing its performance with an auction-based STAPU baseline which is representative of state-of-the-art solutions to this problem. In this evaluation we demonstrate that our novel method outperforms the baseline in computation time for allocation, and the number of reallocation events needed, whilst additionally providing guarantees on the expected value of number of tasks completed at each state.

2 RELATED WORK

Within the existing literature on multi-robot systems, we can consider the following distinctions: 1) *multi-agent path finding* (MAPF) approaches, which have rich models of inter-agent spatial interactions but can only solve path planning problems, versus more general planning approaches which can reason about a greater range of tasks; 2) planning approaches which explicitly model *uncertainty*, versus those with deterministic models; 3) approaches which produce solutions using *verification*-based methods (and thus can produce formal guarantees on behaviour), versus other solution approaches; and 4) approaches which *integrate* task allocation and planning, versus those that separate these processes.

MAPF [20] is a widely-studied problem which focuses on ensuring efficient, collision-free movement of a robot team through an environment. By focusing purely on path finding, domain-specific heuristics and algorithms can be used to efficiently solve larger problem instances than would otherwise be possible [56]. A more general class of problems is multi-agent planning, which allows robot actions to have preconditions and effects across state features in the problem, and can therefore represent a wider range of robot tasks [61, 66]. These typically focus on problems where interactions and coordination between agents are required to solve a single task (e.g., one robot needs to place an object onto a second robot), but these techniques do not usually involve explicit allocation of tasks as is required in a STAPU problem.

Multi-robot task allocation (MRTA) is a canonical problem in multi-robot systems research [47] and as such has been addressed from various perspectives. Good surveys on this topic are [24, 31]. These also propose a taxonomy for MRTA. The problem tackled in this paper can be framed within the MRTA taxonomy as an instance of the ID [ST-SR-TA] class. In words, we consider robots that can execute at most one task at a time (ST), and tasks that require a single robot to be completed (SR) and take some time to be executed (TA). Furthermore, given that the use of LTL specifications allow for introducing subtask dependencies, we are considering problems that have intra-schedule dependencies (ID).

A number of approaches have looked into combining task allocation and planning without uncertainty (i.e., STAP) [18, 29, 42, 43, 46, 50, 52, 54]. The work of Schillinger et al. [52, 54] is of particular relevance for this paper. They use a logical model of robot operation plus a team task specification in LTL and propose an algorithm that plans and allocates tasks to robots in order to minimise the maximum cost any robot will incur to complete its tasks. To overcome the complexity of reasoning in a space which grows exponentially in the number of robots, the authors propose an approach where separate planning models for each robot are linked sequentially by *switch transitions*, which allow one robot to pass tasks to the next robot in the team. They exploit these transitions to produce multi-robot plans which allocate tasks across the team in order to minimise the aforementioned solution metric of minimising the largest robot cost. Our work takes this approach as an inspiration, but modifies it extensively to reason over uncertainty in the effects of robot actions.

Other works have considered LTL in the context of MRTA in different ways. [29] uses Petri nets to model the environment. This keeps the model structure constant. They then use LTL to specify the mission and provide a solution using mixed integer linear programming (MILP). The size of the MILP problem solved is exponential in the number of atomic propositions in the specification, hindering scalability of the method. Similarly, [35] considers a Petri net model of a multi-robot system and avoids scalability issues by enforcing safe LTL rules using ideas from supervisory control theory. However, they can only disable actions to maintain the system safe, and cannot plan to reach goals. [50] considers an extension of LTL that allows specification of how many robots should execute a given task. The problem is solved using an ILP formulation. For teams of homogeneous robots, they are able to make their approach scale to large numbers of robots. However, the number of tasks contained in their specification is typically small. Similarly, [13] consider LTL-based control of swarms of robots. [42, 46] avoid enumerating the full state-space for STAP problems by using sampling-based methods to allocate tasks and generate policies. They differ in the type of sampling method used, however both use finite transition systems to model agents, and LTL for mission specification. The solution quality and time of such algorithms is variable and they rely on heuristics which may be hard to compute. Finally, [18] consider the problem of distributing a new set of LTL tasks across a group of heterogeneous robots which are already performing a set of tasks they have been allocated previously.

The various planning fields surveyed above also have analogues which include uncertainty. MAPF approaches have included uncertainty to account for the performance of mobile robot localisation and navigation reliability [3, 44, 60, 64]. Many single robot planning approaches assume that the environment is fully observable but responds probabilistically to robot actions and thus formulate planning problems using MDPs [38, 57]. Approaches in this space include our prior work [34, 37] which uses verification-based methods to produce probabilistically-guaranteed behaviour policies for a mobile robot, where elements of the MDP are learnt from experience. Other approaches use sets of LTL specifications to define *non-Markovian rewards* over an MDP [5, 11, 27, 55]. The optimisation objective we consider takes a similar approach, aiming to maximise the expectation of a non-Markovian reward function, but adding a hard safety constraint. Furthermore, our setting is one of MRTA, where the tasks must be distributed amongst robots, whilst the aforementioned approaches consider a set of tasks to be executed by a single robot [5, 11, 27], or a set of uncertain specifications, where there is a belief over which task is the actual objective [55].

When extending MDP planning approaches to multi-robot settings, authors either: assume communication and sparse interactions between robots in order to maintain full observability and mitigate scalability issues [4, 14–17, 21, 22, 45, 51]; resort to auctioning approaches for TA, thus keeping the planning over single robot models [12, 53, 58]; or otherwise use the computationally-demanding decentralised, partially-observable MDP formalisation which accounts for the unknown state of other robot in the problem [1, 2]. As is appropriate in many service robot domains, we make the assumption of perfect communication, thus allowing this work to retain the MDP formalisation.

Our work addresses a planning scenario where it may be impossible to achieve the entire mission due to events in the environment or the choices of the agents themselves. However, we assume that it is beneficial to achieve as much of the mission as possible. Existing work addressing such partial satisfaction includes [37, 39] who use co-safe LTL to create metrics to guide planning for single agents in order to maximise the amount of a partial task achieved. Unlike [39], the metric in [37] is independent of user input. This is useful in scenarios where end users do not have complete access to the robot's mission. Alternative approaches to partial satisfaction include using soft constraints. For example, [63] generates a minimum violating plan for an autonomous vehicle using a horizon-based planner and rapidly-exploring random trees (RRTs) [40]. They specify constraints in LTL and a penalty cost is associated with each constraint that can be violated. For scenarios where a robot must repeatedly satisfy some part of the specification, [25] generates a policy which minimises the probability of getting into a state that violates the specification. There are other approaches to multi-robot planning based on formal models and LTL specifications [26, 28]. Like us, these approaches are designed to cope with individual robot failures, but do not explicitly model or quantify failure probabilistically in the way we do.

Whilst there is work on TA under uncertainty [15], there is very little existing literature which addresses STAP under uncertainty. [14] deals with TA and MAPF under uncertainty in a warehouse environment. The focus is on scalability and thus [14] only addresses pathfinding and uses an approximate solution method which cannot provide probabilistic guarantees. [23] addresses STAP but uses online policy execution to deal with unmodelled environmental disturbances. This also fails to provide guarantees over behaviour in advance, and cannot optimise for objectives over the team behaviour. In contrast, the work in this paper allows for rich LTL task specifications, and solves the problem exactly to provide probabilistic guarantees over the resulting behaviour.

3 PRELIMINARIES

3.1 Markov Decision Processes

We use *Markov decision processes* (MDPs) to model the evolution of robots and their environment.

Definition 1 (MDP). An *MDP* is a tuple of the form $\mathcal{M} = \langle S, \overline{s}, A, \delta, AP, L \rangle$, where:

- S is a finite set of states;
- $\overline{s} \in S$ is the initial state;
- A is a finite set of actions;
- $\delta : S \times A \times S \to [0, 1]$ is a probabilistic transition function, where $\forall s \in S, a \in A : \sum_{s'} \delta(s, a, s') \in \{0, 1\}$;
- AP is a set of atomic propositions;
- $L : S \to 2^{AP}$ is a labelling function, such that $p \in L(s)$ if and only if p is true in $s \in S$.

In each state s of an MDP $\mathcal{M}$, there is a decision between the actions that are *enabled* in s, i.e., those in the set $A_s = \{a \in A \mid \delta(s, a, s') > 0 \text{ for some } s' \in S\}$. If action $a \in A_s$ is taken in state s, then the probability that the next state is s' is given by $\delta(s, a, s')$. A sequence of such transitions $\sigma = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \dots$ where $\delta(s_i, a_i, s_{i+1}) > 0$ for all $i \in \mathbb{N}$ is an (infinite) *path* through the MDP. A *finite path* $\rho = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \dots \xrightarrow{a_{n-1}} s_n$ is a prefix of an infinite path. We denote the sets of all finite and infinite paths of $\mathcal{M}$ starting from state s by $FPath_{\mathcal{M},s}$ and $IPath_{\mathcal{M},s}$.

The choice of action to take at each step of the execution of an MDP $\mathcal{M}$ is made by a *policy*, which can base its decision on the history of $\mathcal{M}$ up to the current state.

Definition 2 (Policy). A *policy* for MDP $\mathcal{M}$ is a function $\pi : FPath_{\mathcal{M},\overline{s}} \to A$ such that, for any finite path ρ ending in state s_n, we have $\pi(\rho) \in A_{s_n}$.

In this work, we will use *memoryless* policies $\pi : S \to A$, which only base their choice of action on the current state, and *finite-memory* policies, which track a finite set of "modes" needed, in conjunction with the current state, to choose an action. For a policy π, we can define a probability space $Pr^{\pi}_{\mathcal{M},s}$ over the set of infinite paths $IPath_{\mathcal{M},s}$. Furthermore, for a measurable function $X : IPath_{\mathcal{M},s} \to \mathbb{R}$, we write $E^{\pi}_{\mathcal{M},s}(X)$ for the expected value of X with respect to $Pr^{\pi}_{\mathcal{M},s}$.

We also use MDP *reward structures*, which assign non-negative values to state-action-state triples.

Definition 3 (Reward structure)**.** A *reward structure* for an MDP $\mathcal{M}$ is a function $r : S \times A \times S \to \mathbb{R}_{\geq 0}$.

Of particular interest in this paper is the expected amount of reward accumulated up until a target is reached.

Definition 4 (Expected cumulative reward)**.** For reward structure r on MDP $\mathcal{M}$ and target label $b \in AP$, we define the function $cumul_r^b : IPath_{\mathcal{M},s} \to \mathbb{R}_{\geq 0}$ as:

$$cumul_r^b(s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \dots) = \sum\nolimits_{i=0}^{n_b - 1} r(s_i, a_i, s_{i+1}),$$

where n_b is the first index for which $b \in L(s_{n_b})$. The *expected cumulative reward* under policy π of $\mathcal{M}$ is defined as $E^{\pi}_{\mathcal{M},s}(cumul_r^b)$.

3.2 Linear Temporal Logic

Linear temporal logic (LTL) [48] is an extension of propositional logic which allows reasoning about infinite sequences of states. The syntax of LTL is as follows.

Definition 5 (LTL syntax)**.** LTL formulas φ over atomic propositions AP are defined using the following grammar:

$$\varphi ::= true \mid p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathtt{X}\,\varphi \mid \varphi\,\mathtt{U}\,\varphi, \text{ where } p \in AP.$$

The $\mathtt{X}$ operator is read “next”, meaning that the formula it precedes will be true in the next state. The $\mathtt{U}$ operator is read “until”, meaning that its second argument will eventually become true in some state, and the first argument will be continuously true until this point. The other propositional connectives can be derived from the ones above in the usual way. Moreover, other useful LTL operators can be derived from the ones above. Of particular interest for our work are the “eventually” operator $\mathtt{F}\,\varphi$, which requires that φ is satisfied in some future state; the “always” operator $\mathtt{G}\,\varphi$, which requires φ to be satisfied in all future states; and the “weak until” operator $\mathtt{W}$, which relaxes the until operator to include infinite sequences for which ψ is never satisfied but φ is always satisfied: $\mathtt{F}\,\varphi \equiv true\,\mathtt{U}\,\varphi$; $\mathtt{G}\,\varphi \equiv \neg\,\mathtt{F}\,\neg\varphi$; and $\varphi\,\mathtt{W}\,\psi \equiv (\varphi\,\mathtt{U}\,\psi) \vee (\mathtt{G}\,\varphi)$. Given an infinite path σ, we write $\sigma \models \varphi$ to denote that σ satisfies formula φ.

The semantics of full LTL is defined over infinite paths. However, in this work, we are interested in specifying behaviours that occur within finite time. So, we use two well-known subsets of LTL for which properties are meaningful when evaluated over finite paths: *safe* and *co-safe* LTL [32]. These are based on the notions of *bad prefix* and *good prefix*. A bad prefix for φ is a finite path that cannot be extended in such a way that φ is satisfied, and a good prefix for φ is a finite path that cannot be extended in such a way that φ is *not* satisfied. Safe LTL is defined as the set of LTL formulas for which all non-satisfying infinite paths have a finite bad prefix. Conversely, co-safe LTL is the set of LTL formulas for which all satisfying infinite paths have a finite good prefix. If φ is a formula of safe LTL, then its negation $\neg\varphi$ is co-safe, and vice-versa. For any co-safe LTL formula φ written over AP, we can build a deterministic finite automaton (DFA) equivalent to φ.

