Simultaneous Task Allocation and Planning for Temporal Logic Goals in Heterogeneous Multi-Robot Systems

Journal Title
XX(X):1–19
©The Author(s) 2016
Reprints and permission:
sagepub.co.uk/journalsPermissions.nav
DOI: 10.1177/ToBeAssigned
www.sagepub.com/
SAGE

Philipp Schillinger[1,2], Mathias Bürger[1], and Dimos V. Dimarogonas[2]

Abstract

This paper describes a framework for automatically generating optimal action-level behavior for a team of robots based on Temporal Logic mission specifications under resource constraints. The proposed approach optimally allocates separable tasks to available robots, without requiring a-priori an explicit representation of the tasks or the computation of all task execution costs. Instead, we propose an approach for identifying sub-tasks in an automaton representation of the mission specification and for simultaneously allocating the tasks and planning their execution. The proposed framework avoids the need of computing a combinatorial number of possible assignment costs, where each computation itself requires solving a complex planning problem. This can improve computational efficiency compared to classical assignment solutions, in particular for on-demand missions where task costs are unknown in advance. We demonstrate the applicability of the approach with multiple robots in an existing office environment and evaluate its performance in several case study scenarios.

Keywords

LTL, Robotics, Behavior Synthesis, Constrained Planning, Multi-Agent Planning, Task Allocation

1 Introduction

High-level specifications of goals and rules allow robotic systems to automatically generate behaviors that are guaranteed to be correct. As for example summarized by Belta et al. (2007), this simplifies behavior design by shifting the focus away from explicitly programming execution plans in order to reach a goal. Instead, the goal itself is specified and required actions can be planned by the robotic system. For example, Lacerda et al. (2014), Amato et al. (2015), and Lahijanian et al. (2016) demonstrate the successful application of different planning methods to a range of various robotic systems.

Linear Temporal Logic (LTL) provides a useful formalism for such high-level specifications, see Baier and Katoen (2008). Specifically, LTL not only allows a user to specify a Boolean goal condition, evaluated at a single point in time, but also to specify temporal relationships that are evaluated over the full sequence of actions. This is a far from trivial extension to standard high-level specifications, making them applicable for highly sophisticated robotic applications Kress-Gazit et al. (2009). At the same time, the goal can be provided in a user-friendly and efficient way. An LTL specification is already a more compact formulation than describing a specific task as a transition system or a similar model. Even more, LTL itself can, for example, be generated from structured English Kress-Gazit et al. (2008), Autili et al. (2015). Combining these properties constitutes LTL a suitable formalism to specify on-demand tasks for a deployed robotic system.

In many applications, a robotic system benefits from using not only one, but rather multiple robots to carry out the required tasks. See for example Ma et al. (2017) for a discussion of research directions in the area of multi-robot path finding. While a multi-robot system is typically more efficient, planning behaviors to fulfill complex tasks as well as assigning them to the individual robots is computationally hard, see Gerkey and Matarić (2004).

The complexity of planning for multi-robot systems, as for example detailed by Durfee and Zilberstein (2013), mainly results from two problem properties. First, the state space of a multi-robot system grows exponentially in the number of robots. Second, not only actions need to be planned, but also allocation of parts of the mission needs to be considered. These two problems have traditionally been considered independently.

We propose here an approach termed *Simultaneous Task Allocation and Planning (STAP)*. The central aspect of STAP is to construct a *team model* which is not only used to plan execution, but also contains choices for decomposing the mission and allocating the decomposed parts to different robots. Consequently, it is possible to utilize the interplay of allocation and planning, which bears the potential to improve the efficiency of finding an optimal solution. This approach is primarily well-suited in cases where task execution plans cannot be computed in advance, as it is for example the case for LTL-based planning of on-demand missions.

STAP is motivated by service robot use cases, e.g., in an office environment. Typical use cases include transportation

[1]Bosch Center for Artificial Intelligence, Renningen, Germany.
[2]KTH Centre for Autonomous Systems and ACCESS Linnaeus Center, EES, KTH Royal Institute of Technology, Stockholm, Sweden.