Definition 6 (DFA representation)**.** Let φ be a co-safe LTL formula. The DFA representing φ is a tuple $\mathcal{A}_{\varphi} = \langle Q, \overline{q}, Q_F, 2^{AP}, \delta_{\varphi} \rangle$, where:

- Q is a finite set of states;
- $\overline{q} \in Q$ is the initial state;
- $Q_F \subseteq Q$ is the set of accepting states;
- 2^{AP} is the alphabet; and
- $\delta_{\varphi} : Q \times 2^{AP} \to Q$ is a transition function;

such that the language of finite words accepted by $\mathcal{A}_\varphi$ is the set of good prefixes of paths that satisfy φ (or, more precisely, the sequences of state labellings from those paths) [32]. Conversely, if φ is a formula in *safe* LTL, then the DFA $\mathcal{A}_{\neg\varphi}$ for its negation represents the *bad prefixes* of φ.

We will consider the class of syntactically safe and co-safe LTL$_{-\mathtt{X}}$ formulas as our specification language. This is obtained by two syntactic restrictions. We assume that all formulas are in positive normal form (negation can only appear next to atomic propositions). First, we disallow the $\mathtt{X}$ operator. This is due to technicalities related to our mechanism for resolving conflicting actions and will be further explained later. Second, we consider syntactically safe LTL$_{-\mathtt{X}}$, i.e., the set of formulas for which only the $\mathtt{G}$ and $\mathtt{W}$ temporal operators occur, and syntactically co-safe LTL$_{-\mathtt{X}}$, i.e., the set of formulas for which only the $\mathtt{F}$ and $\mathtt{U}$ temporal operators occur. Note that (co-)safe LTL$_{-\mathtt{X}}$ formulas are a subset of (co-)safe LTL which, in turn, are a subset of full LTL. All the results we describe next for a given class of formulas apply to the contained classes as well.

3.3 LTL Specifications for MDPs

Given an MDP $\mathcal{M}$ and an LTL formula φ over the set of atomic propositions AP used to label the MDP, we write $Pr^{\pi}_{\mathcal{M},s}(\varphi)$ for the probability of a path satisfying φ from state s in MDP $\mathcal{M}$ under a policy π:

$$Pr^{\pi}_{\mathcal{M},s}(\varphi) = Pr^{\pi}_{\mathcal{M},s}(\{\sigma \in IPath_{\mathcal{M},s} \mid \sigma \models \varphi\}).$$

Furthermore, we write $Pr^{\max}_{\mathcal{M},s}(\varphi)$ to denote the maximum probability (over all policies) of satisfying φ from state s.

Another useful property is the expected amount of reward accumulated until a co-safe LTL formula φ is satisfied. For reward structure r, we define this using the function:

$$cumul^{\varphi}_{r}(s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \ldots) = \sum\nolimits_{i=0}^{n_\varphi - 1} r(s_i, a_i, s_{i+1}),$$

where n_φ is the first index for which $s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \ldots \xrightarrow{a_{n-1}} s_{n_\varphi}$ is a good prefix for φ. The expected amount of reward r accumulated before φ is satisfied under policy π on $\mathcal{M}$ is then defined as $E^{\pi}_{\mathcal{M},s}(cumul^{\varphi}_{r})$, and we write $E^{\max}_{\mathcal{M},s}(cumul^{\varphi}_{r})$ to denote the maximum expected value over all policies of $\mathcal{M}$.

For a co-safe LTL formula φ, we can compute both the maximum probability $Pr^{\max}_{\mathcal{M},s}(\varphi)$ and the maximum expected reward $E^{\max}_{\mathcal{M},s}(cumul^{\varphi}_{r})$, by building and solving a *product MDP*, which combines the MDP $\mathcal{M}$ with a DFA $\mathcal{A}_\varphi$ for φ.

Definition 7 (Product MDP)**.** If $\mathcal{M} = \langle S, \bar{s}, A, \delta, AP, L\rangle$ is an MDP and φ is a co-safe LTL formula over AP represented by DFA $\mathcal{A}_\varphi = \langle Q, \overline{q}, Q_F, 2^{AP}, \delta_\varphi\rangle$, the *product MDP* is the MDP $\mathcal{M} \otimes \mathcal{A}_\varphi = \langle S_\otimes, \bar{s}_\otimes, A, \delta_\otimes, AP, L_\otimes\rangle$ where:

- $S_\otimes = S \times Q$
- $\bar{s}_\otimes = (\bar{s}, \delta_\varphi(\overline{q}, L(\bar{s})))$
- $\delta_\otimes((s,q), a, (s', q')) = \begin{cases} \delta(s,a,s') & \text{if } q' {=} \delta_\varphi(q, L(s')) \\ 0 & \text{otherwise} \end{cases}$
- $L_\otimes(s,q) = \begin{cases} L(s) \cup \{acc_\varphi\} & \text{if } q \in Q_F \\ L(s) & \text{otherwise} \end{cases}$

The construction of the product MDP $\mathcal{M} \otimes \mathcal{A}_\varphi$ is well known; see, e.g., [6]. The product behaves like the original MDP $\mathcal{M}$ (it preserves the probabilities of paths from $\mathcal{M}$) but is augmented with information about the satisfaction of φ. Once a path of $\mathcal{M} \otimes \mathcal{A}_\varphi$ reaches an *accepting state* (i.e., a state of the form (s, q_F) for $q_F \in Q_F$), it is a good prefix for φ and we know that φ is satisfied. This reduces the problem of computing $Pr^{\max}_{\mathcal{M}}(\varphi)$ to finding the maximum probability of reaching an accepting state in the product. Since we label such states with a new atomic proposition acc_φ, we have:

$$Pr^{\max}_{\mathcal{M}}(\varphi) = Pr^{\max}_{\mathcal{M}\otimes\mathcal{A}_\varphi}(\mathsf{F}\ acc_\varphi).$$

Furthermore, a (memoryless) optimal policy for reaching acc_φ in $\mathcal{M}\otimes\mathcal{A}_\varphi$ can be converted to an optimal policy for φ in $\mathcal{M}$. The latter is a finite-memory policy whose modes are the DFA states Q. Optimal values and policies can be found using standard techniques over the product such as value iteration [49].

In a similar fashion, we can use the product MDP to compute the maximum expected cumulative reward until φ is satisfied, and a corresponding optimal policy:

$$E^{\max}_{\mathcal{M}}(cumul^{\varphi}_{r}) = E^{\max}_{\mathcal{M}\otimes\mathcal{A}_\varphi}(cumul^{acc_\varphi}_{r_\varphi}),$$

where r is a reward structure for $\mathcal{M}$ and r_φ is the corresponding reward structure for $\mathcal{M}\otimes\mathcal{A}_\varphi$, with reward values copied directly to their corresponding transitions in the product.

We will often use superscript notation when referring to product models, e.g., writing $\mathcal{M}^\varphi$ for $\mathcal{M}\otimes\mathcal{A}_\varphi$. For a list $\Phi = \langle\varphi_1,\dots,\varphi_m\rangle$ of co-safe LTL formulas, we can apply Definition 7 repeatedly to build the product $\mathcal{M}\otimes\mathcal{A}_{\varphi_1}\otimes\cdots\otimes\mathcal{A}_{\varphi_m}$ and we will sometimes use the shorthand $\mathcal{M}^\Phi$ for this.

4 PROBLEM FORMULATION

We now formalise the problem of multi-robot planning under uncertainty that we tackle in this paper. We explain the modelling of each individual robot, the joint multi-robot model, the mission specification for the robots, and then how these are combined to define a problem over an MDP.

4.1 Robot Models

Overall, we assume a set of n robots. The operation of each individual robot i as it attempts to perform tasks is modelled by an MDP $\mathcal{M}_i$. The state space of $\mathcal{M}_i$ comprises both the *local state* of the robot and also the *global state* common to all robots. In our work, the local state is typically the robot's location within a topological map used to model its environment. The topological map is a graph where nodes represent physical locations of interest and edges represent the robot's ability to move between any two nodes.

Example 1 (Topological Map)**.** Figure 1 shows the topological map for a toy example with 8 locations $v_0,\dots,v_7$. Green circles show locations that the robots must visit and the red circle is a location that all robots must avoid (see Section 4.3). Edges drawn as solid lines indicate that the robot can move freely between a pair of locations. A dashed edge indicates a constraint on the global state, in this case on the status of a door.

We assume that the global state comprises k separate state features. In the example above, there is a single global feature representing the state of the door. The MDP $\mathcal{M}_i$ for robot i has an action set A_i, which is partitioned into actions A^l_i that update the local state (e.g., representing navigation between locations) and action sets A^g_j that update global feature j (e.g., opening or closing a door). The MDP $\mathcal{M}_i$ for robot i is called a *local MDP* and is defined as follows.

Definition 8 (Local MDP)**.** A *local MDP* for robot i is an MDP $\mathcal{M}_i = \langle S_i, \bar{s}_i, A_i, \delta_i, AP, L_i\rangle$ where:

- the state space $S_i = S^l_i \times S^g_1 \times \dots \times S^g_k$ is the product of the robot's *local* state space S^l_i and the state space S^g_j for k *global* state features;
- actions $A_i = A^l_i \cup A^g_1 \cup \cdots \cup A^g_k$ are partitioned into action sets that update the local state and global state features.

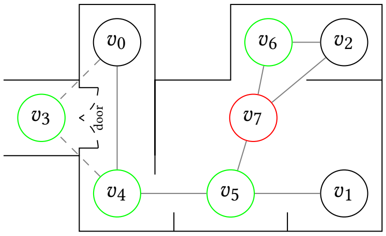


Fig. 1. Example topological map. Robots start in v_0, v_1, v_2; locations to visit in green and to avoid in red; mission specification $\Phi = \langle \mathsf{F}\,(v_5 \wedge \mathsf{F}\,v_4), \mathsf{F}\,v_3, \mathsf{F}\,v_6, \mathsf{G}\,\neg v_7 \rangle$.

Actions from each set update the corresponding part of the state, i.e., for states $s = (s^l, s^g_1, \ldots, s^g_k)$ and $t = (t^l, t^g_1, \ldots, t^g_k)$ in S_i, we require that, if $\delta_i(s, a, t) > 0$, then either:

- $a \in A^l_i$ and $s^g_j = t^g_j$ for all j; or
- $a \in A^g_j$, $s^l = t^l$ and $s^g_{j'} = t^g_{j'}$ for all $j' \neq j$.

For convenience, we assume there is a special action $\perp \in A^l_i$, which represents the robot staying idle. This is hence enabled in all states and corresponds to a self-loop transition. Probabilities in the MDP $\mathcal{M}_i$ represent uncertainty regarding either the environment or the behaviour of the robot. A typical example of the former would include when the robot attempts to navigate to a particular location, but an obstacle in the environment means that with some probability the robot moves to a different location or remains where it is. For the latter, in this work we are particularly interested in (permanent) robot *failure*, indicating that it is no longer able to perform any tasks. We assume a designated failure state $s^{fail}_i \in S^l_i$ for each robot i which, once entered, cannot be left. Probabilistic transitions to s^{fail}_i can occur from any state elsewhere in S^l_i. Whenever we say that "a robot fails", we model this through a transition to its failure state.

Example 2 (Local MDP)**.** Figure 2 shows a fragment of an example local MDP corresponding to the map in Example 1 (see Figure 1). There is one local state feature, the robot's location, with $S^l_i = \{v_0, \ldots, v_7\}$, and one global state feature, modelling the status of a door (open, closed, unknown) with $S^g_1 = \{o, c, ?\}$.

States are of the form $(v_j, door)$ and the initial state is $(v_0, ?)$, where the robot is in location v_0 and the door status is unknown. Actions updating the local state, in the set A^l_i, are of the form m_{jk} ("move from v_j to v_k"). Actions updating the global state, in the set A^g_1, are of the form cd_j ("check the door status when in v_j"). Some instances of both types of actions exhibit probabilistic behaviour. For example, when checking the door in v_0 (action cd_0), the door is open with probability 0.8; and moving from v_0 to v_3 (action m_{03}) succeeds with probability 0.9, or results in a transition to the robot's failure state with probability 0.1. We omit $\perp$ actions from Figure 2.

4.2 Multi-Robot Models

Given MDPs defining the local model for each of the n individual robots, we define the *joint MDP* as an MDP $\mathcal{M}_J$ formed as the product of these. This models the combined execution of the n robots in their environment.

The (joint) state space S_J of the global model includes the local state of each robot i and the state of each global feature j. We start by defining a function that projects joint states onto local robot states.

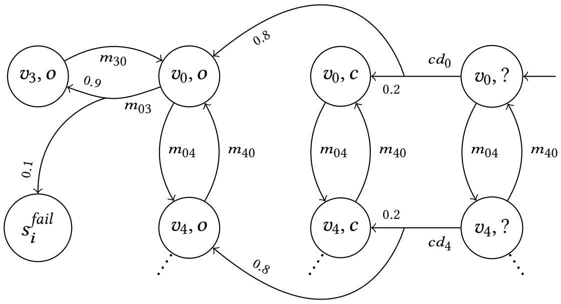


Fig. 2. Fragment of an example local MDP for robot i corresponding to the map in Figure 1 (see Example 2).

Definition 9. Let $S_J = S_1^l \times \cdots \times S_n^l \times S_1^g \times ... \times S_k^g$, and $i \in \{1, \ldots, n\}$. We write $[.]_i : S_J \to S_i$ for the function that projects states of $\mathcal{M}_J$ to the corresponding states of $\mathcal{M}_i$, defined as:

$$[s_1^l, \ldots, s_n^l, s_1^g, \ldots, s_k^g]_i = (s_i^l, s_1^g, \ldots, s_k^g).$$

In each transition of the global model, all robots make transitions simultaneously. We therefore require that, in any transition, at most one robot updates each global state feature. Since we partition actions according to the parts of the state space that they update, we can enforce this condition simply by constraining the set of joint actions A that are allowed in the global model MDP. Note that the partition of the action set is not necessary to enforce the condition, and one can do it when considering arbitrary action sets. However, to keep notation simple, we choose to introduce the model under the partitioned action set assumption.

Definition 10 (Joint MDP)**.** Let the model for each robot i be a local MDP $\mathcal{M}_i = \langle S_i, \bar{s}_i, A_i, \delta_i, AP, L_i \rangle$, where:

- $S_i = S_i^l \times S_1^g \times \cdots \times S_k^g$;
- $\bar{s}_i = (\bar{s}_i^l, \bar{s}_1^g, \ldots, \bar{s}_k^g)$;
- $A_i = A_i^l \cup A_1^g \cup \cdots \cup A_k^g$;

We assume that the k global state features are the same for all robots (i.e., their state spaces S_j^g, initial values $\bar{s}_j^g$ and action sets A_j^g are the same in each $\mathcal{M}_i$), as is the set AP. The *joint MDP* is an MDP $\mathcal{M}_J = \langle S_J, \bar{s}_J, A_J, \delta_J, AP, L_J \rangle$ where:

- $S_J = S_1^l \times \cdots \times S_n^l \times S_1^g \times ... \times S_k^g$;
- $\bar{s}_J = (\bar{s}_1^l, \ldots, \bar{s}_n^l, \bar{s}_1^g, \ldots, \bar{s}_k^g)$;
- $A_J = \{(a_1, \ldots, a_n) \mid a_i \in A_i$ and, for each $1 \leq j \leq k$, we have $a_i \in A_j^g$ for at most one $i\}$.

Then, for states $s, t \in S_J$ and action $a = (a_1, \ldots, a_n) \in A_J$, we define δ_J and L_J as follows:

- $\delta_J(s, a, t) = \prod_{i=1}^n \delta_i([s]_i, a_i, [t]_i)$;
- $L_J(s) = \bigcup_{i=1}^n L_i([s]_i)$.

Our proposed joint MDP model is an instance of a multi-agent MDP [10] (MMDP), where we impose extra structure on local and global state features. Thus, our joint model has similar assumptions as MMDPs, namely that robots have access to the full state space of the team, and that actions are executed in a synchronised fashion, using a common timestep across the robots. We also assume robots can coordinate locally and navigate around each other efficiently, and do not consider issues related to collisions and obstacle avoidance in the model. One can achieve efficient local coordination by using specific motion planning approaches such as [7, 41]. Our focus in this paper is instead on planning at a higher level of abstraction: we aim to provide effective and adaptive task allocation in the presence of individual (and permanent) robot failures.

4.3 Mission Specification

We use a mix of co-safe and safe LTL$_{-\mathsf{X}}$ formulas to specify the robots' mission. A *mission specification* $\Phi = \langle \varphi_1, \ldots, \varphi_m, \varphi_{safe} \rangle$ consists of a list of co-safe LTL$_{-\mathsf{X}}$ *task specifications* $\varphi_1, ..., \varphi_m$ and a *safety specification* φ_{safe} in safeLTL$_{-\mathsf{X}}$, all written over the set *AP*. We assume the team's mission is composed of formulas which are (i) non-conflicting, i.e., (non-)satisfaction of one formula must not violate any other formula in the mission, and (ii) achievable by a single robot. These assumptions have been made in other work on this topic [18, 54].

Example 3 (Mission Specification)**.** We return to the running example. Let v_j be an atomic proposition indicating that a robot is in location v_j. When used to label a state of local MDP $\mathcal{M}_i$, this means that robot i is in v_j. Since the state labelling in the joint MDP takes the union of individual labellings for local MDPs, when v_j labels a state in $\mathcal{M}_J$, it indicates that *some* robot is in v_j.

An example mission specification for this model is $\Phi = \langle \mathsf{F}(v_5 \wedge \mathsf{F}\, v_4), \mathsf{F}\, v_3, \mathsf{F}\, v_6, \mathsf{G}\, \neg v_7 \rangle$. Task specification $\varphi_1 = \mathsf{F}(v_5 \wedge \mathsf{F}\, v_4)$ requires first v_5 then v_4 to be visited; the other two tasks are to visit v_3 and v_6, respectively. The safety specification $\varphi_{safe} = \mathsf{G}\, \neg v_7$ states that location v_7 must be avoided.

4.4 Problem Statement

Given a set of n robots and a mission specification $\Phi = \langle \varphi_1, \ldots, \varphi_m, \varphi_{safe} \rangle$, our aim is to derive a joint policy for the robots which allows them to collectively achieve the tasks $\varphi_1, ..., \varphi_m$ without violating the safety constraint φ_{safe}. This incorporates both the allocation of tasks to robots, and the planning for each robot to achieve its tasks. Since our model explicitly includes the possibility of individual robot failures (after which they are unable to perform any tasks), policies can also dynamically reallocate tasks to robots as needed. In addition, we aim to produce *probabilistic guarantees* for these policies which quantify their effectiveness or reliability.

Formally, if the behaviour of each robot i is defined by local MDP $\mathcal{M}_i$, then our goal becomes to synthesise an appropriate *joint policy*, i.e., a policy π_J for the joint MDP $\mathcal{M}_J$. One possibility would be to find a policy that maximises the probability $Pr^{\pi_J}_{\mathcal{M}_J}(\varphi_1 \wedge \cdots \wedge \varphi_m \wedge \varphi_{safe})$ of satisfying *all* formulas in Φ, as in [19]. However, in some scenarios, one or more tasks may become unachievable (for example, if a door is closed, then some locations may become inaccessible for all robots) reducing this probability to zero.

So, instead, we target policies that *maximise the expected number of tasks φ_i completed without violating the safety constraint φ_{safe}*. To formalise this, we use the product construction described in Section 3.3. The co-safeLTL$_{-\mathsf{X}}$ formulas φ_i are used to construct a reward structure over the product that counts the number of tasks that are completed; and we consider the expected amount of this reward accumulated until the negation of the safe LTL$_{-\mathsf{X}}$ formula φ_{safe} is satisfied. Formally, let $\mathcal{M}'_J = \mathcal{M}_J \otimes \mathcal{A}_{\varphi_1} \otimes \cdots \otimes \mathcal{A}_{\varphi_m}$ be the product of the joint MDP $\mathcal{M}_J$ with DFAs for the m formulas φ_i and $\mathcal{M}^{\Phi}_J = \mathcal{M}'_J \otimes \mathcal{A}_{\neg\varphi_{safe}}$ be the product of $\mathcal{M}_J$ with DFAs for all $m + 1$ formulas in the mission specification Φ. We fix a reward structure *tasks* for $\mathcal{M}'_J$ that counts the number of tasks that are completed, i.e., which assigns to each transition $s \xrightarrow{a} s'$ the number of tasks φ_i for which it is a transition into an accepting state for $\mathcal{A}_{\varphi_i}$. Then, our goal is to compute the expected cumulative value of *tasks* up until a point where $\neg\varphi_{safe}$ becomes true, which reduces to computing the expected cumulative reward until reaching accepting states for $\mathcal{A}_{\neg\varphi_{safe}}$ in the product model $\mathcal{M}_J^{\Phi}$:

$$E^{\max}_{\mathcal{M}'_J}(cumul^{\neg\varphi_{safe}}_{tasks}) \;=\; E^{\max}_{\mathcal{M}^{\Phi}_J}(cumul^{acc_{\neg safe}}_{tasks}),$$

where, by slight abuse of notation, we use *tasks* to refer to the reward structures for both models.

Note that the cumulative value of *tasks* is always finite, because we consider a finite number of co-safe tasks. Thus, in order to compute $E^{\max}_{\mathcal{M}^{\Phi}_J}(cumul^{acc_{\neg safe}}_{tasks})$, we can make states in $acc_{\neg safe}$ absorbing and compute the total expected reward in the resulting model. In practice, this means that policies will be defined until we reach a state where $\neg\varphi_{safe}$ becomes true (i.e., a state where the safety specification has been broken); or the expected cumulative value of the *tasks* reward structure becomes zero (i.e., a state from which the team cannot achieve more tasks). In the latter case, policies will finish without breaking the safety specification. This means the policies will try to avoid states where $\neg\varphi_{safe}$ is satisfied, as they cannot accumulate more reward after such states are reached and the goal is to maximise reward. Finally, we note that the *tasks* reward can be adapted to assign different values to each task, thus allowing for prioritising the satisfaction of some co-safe $\text{LTL}_{-\chi}$ tasks over others.

It may not be feasible to find the *optimal* policy for the objective above, but for any policy that we do synthesise, we will also compute the actual expected number of tasks completed without violating the safety constraint, which represents a *probabilistic guarantee* on its performance. We can also compute separate guarantees, for example, the probability with which a particular task φ_i is completed or with which the safety specification φ_{safe} holds.

5 SIMULTANEOUS TASK ALLOCATION AND PLANNING UNDER UNCERTAINTY (STAPU)

The problem of determining a joint robot policy, as formulated in the previous section, can be undertaken in a number of ways. In this paper, we propose an approach called *simultaneous task allocation and planning under uncertainty (STAPU)*, which combines the processes of task allocation and task planning in order to exploit information about individual robot plans into the allocation process.

In principle, we could achieve this by constructing and solving the global MDP product $\mathcal{M}_J^{\Phi}$, following the problem formulation given in Section 4.2, which would yield an optimal joint policy. However, this approach is typically infeasible since the MDP grows exponentially with the number of robots.

Instead, inspired by the (non-probabilistic) approach of [52], we propose a method that initially plans on a *sequential* model of the robots, avoiding the construction of the fully synchronised joint model. This exploits the assumption that tasks have no interdependencies and can each be completed by a single robot. We consider each robot independently in turn, allowing it to (simultaneously) choose tasks to undertake and decide how best to complete them.

To achieve this, we build and solve a *team MDP*, which can be viewed as a sequence of models for each individual robot. More precisely, each individual model for robot i is a *local product MDP*, encoding: the dynamics of the robot (from local MDP $\mathcal{M}_i$); the definitions of the tasks, along with the extent to which they have been completed so far (using DFAs $\mathcal{A}_\varphi$ for the co-safe formulas φ in the mission specification Φ); and the state of the safety constraint (using the state of $\mathcal{A}_{\neg\varphi_{safe}}$) .