Corresponding author:
Philipp Schillinger, Bosch Center for Artificial Intelligence, Robert-Bosch-Campus 1, DE-71272 Renningen, Germany.
Email: philipp.schillinger@de.bosch.com

of documents and supplies, service tasks like emptying paper bins, and various assistance tasks. Specifically, we assume that a general model of the system, such as a topological map of the environment and internal states of the robots, exists. A mission specification may be given to the system at any time with the aim of using the available robots to satisfy the specification in the shortest possible time. In order to improve long-term performance of the system, as well as to handle more types of missions, continuous resource constraints and requirements have to be considered. We use LTL for the mission specifications in order to express more complex, temporally extended requirements on the synthesized behaviors as discussed previously. Throughout this paper, we consider the use case of emptying paper bins as an illustrative example for explaining the technical details.

The efficiency of STAP requires a set of assumptions which are primarily motivated by the service robot use case. We assume here that actions are deterministic. This is justified since the environment is known and local recoveries can be executed to ensure the success of an action. Furthermore, we assume a central planning server since, on the one hand, the robots need to communicate with a central server anyway when ready for a new mission and, on the other hand, a central server can usually have much more computational power than a set of mobile robots. Finally, we only allow finite LTL goal specifications, which more intuitively reflect the notion of mission completion.

1.1 Related Work

In the multi-agent literature, several approaches exist to efficiently and optimally allocate a set of known tasks to a set of agents, as for example summarized by Burkard and Cela (1999), Korsah et al. (2013), Pentico (2007), Yan et al. (2013). A widely-used method for planning task allocation is to construct a Mixed-Integer Linear Program, for example as proposed by Gombolay et al. (2013). Another popular method for task allocation is a market-based approach. For instance, Zlot and Stentz (2005) formulate tasks as trees where subtasks can be contracted to different agents. In contrast, Agarwal et al. (2014) propose an evolutionary algorithm to optimize allocation of a set of tasks to robots with different capabilities. Based on the vehicle routing problem (VRP), Karaman and Frazzoli (2008) propose an approach to allocate a set of spacial tasks to agents for a fragment of Metric Temporal Logic specifications. Turpin et al. (2015) approximate optimal solutions to the VRP for a min-max objective function.

The above approaches assume that the cost or utility for executing one task with a particular agent is known in advance. This is a potential limitation when task execution needs to be planned as well. Each cost calculation is a planning problem itself in order to find the optimal task execution. This can be inefficient since most of these plans are not executed in the end.

Another aspect is that the decomposition of a goal specification into multiple tasks is not necessarily explicitly given. Guo and Dimarogonas (2015), for example, assume task specifications given explicitly to the single agents. Especially when planning for cost-optimal execution, it is a prerequisite that all options to decompose a specification are known. In our previous work, Schillinger et al. (2016a), we propose an approach for automatically identifying all possible decomposition choices of a specification into independent tasks.

There exist other approaches in the direction of STAP. Chen et al. (2012), Ulusoy et al. (2013) construct a team model and use the formalism of trace-closed languages to identify the individual tasks by projecting the solution onto each of the agents. Nikou et al. (2016) augment agent-specific goals with a global goal specification to be satisfied by all agents. These approaches construct a product model of the team which has two relevant drawbacks. First, the team model size grows exponentially with the number of agents and second, actions of the agents need to be synchronized during execution. Tumova and Dimarogonas (2016) propose the construction of a reduced product model where only cooperative actions need to be synchronized under the assumption that task allocation is already known. Nissim and Brafman (2014) present an efficient approach to solve multi-agent planning problems based on the *MA-STRIPS* formulation in Brafman and Domshlak (2008) by a distributed, privacy-preserving version of the heuristic forward search based on Helmert (2006).

The specific problem characteristics of interest are summarized by the following three aspects based on the target use cases. First, we assume that mission specifications are provided as LTL formulas and only assume finite LTL specifications like co-safe LTL, Kupferman and Vardi (2001), or LTL_f, De Giacomo and Vardi (2013), which is sufficient for the multi-robot applications in hand. Second, we explicitly consider required resources and resource consumption of actions. Shiroma and Campos (2009) consider shared resources such as limited communication bandwidth, but do not model resource consumption. Irnich and Desaulniers (2005) address resource constraints for general shortest path problems. In the case of uncertain action outcomes, de Nijs et al. (2017) provide guarantees on the satisfaction of resource constraints. In the context of planning surveillance tasks given as temporal logic specifications, Leahy et al. (2016) explicitly consider battery constraints and include charging actions in the model definition. Third, by modeling costs as the required execution time, the goal is to minimize the maximum of agent costs instead of their sum like most of the existing approaches do. This can be addressed by multi-objective optimization, see for example Gandibleux et al. (2006) or Paixão and Santos (2013), where the cost of each agent is considered as one objective as proposed in our previous work Schillinger et al. (2017).