The local product MDPs are joined using *switch transitions*, which represent changes in the allocation of a task from robot i to $i+1$ (the next robot in the sequential model). These transitions are added in every state of robot i's model where no task has yet been started, or one has just been completed, as determined by whether the DFAs for each task are in their initial or accepting states. When a switch transition occurs, robot $i+1$ begins in its initial state. The state of the DFAs remains the same, so that information about task completion is preserved. Considered sequentially, this

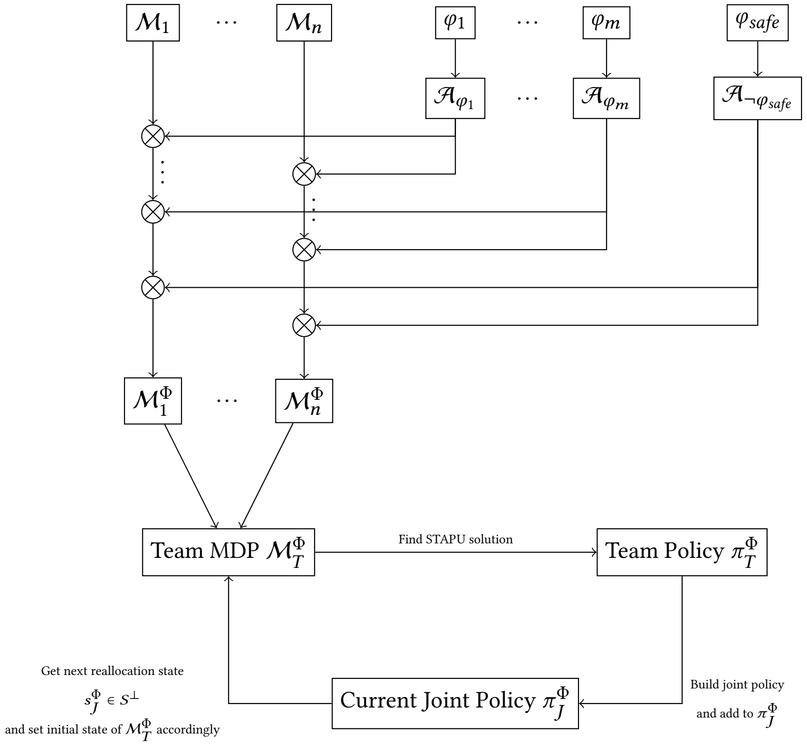

$\mathcal{M}_1$
$\cdots$
$\mathcal{M}_n$
φ_1
$\cdots$
φ_m
φ_{safe}
$\mathcal{A}_{\varphi_1}$
$\cdots$
$\mathcal{A}_{\varphi_m}$
$\mathcal{A}_{\neg\varphi_{safe}}$
$\mathcal{M}_1^\Phi$
$\cdots$
$\mathcal{M}_n^\Phi$
Team MDP $\mathcal{M}_T^\Phi$
Find STAPU solution
Team Policy π_T^Φ
Build joint policy and add to π_J^Φ
Current Joint Policy π_J^Φ
Get next reallocation state $s_J^\Phi \in S^\perp$ and set initial state of $\mathcal{M}_T^\Phi$ accordingly

Fig. 3. Overview of STAPU with reallocation. Each robot MDP, $\mathcal{M}_i$ is combined with DFA, $\mathcal{A}_{\varphi_1}, \ldots, \mathcal{A}_{\varphi_m}, \mathcal{A}_{\neg\varphi_{safe}}$ to generate a local product MDP $\mathcal{M}_i^\Phi$. The local product MDPs for all n robots are used to create a team MDP, $\mathcal{M}_T^\Phi$. A sequential team policy, π_T^Φ is generated by maximising the expected cumulative task reward over $\mathcal{M}_T^\Phi$. The sequential policy, π_T^Φ is used to construct a a joint policy, π_J^Φ. While constructing π_J^Φ, reallocation states $s_J^\Phi \in S^\perp$ are identified. Replanning takes place for each of these states, ordered by decreasing reachability probability.

model allows each robot a choice, before or after executing any task, as to whether it or the subsequent robots should tackle the unallocated tasks.

We find a sequential policy, by computing an optimal policy for the team MDP that maximises the expected number of tasks completed without violating the safety constraint. We then convert this sequential policy into a joint policy, where the robots execute concurrently. In situations where an action for this policy cannot be generated (e.g., because a robot has failed), we perform a *replanning* process from that joint state. Figure 3 provides an overview of these steps which are described in more detail in the following sections.

5.1 Team MDP and Sequential Policy

The *team MDP* $\mathcal{M}_T^\Phi$ is built as the *union* of MDPs for the n robots, and models their joint behaviour in a *sequential* fashion. The MDP for each individual robot i is the local product MDP $\mathcal{M}_i^\Phi = \mathcal{M}_i \otimes \mathcal{A}_{\varphi_1} \otimes \cdots \otimes \mathcal{A}_{\varphi_m} \otimes \mathcal{A}_{\neg\varphi_{safe}}$, combining the local model $\mathcal{M}_i$ of robot i's behaviour with DFAs representing the satisfaction of the formulas in the mission $\Phi = \langle \varphi_1, \ldots, \varphi_m, \varphi_{safe} \rangle$.

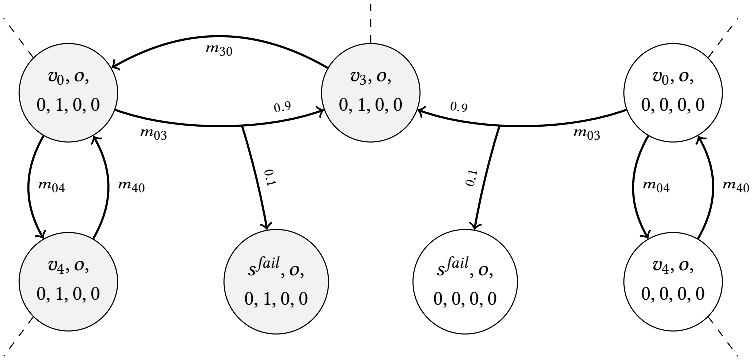


Fig. 4. Fragment of the local product MDP for robot i. States are shaded if task 2 (reaching location v_3) has been completed.

Example 4 (Local Product MDP)**.** Consider the local MDP depicted in Figure 2. The mission specification is as described in Example 3, i.e., $\Phi = \langle \mathsf{F}(v_5 \wedge \mathsf{F}\, v_4), \mathsf{F}\, v_3, \mathsf{F}\, v_6, \mathsf{G}\, \neg v_7 \rangle$. The local product MDP for each robot contains the states of the robot, the state of the door and the states for the mission. A fragment of the local product MDP is shown in Figure 4. For simplicity, we depict a small part which includes four states from the local MDP in Figure 2. Each state is of the form $(v_i, door, q_1, q_2, q_3, q_{safe})$ where v_i denotes the robot's position on the topological map, *door* denotes the state variable of the door, and q_i, q_{safe} denote the states of the DFAs for task φ_i and safety specification φ_{safe}, respectively. If the robot starts in state $(v_0, o, 0, 0, 0, 0)$ and chooses action m_{03}, it moves to state $(v_3, o, 0, 1, 0, 0)$ with probability 0.9, completing task 2 ($\mathsf{F}\, v_3$). The DFA for this task has two states: the initial state 0 and accepting state 1. The robot cannot subsequently go back to a state where the task is not complete.

States of the team MDP $\mathcal{M}_T^{\Phi}$ take the form (i, s_i^{Φ}) where i denotes the robot currently executing tasks and s_i^{Φ} is a state of the local product MDP $\mathcal{M}_i^{\Phi}$. So, $s_i^{\Phi} = (s_i, q_1, \dots, q_m, q_{safe})$ comprises a state $s_i \in S_i$ from the local MDP $\mathcal{M}_i$ for robot i and one for each of the DFAs $\mathcal{A}_{\varphi_1}, \dots, \mathcal{A}_{\varphi_m}, \mathcal{A}_{\neg\varphi_{safe}}$.

Definition 11 (Team MDP)**.** Given a mission Φ and the local product MDPs $\mathcal{M}_i^{\Phi} = \langle S_i^{\Phi}, \bar{s}_i^{\Phi}, A_i, \delta_i^{\Phi}, AP^{\Phi}, Lab_i^{\Phi} \rangle$ for each robot i, the *team MDP* is defined as the MDP $\mathcal{M}_T^{\Phi} = \langle S_T^{\Phi}, \bar{s}_T^{\Phi}, A_T, \delta_T^{\Phi}, AP^{\Phi}, Lab_T^{\Phi} \rangle$ where:

- $S_T^{\Phi} = \bigcup_{i=1}^{n} (\{i\} \times S_i^{\Phi})$
- $\bar{s}_T^{\Phi} = (1, \bar{s}_1^{\Phi})$
- $Lab_T^{\Phi}(i, s_i^{\Phi}) = Lab_i^{\Phi}(s_i^{\Phi})$
- $A_T = \{\zeta\} \cup \bigcup_{i=1}^{n} A_i$, where ζ labels a *switch transition*
- δ_T^{Φ} is defined as follows. For action $a_i \in A_i$ of robot i, the team MDP mirrors the local product MDP:

$$\delta_T^{\Phi}((i, s_i^{\Phi}), a_i, (i, t_i^{\Phi})) = \delta_i^{\Phi}(s_i^{\Phi}, a_i, t_i^{\Phi}),$$

where $s_i^{\Phi}, t_i^{\Phi} \in S_i^{\Phi}$ are states of the local product MDP for robot i. For switch transitions:

$$\delta_T^{\Phi}((i, s_i^{\Phi}), \zeta, (j, t_j^{\Phi})) = 1,$$

if all of the following conditions hold:

(i) $i < n$ and $j = i + 1$;

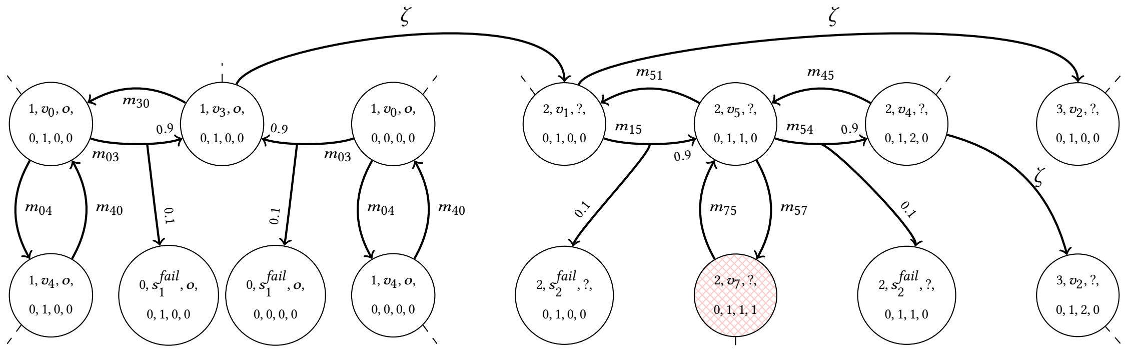


Fig. 5. A fragment of the team MDP. The first variable in each state refers to the robot that is being represented. The red state must be avoided as per the safety specification.

(ii) $i = 1$ and s_1^Φ is the initial local product state for robot 1, or (i, s_i^Φ) has an incoming switch transition, or s_i^Φ includes a "newly reached" accepting state (i.e., for some task φ_j, q_j is an accepting state and has an incoming transition in $\mathcal{M}_i^\Phi$ from a state that is not accepting, indicating the task just been achieved);

(iii) local product MDP states $s_i^\Phi{=}(s_i, q_1, \ldots, q_m, q_{safe})$ and $t_j^\Phi{=}(t_j, q_1, \ldots, q_m, q_{safe})$ have the same DFA state components $q_1, \ldots, q_m, q_{safe}$;

(iv) t_j is the initial state in robot j's local MDP $\mathcal{M}_j$.

For all other pair of states, $\delta_T^\Phi((i, s_i^\Phi), \zeta, (j, t_j^\Phi)) = 0$.

The team MDP considers the local product models sequentially. Special *switch transitions* are added to connect the local product model of each robot with the local product model of the subsequent robot. These represent moving the task allocation and planning process to the next robot. For robot i, switch transitions can occur when robot i has not acted (modelling the case where we do not allocate any task to i), or when robot i has just completed a task. The team MDP encodes the execution of the tasks by the team, whilst avoiding the construction of the joint MDP. Note that, while the number of states in a joint MDP is exponential in the number of robots, for the team MDP it is linear. This will allow us to scale to much larger models, as will be highlighted in Section 6.

Example 5 (Team MDP)**.** Figure 5 shows a fragment of the team MDP for the problem in Example 1. There are three copies of the local product MDPs, one per robot. Each copy is only linked to the next copy through switch transitions ζ that originate from states where tasks have just been completed, or from states that have incoming switch transitions. The switch transitions maintain the values of the automata states and jump to the initial location of the next robot.

Remark 1 (Switch Transitions and Failure States). In our definition of team MDP, there are no switch transitions from designated failure states or states where the safety specification is violated. This is because one way to maximise the expected task reward is to have each robot attempt to satisfy all the tasks in the mission specification in turn: if the first robot fails, the second robot continues from then on; if the second robot fails, the next robot continues; and so on. If the first robot does not fail, it completes the entire mission while the other robots in the team do nothing. This behaviour does not effectively allocate tasks across the team, and does not consider the total number of steps in the plan. Thus, we do not add switch transitions to designated failure states to prevent the policies from behaving in the way described above. We will deal with failures by replanning when the policy is undefined, as will be explained in Subsection 5.3.

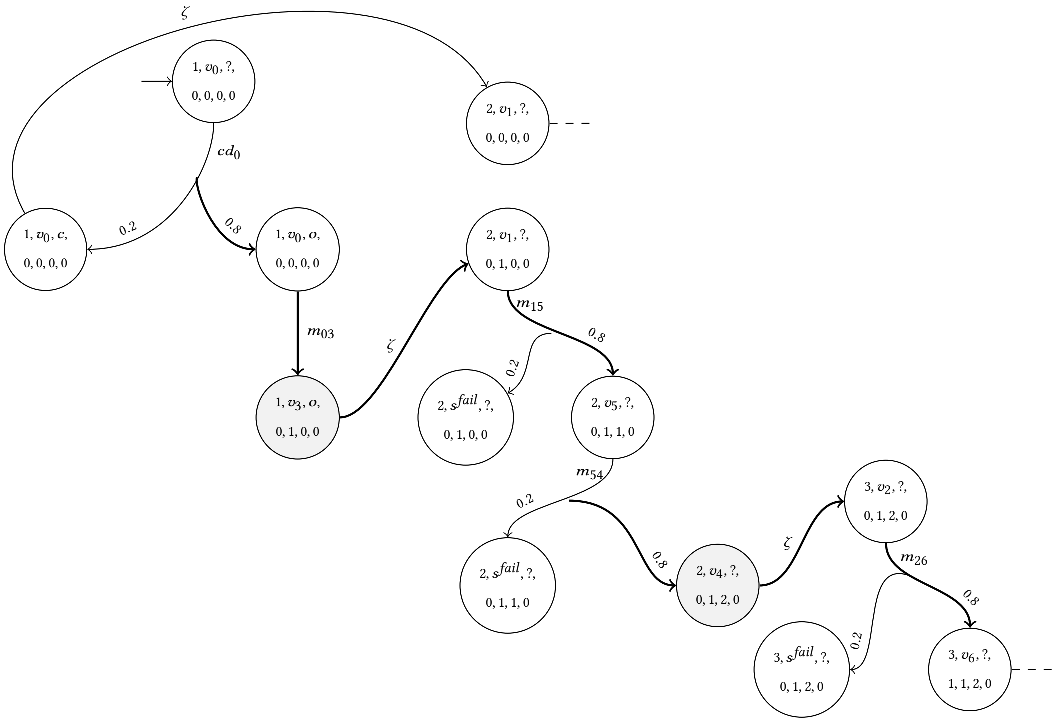


Fig. 6. A fragment of the sequential policy extracted from the team MDP. The thick lines show the most likely path in this fragment. Shaded states represent the most probable terminal state.

Our first step towards creating a joint policy is to construct a *sequential policy* π_T^Φ by solving the team MDP $\mathcal{M}_T^\Phi$. Since $\mathcal{M}_T^\Phi$ already includes the DFAs for the LTL$_{-\mathsf{X}}$ formulas in the mission specification Φ, we generate the policy π_T^Φ using the same approach formalised in Section 4.4, i.e., finding an optimal policy that maximises the expected number of tasks completed without violating the safety condition:

$$E^{\max}_{\mathcal{M}_T^\Phi}(cumul^{acc_{\neg safe}}_{tasks}).$$

The atomic proposition $acc_{\neg safe}$ labels states where the DFA for φ_{safe} is in an accepting states; these are transferred to $\mathcal{M}_T^\Phi$ when copying the labelling from the local product MDPs $\mathcal{M}_i^\Phi$ (see Definition 11). The reward structure *tasks* is defined exactly as in Section 4.4, by counting the number of tasks that are completed when executing a given transition.

Example 6 (Sequential Policy)**.** In Figure 6, we depict a fragment of the sequential policy obtained from the team MDP in Figure 5. Note the sequential nature of this policy: it starts (in state $(1, v_0, ?, 0, 0, 0, 0)$) with actions for robot 1 (cd_0 and m_{03}), then moves to robot 2, and finally to robot 3. Given that these policies will be executed in parallel, we reset the value of global state features (the door state, here) when applying switch transitions, to ensure that robots do not follow policies under wrong assumptions about the state of the system.

Note that the quality of the sequential policy π_T^Φ is dependent on the ordering over robots used to build the team MDP $\mathcal{M}_T^\Phi$. Finding good orderings is outside the scope of this paper. This problem has, however, been addressed in the context of MAPF, where it has been shown that finding an optimal ordering is NP-hard [8] and for which several heuristic and optimisation approaches have been proposed [9, 62, 65].

5.2 Building a Joint Policy

We start with a (memoryless) optimal policy $\pi_T^\Phi : S_T^\Phi \to A_T$ for the team MDP $\mathcal{M}_T^\Phi$. This policy is *sequential* in nature, i.e., it starts by prescribing actions for robot 1, then a switch transition occurs and actions are prescribed for robot 2, and so on. However, our goal is for these policies to be executed *concurrently.* In this section, we describe how we use this sequential policy to construct a joint policy π_J. This joint policy can then be used for execution, and also for providing more accurate performance guarantees over the joint model. Note that, whilst enumerating all states of the joint model is typically infeasible, the states visited *under a policy* are usually just a small fraction of the full joint model state space. Thus providing guarantees of joint execution is feasible.

Since we are working with (co-)safe LTL$_{-\mathsf{X}}$ specifications, we construct a memoryless joint policy π_J^Φ which applies to the product of the joint model $\mathcal{M}_J$ and the DFAs for the formulas in the mission specification Φ. This can easily be converted into a finite-memory policy π_J for $\mathcal{M}_J$. In constructing the joint policy π_J^Φ, we will assume that robots communicate with each other after executing each action.

We start by defining the *projection* of a team MDP policy π_T^Φ onto a policy over each local product MDP $\mathcal{M}_i^\Phi$.

Definition 12 (Policy projection)**.** Let $\pi_T^\Phi : S_T^\Phi \to A_T$ be a policy for team MDP $\mathcal{M}_T^\Phi$. Its projection onto $\mathcal{M}_i^\Phi$ is the policy $[\pi_T^\Phi]_i : S_i^\Phi \to A_i$, defined by:

$$[\pi_T^\Phi]_i(s) = \begin{cases} \pi_T^\Phi(i,s) & \text{if } \pi_T^\Phi(i,s) \text{ is defined and } \pi_T^\Phi(i,s) \neq \zeta \\ \bot & \text{otherwise.} \end{cases}$$

Example 7 (Policy Projection)**.** Examples of the actions taken in states by the policy projection of the sequential policy depicted in Figure 6 are:

$$\begin{aligned} &[\pi_T^\Phi]_1(v_0, ?, 0, 0, 0, 0) = \pi_T^\Phi(1, v_0, ?, 0, 0, 0, 0) = cd_0, \\ &[\pi_T^\Phi]_1(v_3, o, 0, 1, 0, 0) = \bot, \\ &[\pi_T^\Phi]_2(v_1, ?, 0, 1, 0, 0) = \pi_T^\Phi(2, v_1, ?, 0, 1, 0, 0) = m_{15}. \end{aligned}$$

We want to use π_T^Φ to select an action a to take in joint product state $s_J^\Phi = (s_J, q^\Phi)$ where $s_J = (s_1^l, \ldots, s_n^l, s_1^g, \ldots, s_k^g)$ and $q^\Phi = (q_1, \ldots, q_m, q_{safe})$. A first approach one might think of to do so is to choose the joint action:

$$a = ([\pi_T^\Phi]_1([s_J]_1, q^\Phi), \ldots, [\pi_T^\Phi]_n([s_J]_n, q^\Phi)).$$

However, doing so ignores the sequential nature of π_T^Φ, which is synthesised over a model that assumes robots act in order. This means that, in action a above, robot i does not consider the tasks that previous robots intend to complete when choosing its action. This can lead to lack of coordination, with multiple robots attempting to complete the same task. To avoid this issue, the policy action for robot i is chosen using an updated version of q^Φ that assumes the previous robots' execution will be according to the path with highest probability. To define the update to q^Φ, we start by considering the notion of *most probable terminal state* reached in $\mathcal{M}_i^\Phi$ under $[\pi_T^\Phi]_i$. Terminal states are those where no policy action is available, either because a switch transition occurred in the team MDP or because the robot failed.

Definition 13 (Most probable terminal state)**.** Let $term([\pi_T^\Phi]_i) = \{s \in S_i^\Phi \mid [\pi_T^\Phi]_i(s) = \bot\}$. The most probable terminal state for robot i, starting in $s \in S_i^\Phi$ and under policy $[\pi_T^\Phi]_i$ is defined as:

$$s_i^* = \operatorname*{arg\,max}_{s' \in term([\pi_T^\Phi]_i)} Pr^{[\pi_T^\Phi]_i}_{\mathcal{M}_i^\Phi, s}(\mathsf{F}\, s'),$$

where we slightly abuse LTL notation and write $\mathsf{F}\, s'$ to denote "eventually reach state s'".

Example 8 (Most probable terminal state)**.** Consider the policy projections for the sequential policy depicted in Figure 6. The most probable terminal state for each policy is the terminal state that is reached by following the most likely path. Thus, as depicted in the figure, $s_1^* = (v_3, o, 0, 1, 0, 0)$ and $s_2^* = (v_4, ?, 0, 1, 2, 0)$.

Using the notion of most probable terminal state, we can define the notion of a *concurrency-aware projection* of a joint product MDP state to a local product state for i.

Definition 14 (Concurrency-aware projection)**.** Let $s_J^\Phi = (s_J, q^\Phi) \in S_J^\Phi$ with $s_J = (s_1^l, \ldots, s_n^l, s_1^g, \ldots, s_k^g)$ and $q^\Phi = (q_1, \ldots, q_m, q_{safe})$ The *concurrency-aware projection* of s_J^Φ onto the local product state space S_i^Φ of robot i is defined recursively as follows:

$$[s_J^\Phi]^{\|}_1 = (s_1^l, s_1^g, \ldots, s_k^g, q_1, \ldots, q_m, q_{safe}),$$
$$[s_J^\Phi]^{\|}_{i+1} = (s_{i+1}^l, s_1^g, \ldots, s_k^g, q_1^{i*}, \ldots, q_m^{i*}, q_{safe}^{i*}),$$

where $(q_1^{i*}, \ldots, q_m^{i*}, q_{safe}^{i*})$ are the DFA components of the most probable terminal state for robot i, starting in $[s_J^\Phi]^{\|}_i$ and under $[\pi_T^\Phi]_i$.

Broadly speaking, we project the joint MDP states to the corresponding local state, as in Definition 9, but also update the DFA states taking into account the most probable task execution of previous robots.

Example 9 (Concurrency-aware projection)**.** Consider the policies represented in Figure 6 and the joint state $s_J^\Phi = (v_0, v_1, v_2, ?, 0, 0, 0, 0)$ The concurrency-aware projections for each robot are defined as:

$$[s_J^\Phi]^{\|}_1(s_J^\Phi) = (v_0, ?, 0, 0, 0, 0);$$
$$[s_J^\Phi]^{\|}_2(s_J^\Phi) = (v_1, ?, 0, 1, 0, 0);$$
$$[s_J^\Phi]^{\|}_3(s_J^\Phi) = (v_2, ?, 0, 1, 2, 0).$$

We can now define the joint policy obtained from π_T^Φ.

Definition 15 (Joint policy)**.** The *joint policy* $\pi_J^\Phi : S_J^\Phi \to A_J$ is such that:

$$\pi_J^\Phi(s_J^\Phi) = \left([\pi_T^\Phi]_1\left([s_J^\Phi]^{\|}_1\right), \ldots, [\pi_T^\Phi]_n\left([s_J^\Phi]^{\|}_n\right)\right).$$

To avoid introducing extra notation, the definition above does not consider conflicting actions, i.e., joint actions where more than one robot changes the value of the same global state feature. According to Definition 10, these are not in A_J. We address this in practice by allowing the robot with lowest index to execute its policy action, and making the subsequent conflicting robots idle, i.e., making the policy action for the subsequent conflicting robots equal to $\bot$. Note that, because we disallow the X operator, this does not influence the satisfaction of the mission specifications.

The joint policy can be executed by the team of robots in a synchronised fashion. To compute π_J^Φ at each joint product state, we need to compute the concurrency-aware projection of that state to each local product. This requires us to compute the most probable terminal state for the corresponding state in each local product MDP, which can be computationally expensive, as it requires the computation of reachability probabilities. However, by maintaining the results of the reachability probability computations for all states reachable under $[\pi_T^\Phi]_i$, for each robot i, one only needs to compute the reachability probabilities once per local product MDP.

Example 10 (Joint Policy)**.** Considering again the policies represented in Figure 6, we can define the joint policy for joint state $s_J^\Phi = (v_0, v_1, v_2, ?, 0, 0, 0, 0)$ as:

$$\begin{aligned}\pi_J^\Phi(s_J^\Phi) &= \left([\pi_T^\Phi]_1\left([s_J^\Phi]_1^{||}\right), [\pi_T^\Phi]_2\left([s_J^\Phi]_2^{||}\right), [\pi_T^\Phi]_3\left([s_J^\Phi]_3^{||}\right)\right)\\ &= \left([\pi_T^\Phi]_1\left((v_0, ?, 0, 0, 0, 0)\right), [\pi_T^\Phi]_2\left((v_1, ?, 0, 1, 0, 0)\right), [\pi_T^\Phi]_3\left((v_2, ?, 0, 1, 2, 0)\right)\right).\end{aligned}$$

The joint policy is defined for all joint states in a similar manner.

5.3 Reallocation through Replanning

As described in Definition 11, switch actions are only added to states where a task has been completed. This allows the team model to consider robots in a sequential fashion, but also introduces *reallocation states* where the joint policy is not defined. In those cases, our approach is to build another instance of the team MDP, where the initial state is set to the reallocation state. In this subsection, we formally define the reallocation states, and propose an approach for replanning from those states that takes into account how likely they are to occur. By first considering the most probable reallocation states, our algorithm can be viewed as *anytime*, as it can incrementally build a more complete solution, with more accurate performance guarantees, but if interrupted early, it can still provide a partial solution and an under-estimate of the guarantee.