1.2 Contributions

The theoretical contributions of this paper are as follows. (1) We propose an approach for *Simultaneous Task Allocation and Planning (STAP)* in order to combine the planning of tasks with their allocation to agents. Our results show that this leads to a more efficient solution than ignoring their interplay and threating them as separate problems. (2) We integrate previous results regarding automated LTL decomposition as well as resource-constrained multi-agent planning into a single STAP framework in order to more efficiently handle a team size larger than two or three robots. (3) We evaluate relevant aspects of the planning

performance on real-world scenarios to provide an insight into the characteristics of the problems and demonstrate practical applicability.

The previous results referred to in contribution (2) specifically concern our results in Schillinger et al. (2016a) on the decomposition of LTL specifications, which we integrate for the case that only a single mission specification is given. This extends our approach presented here to cases where no set of independent tasks is explicitly defined. Furthermore, we extend our approach in Schillinger et al. (2017) on resource-constrained multi-agent planning to utilize the specific structure of the team model and to support resource constraints as part of the LTL specification.

The rest of this paper is organized as follows. Section 2 recapitulates the formal basis for LTL specifications including notation and semantics. Section 3 discusses the decomposition of LTL missions. Section 4 presents the construction of the team model based on decomposition choices and integrates resources into the model. Section 5 addresses the planning algorithm for our proposed STAP approach and discusses its properties. Section 6 provides an insight into the ROS implementation which is used for the system. Section 7 presents an existing office environment and multiple typical scenarios which we discuss to evaluate the properties of the proposed STAP framework.

2 Preliminaries

In the following, we provide an overview of the concepts which form the basis for our presented work. Based on these concepts, goal specifications and the system model can be expressed.

2.1 LTL Semantics

Linear Temporal Logic (LTL) is a mathematical specification logic which is able to capture temporal relationships. Originally resulting from the field of model checking and verification, see Baier and Katoen (2008), LTL applied to robotic behavior planning provides a formalism to specify the expected behavior in an unambiguous way. As such, an LTL specification can be used to describe the result of an expected behavior, while the way to achieve this result is automatically derived by the system.

An LTL formula ϕ is defined over a set of *atomic propositions* $\pi \in \Pi$, which can be *true* $\top$ or *false* $\bot$. The syntax

$$\phi := \pi \mid \neg\phi_1 \mid \phi_1 \wedge \phi_2 \mid \bigcirc\phi_1 \mid \phi_1 \,\mathcal{U}\, \phi_2 \mid \phi_1 \,\mathcal{R}\, \phi_2 \quad (1)$$

contains the Boolean operators $\neg$ "not" and $\wedge$ "and", as well as the temporal operators $\bigcirc$ "next", $\mathcal{U}$ "until", and $\mathcal{R}$ "release". To express temporal relationships, the semantics of the formula ϕ are defined over a sequence $\sigma\colon \mathbb{N} \to 2^{\Pi}$ and $\sigma(t) \subseteq \Pi$ at time t contains all atomic propositions which are true. In particular, the semantics of the satisfaction relation $\vDash$ for the above operators and a sequence σ are recursively defined as follows.

- $\sigma(t) \vDash \pi$ iff $\pi \in \sigma(t)$
- $\sigma(t) \vDash \neg\phi_1$ iff $\sigma(t) \nvDash \phi_1$
- $\sigma(t) \vDash \phi_1 \wedge \phi_2$ iff $\sigma(t) \vDash \phi_1$ and $\sigma(t) \vDash \phi_2$
- $\sigma(t) \vDash \bigcirc\phi_1$ iff $\sigma(t+1) \vDash \phi_1$
- $\sigma(t) \vDash \phi_1 \,\mathcal{U}\, \phi_2$ iff $\exists t_2 \geq t$ such that $\sigma(t_2) \vDash \phi_2$ and $\forall t_1 \in [t, t_2)$ it holds that $\sigma(t_1) \vDash \phi_1$
- $\sigma(t) \vDash \phi_1 \,\mathcal{R}\, \phi_2$ iff $t_1 = \infty$ or $\exists t_1 \geq t$ such that $\sigma(t_1) \vDash \phi_1$ and $\forall t_2 \in [t, t_1)$ it holds that $\sigma(t_2) \vDash \phi_2$

In addition, the following derived operators are defined to extend the above operators.

- "or": $\phi_1 \vee \phi_2 := \neg(\neg\phi_1 \wedge \neg\phi_2)$
- "implies": $\phi_1 \implies \phi_2 := \neg\phi_1 \vee \phi_2$
- "eventually": $\Diamond\phi_1 := \top \,\mathcal{U}\, \phi_1$
- "always": $\Box\phi_1 := \bot \,\mathcal{R}\, \phi_1$

A special case of the satisfaction relation $\sigma \vDash \phi$ where σ has length one and ϕ only contains Boolean operators is called *Boolean satisfaction*.

In general, a sequence σ might need to be infinite in order to satisfy a formula. For example, consider the formula $\phi_{\text{osc}} = \Diamond\pi_1 \wedge \Box(\pi_1 \implies \Diamond\pi_2) \wedge \Box(\pi_2 \implies \Diamond\pi_1) \wedge \Box\neg(\pi_1 \wedge \pi_2)$. This formula describes an oscillation between π_1 and π_2 and consequently, cannot be satisfied by any finite sequence.

In the case that there exists a finite sequence which satisfies an LTL formula, this formula is called finite. There are several fragments and interpretations of LTL, e.g., *co-safe LTL* by Kupferman and Vardi (2001) and LTL_f by De Giacomo and Vardi (2013), which ensure that specified LTL formulas are finite. In the following, we only address finite LTL formulas since this is sufficient for our considered use cases.

Example 1 (LTL Mission Specification). Consider the scenario of emptying a paper bin next to an office desk. We define the set of atomic propositions Π to mark locations of interest and denote robot states as summarized in Table 1. Based on these propositions, an LTL mission specification can be formulated as follows:

$$\begin{aligned}\phi = \Diamond(&desk \wedge default \\ &\wedge \bigcirc((carrybin \,\mathcal{U}\, dispose) \wedge \Diamond default)) \\ &\wedge \Diamond(desk \wedge emptybin \wedge \bigcirc(desk \wedge default)) \\ &\wedge \Box(carrybin \implies \neg public)\end{aligned} \quad (2)$$

The first line requires the robot to be next to the desk while not carrying anything. As denoted by the *eventually* operator

Proposition	Description
$desk$	Location where the bin should be collected.
$service$	Location where garbage can be disposed.
$storage$	Location where empty bins might be stored.
$public$	Location with public access.
$default$	Normal robot state, not carrying anything.
$carrybin$	Robot carrying a full paper bin.
$emptybin$	Robot equipped with an empty paper bin.
$dispose$	Robot successfully disposed garbage.

Table 1. Propositions $\Pi = \{desk, \ldots, dispose\}$ as used for the examples throughout the paper.

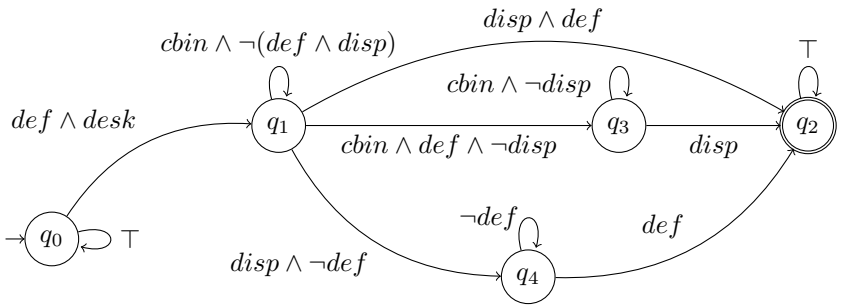

Figure 1. Example NFA for the formula $\phi_1 = \Diamond(desk \wedge default \wedge \bigcirc((carrybin\ \mathcal{U}\ dispose) \wedge \Diamond default))$. The propositions are abbreviated for improved illustration. The automaton has been constructed by using Spot (https://spot.lrde.epita.fr).