Definition 16 (Reallocation states)**.** We define the set of reallocation states as:

$$S^\perp = \{s_J^\Phi \in S_J^\Phi \mid [\pi_T^\Phi]_i\left([s_J^\Phi]_i^{||}\right) = \perp \text{ for all } 1 \leq i \leq n\}.$$

In words, reallocation states are states for which the joint policy (Definition 15) is undefined for all robots, i.e., there are no more actions for the team to execute. In order to replan for a reallocation state $s_J^\Phi \in S^\perp$, we simply set the initial states of each local MDP $\mathcal{M}_i$ to the corresponding projection $[s_J^\Phi]_i$, and reconstruct a team MDP according to Definition 11. We consider more likely reallocation states first, by ordering the set $S^\perp$ in decreasing order of reachability probability under π_J^Φ, i.e. we order states $s_J^\Phi \in S^\perp$ in decreasing order of $Pr^{\pi_J^\Phi}_{\mathcal{M}_i^\Phi, \bar{s}_J^\Phi}(\mathsf{F}\, s_J^\Phi)$.

6 EVALUATION

We have developed an implementation of our STAPU approach, built on top of the probabilistic model checker PRISM [33]. This tool provides construction of MDPs, from a high-level modelling language, and verification of the MDPs against specifications in LTL, which includes functionality required to implement STAPU such as generating DFAs, building MDP-DFA products, and solving MDPs using a variety of techniques. The implementation builds on PRISM's "explicit" model checking engine, which is written in Java.

6.1 Baseline Comparison: Auctioning

The goal of the STAPU approach is to provide an efficient and scalable approach to task allocation and planning, whilst also generating probabilistic guarantees in the form of expected measures of performance for policies. A comparison of STAPU against the naive approach of building and solving the full joint MDP is impractical since this has very limited scalability: for a representative topological map with 100 locations and one basic reachability task per robot, solving the MDP for 3 robots is feasible (needing approximately 8 million states), but quickly becomes infeasible since the joint MDP is exponential in the number of both robots and tasks. Instead, for a more competitive baseline comparison, we implement an alternative approach where task allocation and task planning are *decoupled*, rather than performed simultaneously as in STAPU. To do so, we use an implementation that combines *sequential single-item auctioning* [30] with MDP-based planning.

Sequential single-item auctioning [30] is an approach to task allocation which proceeds in a series of rounds. In every round, each robot makes a *bid* for a task that is not yet unallocated, and provides an estimate of the contribution that this would make to the overall team objective if it carried out this task in addition to the ones that it has already been allocated. The robot with the best bid is allocated the corresponding task and the round concludes. Subsequent rounds run until all tasks are allocated. In our setting, the auction has m rounds, since this is the number of tasks, and robots' bids are based on the highest increase that can be made to the expected number of tasks completed.

In order to ensure that STAPU and the baseline method optimise for similar objectives, our sequential single-item auction also uses the *tasks* reward structure from Section 4.4. For a fair comparison, and to ensure that the auction-based approach provides comparable probabilistic guarantees to STAPU, we adopt similar techniques for the baseline implementation using MDPs and LTL task specifications. Specifically, to calculate robot i's bid, we first solve local product MDPs using the set of tasks already assigned to i and each of the remaining unassigned tasks. From these, we choose the unassigned task that offers the highest increase to the team.

Once all tasks are assigned to robots, the auctioning process stops. We then use this task allocation to generate policies for each robot. To save computation, we re-use the policies generated during the auctioning process. The individual robot policies are used to construct a joint policy in a manner similar to the one described for STAPU. This allows us to restrict actions that modify global states and to identify states where robots need to replan. Note that the local product MDP for each robot includes a subset of the tasks in the mission specification, therefore no conflicts arise but we lose information about the state of tasks not assigned to that particular robot. Unlike STAPU, where replanning only takes place in states that are not reachable under the sequential policy, auctioning does not enable the reallocation of tasks without replanning. For example, whenever a robot fails or a door is closed the auction must be re-run.

6.2 Benchmarks

Our test environments are inspired by the benchmarks already used for multi-agent path finding (MAPF) [59], and consist of a warehouse, a fully connected grid and an office. We use 3 variations of the warehouse environment: one where robots travel from the shelves to the depot, one where robots travel from the depot to the shelves and one where the initial and task locations are random. Similarly, we use 3 variations of the grid environment: one where robots travel left to right, one where robots travel right to left and one where initial and task locations are random. In order to test the use of global state features, we use variants of the warehouse environment with and without doors. The warehouse has 123 locations, and 100 locations for the version with doors. The office model is slightly smaller, with 60 locations. For most of our experiments, we fix the size of the grid environment to 11 × 11, but we also vary this size to evaluate scalability.

Missions comprise tasks to visit various target locations and the safety specification is to avoid a set of locations. By default, we assume 4 robots and 4 tasks, but we also present results for varying numbers of both. We model individual robot failures by introducing transitions to the designated failure state from some of a robot's local states with a fixed probability (we use 0.2 in our experiments). By default, we assume that most states (90%) can suffer failures, we also experiment with varying this percentage.

For each test configuration, we create 10 different variants, in which the initial robot locations, locations for tasks and failure states are chosen randomly. We illustrate a selection of the maps used in Figure 7. The experiments described in this section were run on an Intel i5 with 16GB of RAM running Linux (CentOS).

6.3 Results

We evaluate STAPU, focusing primarily on efficiency and scalability, but also the quality of generated policies, and compare to the baseline auction-based approach.

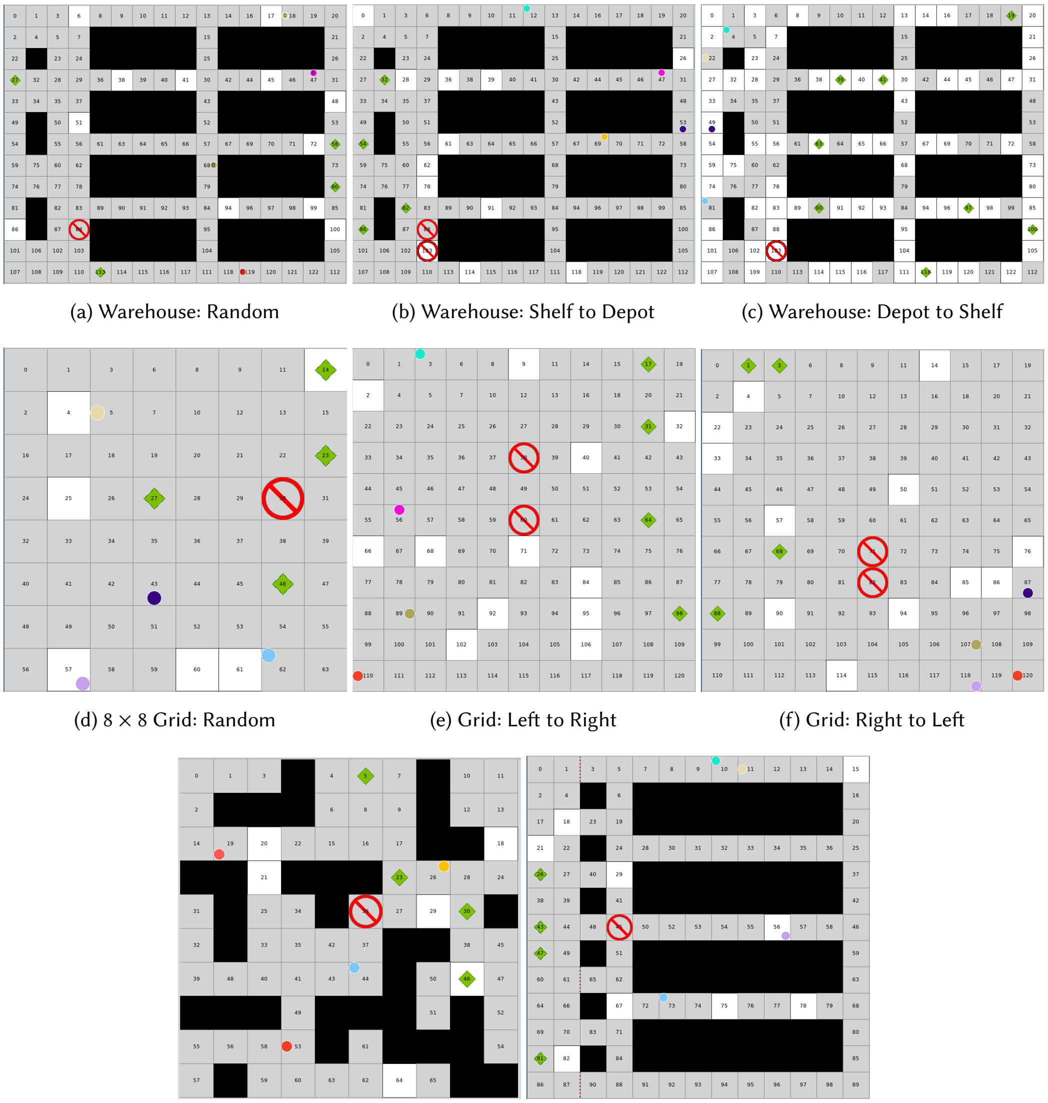
Both our implementations, STAPU and the baseline, consider the efficiency of policies by adding a cost, based on distance, to each action in the local models and minimising expected cumulative cost as a secondary objective. This is achieved by using nested value iteration (NVI) [36] to solve all MDPs, with maximising the expected number of achieved tasks as the primary objective, and minimising the expected cost as the secondary objective. In NVI, the secondary objective is used to *tie-break* between actions with the same value on the primary objective. For the auctioning baseline, both the expected number of tasks and the expected cost are used as the bid, and the expected cost is again used to tie-break between robots that have the same bid in terms of expected number of tasks.

Figure 8 shows the total time required for policy generation across the full set of test environments. We use a scatter plot, with STAPU on the x-axis and auctioning on the y-axis, so points above the diagonal indicate better performance (shorter times) for STAPU. We see that STAPU generally takes less time than auctioning. An exception is the office environment, where the model is smaller and auctioning is usually faster. STAPU solves instances of the team MDP, which is larger and slower to solve than the local robot MDPs solved during auctioning. However, the decoupling of task allocation and planning in the latter leads to more occurrences of replanning in order to reallocate tasks after a robot fails (and therefore a larger number of MDPs need to be built and solved). The number of times that replanning occurs is shown with a similar scatter plot in Figure 9 and shows a correlation with the timing results.

To check that faster performance by STAPU is not at the expense of generating policies of poorer quality, we also compare the probabilistic guarantees offered by the policies that STAPU and auctioning generate. Figure 10 plots the expected number of tasks for each approach, as computed from the generated joint policy. We see that these values are generally very similar for STAPU and auctioning.

Next, we consider how the time required by STAPU and auctioning is affected by various aspects of the test environments. For clarity of presentation we consider a fixed environment in each case, but we have observed similar patterns on the other benchmarks. Figure 11 shows the results. Since there are multiple random instances for each test environment, we use boxplots that show the mean and median values of each set and their range. The box shows the interquartile range and the ends indicate the outer-quartiles. The mean is represented by a triangle, outliers by diamonds and the median is the horizontal bar enclosed in the box.

Figure 11a shows times as we vary the *percentage of states in which failures can occur*. Larger numbers of failures lead to more replanning and thus higher times. The rate of increase in time is less steep for STAPU. For a clearer comparison



(a) Warehouse: Random

(b) Warehouse: Shelf to Depot

(c) Warehouse: Depot to Shelf

(d) 8 × 8 Grid: Random

(e) Grid: Left to Right

(f) Grid: Right to Left

(g) Office

(h) Small Warehouse (3 doors)

Fig. 7. Instances of the topological maps used as test environments for benchmarking. Initial locations of robots (4 in these examples) are denoted by coloured circles, task locations are marked by green diamonds and locations to be avoided for the safety specification are shown by a red forbidden sign. Failure locations (40% of locations in (c); 90% elsewhere) are shaded grey. Doors, as used in (h), are represented as a dashed brown line between two states.

of the difference in effect on STAPU and auctioning, Figure 11b shows relative times: the ratio of times for auctioning and STAPU (i.e., the speedup factor obtained with STAPU). As the percentage of failure states increase, we see that STAPU does better, up to 10 times faster in cases. This is because the sequential team model allows reallocation of tasks after failures and reduces the amount of replanning required. STAPU is also able to modify the existing team MDP for each replan, where as auctioning has to build and solve a new one each time. However, the team MDP is larger

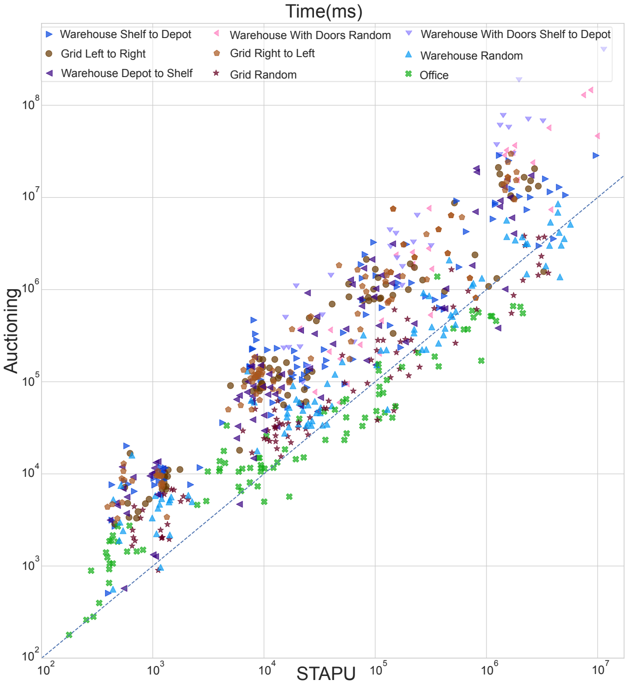


Fig. 8. Scatter plots showing the total runtime taken (in milliseconds) for STAPU (x-axis) vs. auctioning (y-axis) across the full set of test environments.

and slower to solve. For lower numbers of failure states, the additional overhead of solving larger MDPs in STAPU outweighs the gain and auctioning is faster.