$\Diamond$, this might happen at any time, but is required in order to fulfill ϕ. The second line represents the process of emptying the bin and is required to happen next when being at the desk, as denoted by the *next* operator $\bigcirc$. The process is described as carrying the bin until garbage has been disposed and eventually putting the bin away again to reach the $default$ state. In the third line, it is additionally specified that an empty paper bin should be placed next to the desk. Note that this part is, for example, fulfilled by returning the emptied bin back to the desk in order to put it away as required in the second line. Finally, a constraint is specified in the fourth line which requires the robot to always avoid public areas while carrying a bin.

A sequence σ to satisfy ϕ is, for example, given by

$$\sigma = \{default\}\{public, default\}\{desk, default\}\{desk, carrybin\}\{carrybin\}\{service, dispose\}\{emptybin\}\{desk, emptybin\}\{desk, default\}.$$

Note that, although we defined in Table 1 that garbage is disposed at service locations, this is not specified by ϕ and thus, not required for fulfilling it. $\Diamond$

2.2 LTL Automaton

An important and useful property of LTL specifications is the fact that they can be translated into an equivalent automaton representation of the specification. In this context, equivalence is to be understood as identifying the same sequences σ to satisfy the respective specification. First, we define the automaton to be constructed as for example described by Baier and Katoen (2008).

Definition 1. NFA. *A nondeterministic finite automaton is given as the tuple $\mathcal{F} := (Q, Q_0, \alpha, \delta, F)$ consisting of*
(1) *a set of states Q,*
(2) *a set of initial states $Q_0 \subseteq Q$,*
(3) *a set of Boolean formulas α over $\pi \in \Pi$,*
(4) *transition conditions $\delta\colon Q \times Q \to \alpha$,*
(5) *a set of accepting (final) states $F \subseteq Q$.*

Note that our notation of the transition relation is non-standard in order to simplify notation throughout the paper. For two states $q_i, q_j \in Q$, we denote the absence of a transition by $\delta(q_i, q_j) := \bot$. Accordingly, there exists a transition if $\delta(q_i, q_j) \neq \bot$ and the boolean function $\delta(q_i, q_j)$ denotes the transition condition.

A sequence σ over propositions describes a sequence of states $q \in Q$, called a *run* $\rho\colon \mathbb{N} \cup \{0\} \to Q$. A run ρ is called *feasible* if it starts in an initial state $\rho(0) = q_0$ with $q_0 \in Q_0$ and if all transition conditions are satisfied along the run $\sigma(t) \vDash \delta(\rho(t-1), \rho(t))$ for all t where $\vDash$ denotes Boolean satisfaction as defined above. A run ρ is called *accepting* if it is feasible and ends in an accepting state $q_n \in F$.

If σ does not describe a feasible run, we say σ *violates* the LTL specification. If σ describes an accepting run, we say σ *satisfies* the LTL specification. Given by the construction of $\mathcal{F}$ from an LTL specification ϕ, it can be shown that σ indeed describes an accepting run if and only if $\sigma \vDash \phi$, see for example Baier and Katoen (2008). In the case that σ describes a feasible but not an accepting run, it does not satisfy the LTL specification. But under the assumption that $\mathcal{F}$ does not contain dead ends except for accepting states, σ forms a prefix of an accepting run and can be extended to a sequence satisfying the specification. Thus, we refer to this case by saying that σ *partially satisfies* ϕ.

Example 2 (Automaton Representation). Figure 1 depicts the NFA $\mathcal{F}$ equivalent to the first part of the LTL specification ϕ from Example 1 given by $\phi_1 = \Diamond(desk \wedge default \wedge \bigcirc((carrybin\ \mathcal{U}\ dispose) \wedge \Diamond default))$. Considering again the sequence σ as given in Example 1, a run ρ in $\mathcal{F}$ is given by $\rho = q_0q_0q_0q_1q_1q_1q_4q_4q_4q_2$. Some of the states in ρ are repeated since the transition conditions to other states are not immediately satisfied. For example, $\sigma(5) = \{carrybin\} \nvDash dispose \wedge \neg default = \delta(q_1, q_4)$, but still $\sigma(5) \vDash \delta(q_1, q_1) = \delta(\rho(4), \rho(5))$ such that ϕ_1 is not violated at $t = 5$. $\Diamond$

An NFA not only provides an illustrative way of checking if the corresponding LTL formula is fulfilled, but also indicates possible alternative sequences to fulfill the formula. In the following, we will refer to the set of alternative accepting sequences as *strategies*.

3 Mission Specification

We propose an approach for *Simultaneous Task Allocation and Planning (STAP)* in order to increase efficiency of synthesizing the behavior for a group of agents from a mission specification. Specifically, we assume a mission $\mathcal{M}$, given as a Linear Temporal Logic (LTL) specification ϕ, and present an approach to allocate independent parts, called tasks $\mathcal{T}_i$, of the mission to the available agents at the same time as planning how the respective agents should execute their potentially allocated parts.

A mission $\mathcal{M}$ may be given as a set of tasks $\mathcal{M} := \{\mathcal{T}_1, \ldots, \mathcal{T}_n\}$, called a *decomposition*. $\mathcal{M}$ can as well be

given as an LTL formula ϕ or equivalently expressed as an NFA $\mathcal{F}$. Similarly, a task $\mathcal{T}_i$ can be specified by an LTL formula $\phi^{(i)}$ or an NFA $\mathcal{F}^{(i)}$. In the rest of this section, we investigate the relationship between ϕ (or $\mathcal{F}$) and $\phi^{(i)}$ (or $\mathcal{F}^{(i)}$).

Specifically, a decomposition $\mathcal{M} = \{\mathcal{T}_1, \ldots, \mathcal{T}_\mathrm{n}\}$ implies two *decomposition properties* to be fulfilled by all of the tasks:

- **Independence** – (Non-)execution of one task $\mathcal{T}_i$ must not violate another task $\mathcal{T}_j$.
- **Completeness** – Completion of all tasks $\mathcal{T}_1, \ldots, \mathcal{T}_n$ implies completion of the mission $\mathcal{M}$.

While the second property is rather intuitive, the first one is motivated by the fact that an independently acting agent cannot rely on what others do and especially, without any coordination, execution cannot assumed to be synchronized between the agents.

Using the terminology introduced in Section 2.2, the above properties can be expressed by saying that any strategy for a strict subset of tasks $\mathcal{T}_\subset \subset \{\mathcal{T}_1, \ldots, \mathcal{T}_\mathrm{n}\}$, i.e., satisfying $\phi^{(i)}$ for all tasks $\mathcal{T}_i \in \mathcal{T}_\subset$, partially satisfies ϕ. A strategy for the full set of tasks $\mathcal{M} = \{\mathcal{T}_1, \ldots, \mathcal{T}_\mathrm{n}\}$ satisfies ϕ.

Consequently, completing a set of tasks can be associated with reaching a certain state in the mission NFA $\mathcal{F}$. However, not every state implies completion of a set of tasks when requiring the above properties. For this reason, we define a *decomposition set* of states as follows and say that decomposition at state q is possible if and only if q is in the decomposition set.

Definition 2. Decomposition Set. *The decomposition set $D \subseteq Q$ of the NFA $\mathcal{F}$ contains all states q which can be associated with completing a set of tasks such that these tasks are a subset of the decomposition $\mathcal{T}_1, \ldots, \mathcal{T}_\mathrm{n}$.*

Based on this decomposition set, the final planning model can be augmented to contain all possible decomposition choices. This can then be used for efficiently planning the optimal decomposition and corresponding allocation of tasks to agents at the same time as planning action sequences to execute the mission.

The specific construction of D is discussed in the following two subsections for two different cases. First, we assume that a set of LTL tasks $\mathcal{T}_1, \ldots, \mathcal{T}_n$ which defines $\mathcal{M}$ is explicitly given and show how to use simultaneous task allocation and planning instead of planning all allocation options separately. Second, we show how simultaneous task allocation and planning can even be used in the case that no set of tasks is given, but only the single LTL mission specification ϕ. In this case, the decomposition set is automatically derived directly from ϕ.

3.1 Explicit Tasks

First, we assume the simplified case that a set of tasks $\mathcal{T}_1, \ldots, \mathcal{T}_n$ is explicitly given. Then, we can construct a product of the NFAs $\mathcal{F}^{(1)}, \ldots, \mathcal{F}^{(n)}$ from tasks $\mathcal{T}_1, \ldots, \mathcal{T}_n$. Note that a product of NFAs is the equivalent automaton of the conjunction of their LTL formulas $\phi = \phi^{(1)} \wedge \ldots \wedge \phi^{(n)}$. Consequently, the decomposition properties are guaranteed by constructing the NFA product.