In terms of the sizes of problems we consider, key variables are the numbers of robots, tasks and map locations. STAPU avoids generation of the full joint MDP, whose size is exponential in the numbers of both robots and tasks. The largest MDP solved is the team MDP, whose size is $O(n \cdot N_{loc} \cdot (N_{dfa})^m)$ for n robots, m tasks and where N_{loc} and N_{dfa} are the number of map locations and the maximum size of the DFA representing a task (where the latter is typically small). So the size of the MDP is no longer exponential in the number of robots and this is not the limiting factor for scalability. Instead, our primary concern is the computation time, which is influenced not just by the sizes of MDPs to be solved, but the number of times that replanning occurs.

Figure 11c shows the effect on time as the *number of robots* is increased. This increases the size of the team MDP for STAPU linearly, as discussed above, but has a more significant effect on the amount of replanning needed for both methods, so STAPU remains faster in general. Similar trends are seen in Figure 11d, which shows the times as the *number of tasks* increases, and in Figure 11e, where we vary the *number of map locations* (and therefore the size of each robot's local MDP). For the latter, we use fully connected square grids with increasing sizes.

Finally, we evaluate the performance of the two methods as the *number of global state features* increases. For this, we use a variation of the warehouse environment where the status of doors are modelled as global states. The default state of each door is unknown; and in order to go through a door, a robot must check whether the door is open or closed.

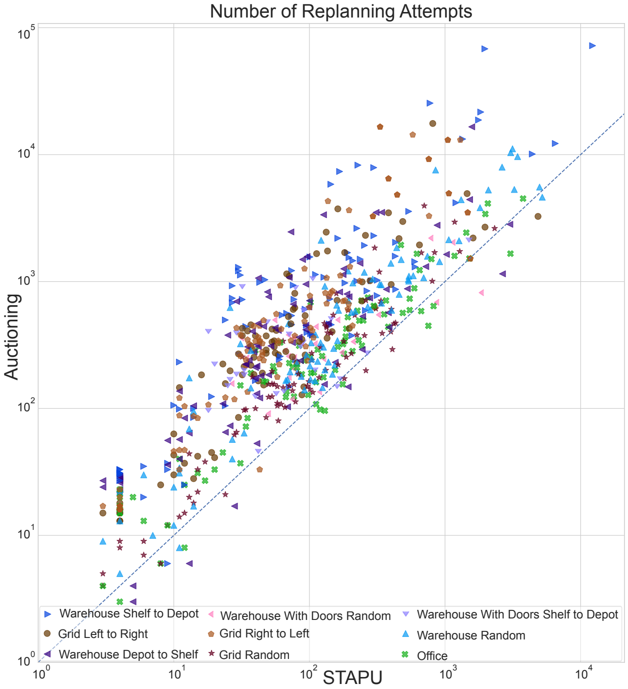


Fig. 9. Scatter plots showing the number of times replanning occurs for STAPU (x-axis) vs. auctioning (y-axis) across the full set of test environments.

STAPU's team model allows robots to transfer knowledge of this value. As seen in Fig 11f STAPU is significantly faster than the auctioning approach as the number of shared states increase.

7 CONCLUSION

We have presented an approach, STAPU, to simultaneous task allocation and planning for multi-robot systems operating in uncertain environments and prone to failures. Leveraging techniques from probabilistic model checking we are able to provide probabilistic guarantees on the performance of the team, in particular, optimising for and providing a guarantee on the expected number of achieved tasks by the team, whilst avoiding the construction of the full joint MDP. We also tackle the topic of task reallocation on failure, providing an *anytime* algorithm that incrementally finds states where reallocation is required, and builds policies from those states.

We have evaluated STAPU extensively with respect to efficiency and scalability, comparing it with a baseline auctioning method, a standard approach for multi-robot task allocation. Our experiments have shown that on average, STAPU is able to compute solutions faster than the baseline. The quality of STAPU's solutions, evaluated as probabilistic guarantees on the expected number of achieved tasks, is also similar to those of the baseline.

Future work includes modelling robot collisions, possibly treating them as reallocation states, and extending our approach to dynamic arrival of new tasks. Furthermore, we will exploit approaches to relax our assumptions of perfect communication and synchronisation, specifically in scenarios where the need for communication is sparse.

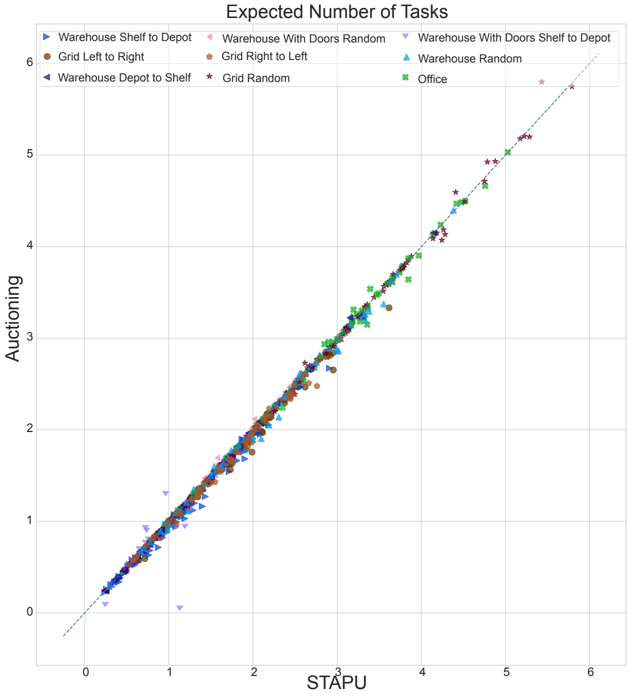


Fig. 10. Scatter plot of the expected number of completed tasks for STAPU (x-axis) vs. auctioning (y-axis) across the full set of test environments.

ACKNOWLEDGMENTS

This work was supported by UK Research and Innovation and EPSRC through the Robotics and Artificial Intelligence for Nuclear (RAIN) research hub [EP/R026084/1], the EPSRC Programme Grant 'From Sensing to Collaboration' [EP/V000748/1] and the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 834115, FUN2MODEL).

REFERENCES

[1] Christopher Amato, George Konidaris, Ariel Anders, Gabriel Cruz, Jonathan P How, and Leslie P Kaelbling. 2015. Policy Search for Multi-Robot Coordination under Uncertainty. In *Proceedings of Robotics: Science and Systems XI (RSS).*

[2] Christopher Amato, George Konidaris, Leslie P Kaelbling, and Jonathan P How. 2019. Modeling and planning with macro-actions in decentralized POMDPs. *Journal of Artificial Intelligence Research* 64 (2019), 817–859.

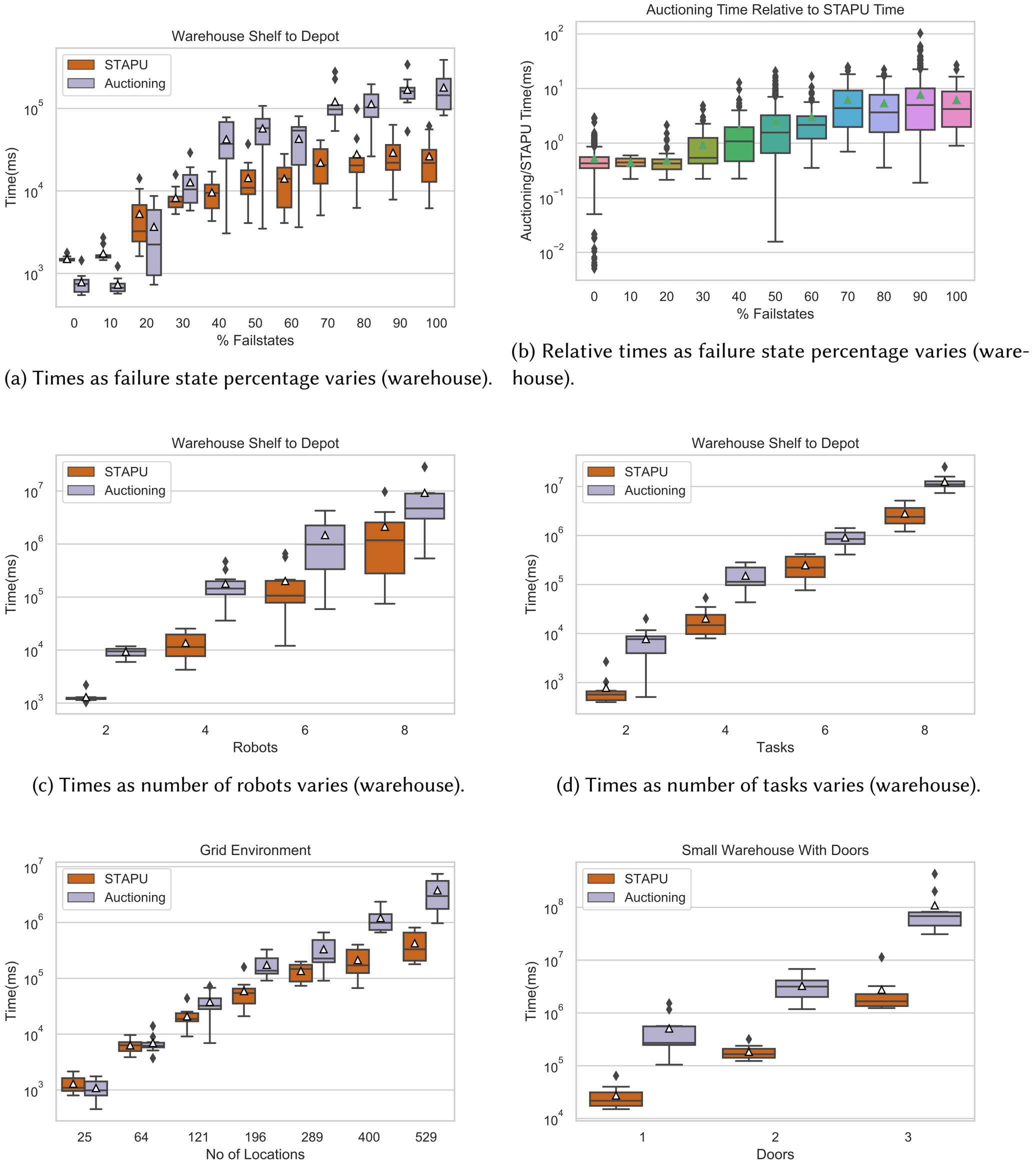
[3] D Atzmon, R Stern, A Felner, N Sturtevant, and Sven Koenig. 2020. Probabilistic Robust Multi-Agent Path Finding. In *Proceedings of the 30th International Conference on Automated Planning and Scheduling (ICAPS).*

[4] Carlos Azevedo, Bruno Lacerda, Nick Hawes, and Pedro Lima. 2020. Long-Run Multi-Robot Planning under Uncertain Action Durations for Persistent Tasks. In *Proceedings of the 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).*

[5] Fahiem Bacchus, Craig Boutilier, and Adam Grove. 1996. Rewarding behaviors. In *Proceedings of the 13th National Conference on Artificial Intelligence (AAAI)*. 1160–1167.

[6] Christel Baier and Joost-Pieter Katoen. 2008. *Principles of model checking.* MIT press.

[7] Andrea Bajcsy, Sylvia L Herbert, David Fridovich-Keil, Jaime F Fisac, Sampada Deglurkar, Anca D Dragan, and Claire J Tomlin. 2019. A scalable framework for real-time multi-robot, multi-human collision avoidance. In *2019 International Conference on Robotics and Automation (ICRA).* IEEE, 936–943.



(a) Times as failure state percentage varies (warehouse).

(b) Relative times as failure state percentage varies (warehouse).

(c) Times as number of robots varies (warehouse).

(d) Times as number of tasks varies (warehouse).

(e) Times as number of map locations varies (grid).

(f) Times as num. global features varies (warehouse+doors).

Fig. 11. Box plots comparing the runtime for STAPU and auctioning as different aspects of the benchmark models are varied.

[8] Maren Bennewitz, Wolfram Burgard, and Sebastian Thrun. 2001. Optimizing schedules for prioritized path planning of multi-robot systems. In *Proceedings of the 2001 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 271–276.

[9] Maren Bennewitz, Wolfram Burgard, and Sebastian Thrun. 2002. Finding and optimizing solvable priority schemes for decoupled path planning techniques for teams of mobile robots. *Robotics and Autonomous Systems* 41, 2-3 (2002), 89–99.

[10] Craig Boutilier. 1996. Planning, Learning and Coordination in Multiagent Decision Processes. In *Proceedings of the 6th Conference on Theoretical Aspects of Rationality and Knowledge (TARK)*.

[11] Ronen Brafman, Giuseppe De Giacomo, and Fabio Patrizi. 2018. LTLf/LDLf non-Markovian rewards. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 32.

[12] Jesus Capitan, Matthijs T.J. Spaan, Luis Merino, and Anibal Ollero. 2013. Decentralized multi-robot cooperation with auctioned POMDPs. *The International Journal of Robotics Research* 32, 6 (2013), 650–671. https://doi.org/10.1177/0278364913483345 arXiv:https://doi.org/10.1177/0278364913483345

[13] Ji Chen, Ruojia Sun, and Hadas Kress-Gazit. 2021. Distributed control of robotic swarms from reactive high-level specifications. In *Proceedings of the 2021 IEEE International Conference on Automation Science and Engineering (CASE)*. IEEE, 1247–1254.

[14] Daniel Claes, Frans Oliehoek, Hendrik Baier, and Karl Tuyls. 2017. Decentralised Online Planning for Multi-Robot Warehouse Commissioning. In *Proceedings of the 16th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*.

[15] Daniel Claes, Philipp Robbel, Frans A. Oliehoek, Karl Tuyls, Daniel Hennes, and Wiebe van der Hoek. 2015. Effective Approximations for Multi-Robot Coordination in Spatially Distributed Tasks. In *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*.

[16] Murat Cubuktepe, Zhe Xu, and Ufuk Topcu. 2020. Policy Synthesis for Factored MDPs with Graph Temporal Logic Specifications. In *Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*.

[17] Frits De Nijs, Erwin Walraven, Mathijs De Weerdt, and Matthijs Spaan. 2021. Constrained multiagent Markov decision processes: A taxonomy of problems and algorithms. *Journal of Artificial Intelligence Research* 70 (2021), 955–1001.

[18] Amy Fang and Hadas Kress-Gazit. 2022. Automated Task Updates of Temporal Logic Specifications for Heterogeneous Robots. In *Proceedings of the 2022 International Conference on Robotics and Automation (ICRA)*.

[19] F. Faruq, B. Lacerda, N. Hawes, and D. Parker. 2018. Simultaneous Task Allocation and Planning Under Uncertainty. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 3559–3564.

[20] Ariel Felner, Roni Stern Solomon, Eyal Shimony, Israel Eli Boyarski, Israel Meir Goldenberg, Guni Sharon, Nathan Sturtevant, Glenn Wagner, and Pavel Surynek. 2017. Search-Based Optimal Solvers for the Multi-Agent Pathfinding Problem: Summary and Challenges. In *Proceedings of the 2017 International Symposium on Combinatorial Search (SoCS)*.

[21] Anna Gautier, Bruno Lacerda, Nick Hawes, and Michael Wooldridge. 2023. Multi-Unit Auctions for Allocating Chance-Constrained Resources. In *Proceedings of the 37th AAAI Conference on Artificial Intelligence (AAAI)*.

[22] Anna Gautier, Bruno Lacerda, Nick Hawes, and Michael Wooldridge. 2023. Risk-constrained planning for multi-agent systems with shared resources. In *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*.

[23] Ivan Gavran, Rupak Majumdar, and Indranil Saha. 2017. Antlab: A Multi-Robot Task Server. *ACM Transactions on Embedded Computing Systems* 16, 5s (2017), 1–19.

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

[25] Meng Guo and Michael M. Zavlanos. 2018. Probabilistic Motion Planning Under Temporal Tasks and Soft Constraints. *IEEE Trans. Automat. Control* 63, 12 (2018), 4051–4066. arXiv:1706.05209

[26] Feifei Huang, Xiang Yin, and Shaoyuan Li. 2022. Failure-Robust Multi-Robot Tasks Planning under Linear Temporal Logic Specifications. In *Proc. 13th Asian Control Conference (ASCC'22)*. 1052–1059.

[27] Rodrigo Toro Icarte, Toryn Q Klassen, Richard Valenzano, and Sheila A McIlraith. 2022. Reward machines: Exploiting reward function structure in reinforcement learning. *Journal of Artificial Intelligence Research* 73 (2022), 173–208.

[28] Samarth Kalluraya, George J. Pappas, and Yiannis Kantaros. 2023. Resilient Temporal Logic Planning in the Presence of Robot Failures. In *Proc. 62nd IEEE Conference on Decision and Control (CDC'23)*.

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

[30] S Koenig, C Tovey, M Lagoudakis, V Markakis, and D Kempe. 2006. The power of sequential single-item auctions for agent coordination. In *Proceedings of the 21st AAAI Conference on Artificial Intelligence (AAAI)*.

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

[32] Orna Kupferman and Moshe Y Vardi. 2001. Model checking of safety properties. *Formal Methods in System Design* 19, 3 (2001).

[33] M. Kwiatkowska, G. Norman, and D. Parker. 2011. PRISM 4.0: Verification of Probabilistic Real-time Systems. In *Proceedings of the 23rd International Conference on Computer Aided Verification (CAV)*.

[34] Bruno Lacerda, Fatma Faruq, David Parker, and Nick Hawes. 2019. Probabilistic planning with formal performance guarantees for mobile service robots. *The International Journal of Robotics Research* 38, 9 (2019), 1098–1123.

[35] Bruno Lacerda and Pedro U. Lima. 2019. Petri net based multi-robot task coordination from temporal logic specifications. *Robotics and Autonomous Systems* 122 (2019), 103289.

[36] Bruno Lacerda, David Parker, and Nick Hawes. 2015. Nested Value Iteration for Partially Satisfiable Co-Safe LTL Specifications. In *AAAI Fall Symposium on Sequential Decision Making for Intelligent Agents*.

[37] Bruno Lacerda, David Parker, and Nick Hawes. 2015. Optimal Policy Generation for Partially Satisfiable Co-Safe LTL Specifications. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI).*

[38] M. Lahijanian and M. Kwiatkowska. 2016. Specification revision for Markov decision processes with optimal trade-off. In *Proceedings of the 55th IEEE Conference on Decision and Control (CDC).*

[39] Morteza Lahijanian, Matthew R. Maly, Dror Fried, Lydia E. Kavraki, Hadas Kress-Gazit, and Moshe Y. Vardi. 2016. Iterative Temporal Planning in Uncertain Environments With Partial Satisfaction Guarantees. *IEEE Transactions on Robotics* 32, 3 (2016).

[40] Steven M. Lavalle. 1998. *Rapidly-Exploring Random Trees: A New Tool for Path Planning.* Technical Report.

[41] Carlos E Luis, Marijan Vukosavljev, and Angela P Schoellig. 2020. Online trajectory generation with distributed model predictive control for multi-robot motion planning. *IEEE Robotics and Automation Letters* 5, 2 (2020), 604–611.

[42] Xusheng Luo, Yiannis Kantaros, and Michael M Zavlanos. 2021. An Abstraction-Free Method for Multi-Robot Temporal Logic Optimal Control Synthesis. *IEEE Transactions on Robotics* 37, 5 (2021).

[43] H. Ma and S. Koenig. 2016. Optimal Target Assignment and Path Finding for Teams of Agents. In *Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems (AAMAS).*

[44] H. Ma, T. K. S. Kumar, and S. Koenig. 2017. Multi-Agent Path Finding with Delay Probabilities. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI).*

[45] Masoumeh Mansouri, Bruno Lacerda, Nick Hawes, and Federico Pecora. 2019. Multi-Robot Planning Under Uncertain Travel Times and Safety Constraints. In *Proceedings of 28th International Joint Conference on Artificial Intelligence (IJCAI).* Macau, China.

[46] Felipe J. Montana, Jun Liu, and Tony J Dodd. 2017. Sampling-Based Path Planning for Multi-robot Systems with Co-Safe Linear Temporal Logic Specifications. In *Proceedings of the Joint 22nd International Workshop on Formal Methods for Industrial Critical Systems and 17th International Workshop on Automated Verification of Critical Systems (FMICS-AVoCS).* Springer International Publishing, 150–164.

[47] Lynne E Parker. 1998. ALLIANCE: An architecture for fault tolerant multirobot cooperation. *IEEE Transactions on Robotics and Automation* 14, 2 (1998), 220–240.

[48] A. Pnueli. 1981. The Temporal Semantics of Concurrent Programs. *Theoretical Computer Science* 13 (1981).

[49] Martin L. Puterman. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming* (1st ed.). John Wiley & Sons, Inc., New York, NY, USA.

[50] Yunus Emre Sahin, Petter Nilsson, and Necmiye Ozay. 2019. Multirobot coordination with counting temporal logics. *IEEE Transactions on Robotics* 36, 4 (2019), 1189–1206.

[51] Joris Scharpff, Diederik M. Roijers, Frans A. Oliehoek, Matthijs T. J. Spaan, and Mathijs M. De Weerdt. 2016. Solving Transition-Independent Multi-agent MDPs with Sparse Interactions. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence (AAAI).*

[52] Philipp Schillinger, Mathias Bürger, and Dimos V Dimarogonas. 2016. Decomposition of finite LTL specifications for efficient multi-agent planning. In *Proceedings of the 2016 International Symposium on Distributed Autonomous Robotic Systems (DARS).*

[53] Philipp Schillinger, Mathias Bürger, and Dimos V. Dimarogonas. 2018. Improving Multi-Robot Behavior Using Learning-Based Receding Horizon Task Allocation. In *Proceedings of Robotics: Science and Systems XIV (RSS).* https://doi.org/10.15607/RSS.2018.XIV.031

[54] Philipp Schillinger, Mathias Bürger, and Dimos V. Dimarogonas. 2018. Simultaneous task allocation and planning for temporal logic goals in heterogeneous multi-robot systems. *The International Journal of Robotics Research* 37, 7 (2018), 818–838. https://doi.org/10.1177/0278364918774135

[55] Ankit Shah, Shen Li, and Julie Shah. 2020. Planning with uncertain specifications (puns). *IEEE Robotics and Automation Letters* 5, 2 (2020), 3414–3421.

[56] Guni Sharon, Roni Stern, Ariel Felner, and Nathan R. Sturtevant. 2015. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence* 219 (2015), 40–66.

[57] Stephen L Smith, Jana Tumova, Calin Belta, and Daniela Rus. 2011. Optimal path planning for surveillance with temporal-logic constraints. *The International Journal of Robotics Research* 30, 14 (2011).

[58] M. T. J. Spaan, N. Gonçalves, and J. Sequeira. 2010. Multirobot coordination by auctioning POMDPs. In *Proceedings of the 2010 IEEE International Conference on Robotics and Automation (ICRA).* 1446–1451. https://doi.org/10.1109/ROBOT.2010.5509614

[59] Roni Stern, Nathan Sturtevant, Ariel Felner, Sven Koenig, Hang Ma, Thayne Walker, Jiaoyang Li, Dor Atzmon, Liron Cohen, T. K. Satish Kumar, Eli Boyarski, and Roman Bartak. 2019. Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks. In *Proceedings of the 2019 International Symposium on Combinatorial Search (SoCS).*

[60] Charlie Street, Sebastian Pütz, Manuel Mühlig, Nick Hawes, and Bruno Lacerda. 2022. Congestion-Aware Policy Synthesis for Multirobot Systems. *IEEE Transactions on Robotics* 38, 1 (2022).

[61] Jana Tumova and Dimos V. Dimarogonas. 2016. Multi-agent planning under local LTL specifications and event-based synchronization. *Automatica* 70, October (2016).

[62] Jur P Van Den Berg and Mark H Overmars. 2005. Prioritized Motion Planning for Multiple Robots. In *Proceedings of the 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).* IEEE, 430–435.

[63] Alberto Viseras, Valentina Karolj, and Luis Merino. 2017. An asynchronous distributed constraint optimization approach to multi-robot path planning with complex constraints. In *Proceedings of the 32nd ACM Symposium on Applied Computing (SAC).*

[64] Glenn Wagner and Howie Choset. 2017. Path Planning for Multiple Agents Under Uncertainty. In *Proceedings of the 27th International Conference on Automated Planning and Scheduling (ICAPS).*

[65] Wenying Wu, Subhrajit Bhattacharya, and Amanda Prorok. 2020. Multi-Robot Path Deconfliction through Prioritization by Path Prospects. In *Proceedings of the 2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 9809–9815.

[66] Shiqi Zhang, Yuqian Jiang, Guni Sharon, and Peter Stone. 2017. Multirobot Symbolic Planning Under Temporal Uncertainty. In *Proceedings of the 16th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*.