Definition 3. NFA Product. *The product of two NFAs $\mathcal{F} = \mathcal{F}^{(i)} \otimes \mathcal{F}^{(j)} := (Q, Q_0, \alpha, \delta, F)$ is constructed as*
(1) a set of states $Q = Q^{(i)} \times Q^{(j)}$,
(2) a set of initial states $Q_0 = \{(q_i, q_j) \in Q \colon q_i \in Q_0^{(i)}, q_j \in Q_0^{(j)}\}$,
(3) a set of Boolean formulas $\alpha = \{a_i \wedge a_j \colon a_i \in \alpha^{(i)}, a_j \in \alpha^{(j)}\}$,
(4) transition conditions $\delta \colon Q \times Q \to \alpha$ defined as below,
(5) a set of accepting states $F = \{(q_i, q_j) \in Q \colon q_i \in F^{(i)}, q_j \in F^{(j)}\}$.

The transition conditions of the product $\mathcal{F}$ need to capture the conditions of both NFAs $\mathcal{F}^{(i)}$, $\mathcal{F}^{(j)}$ and thus, are given by $\delta \colon ((q_i^\mathrm{s}, q_j^\mathrm{s}), (q_i^\mathrm{t}, q_j^\mathrm{t})) \mapsto \delta^{(i)}(q_i^\mathrm{s}, q_i^\mathrm{t}) \wedge \delta^{(j)}(q_j^\mathrm{s}, q_j^\mathrm{t})$. Specifically, a transition in $\mathcal{F}$ requires that there is a transition in both included NFAs, since otherwise, $\delta^{(i)} = \bot$ implies that $\delta = \bot$.

Furthermore, note that a product of multiple NFAs can be constructed by applying Definition 3 iteratively. In this case, for simplicity of notation, we refer to a state as the n-tuple $q := (q_1, \ldots, q_n) \in Q$ for q_i being the state component of NFA $\mathcal{F}^{(i)}$.

Finally, the decomposition set D from Definition 2 can be constructed as follows:

$$D := \{q \in Q \colon q_i \in Q_0^{(i)} \cup F^{(i)}, \forall q_i\}. \tag{3}$$

According to Equation (3), decomposition at a state is possible if and only if all tasks are either not yet started or already finished. Essentially, this prevents splitting of single tasks and preserves them as indivisible units, but enables to assign different tasks to different agents.

3.2 Automated Decomposition

In many applications, the set of tasks $\mathcal{T}_1, \ldots, \mathcal{T}_\mathrm{n}$ is either not explicitly given or can be partitioned further. Therefore, we describe here a method to automatically compute D from a single LTL mission specification ϕ. This method is based on previous results presented in Schillinger et al. (2016a). We summarize its application here, but refer to the related reference for further details and proofs regarding LTL decomposition.

The automated decomposition is essentially the opposite direction of the case described in the previous section and gives a non-trivial solution, i.e., more than one task, if ϕ implicitly contains parts which can be executed independently. Consider the following example for illustration of a decomposition into independent parts.

Example 3 (Decomposition). Assume a mission $\mathcal{M}$ given by the specification ϕ as introduced in Example 1. If executed by a single robot, the most efficient way to complete the mission might be to return the emptied bin to the desk. However, as discussed before, this is not explicitly required by the specification. When given to a team of at least two robots, one robot can pick up the bin and empty it, while another robot already brings an empty bin to the desk. Consequently, the mission $\mathcal{M}$ can be understood as consisting of two independent tasks. First, as task $\mathcal{T}_1$, the garbage in the full bin needs to be disposed and second, as task $\mathcal{T}_2$, an empty bin needs to be provided at the desk. Clearly both tasks need to be completed in order to fulfill

$\mathcal{M}$. But their execution does not depend on each other and it is in fact possible to complete $\mathcal{T}_2$ before $\mathcal{T}_1$, even for a single robot. ◇

The decomposition properties, independence and completeness, are captured by the following definition.

Definition 4. LTL Decomposition (Schillinger et al. (2016a)). *Let $\phi^{(i)}$ with $i \in \{1, ..., n\}$ be a set of finite LTL specifications for the tasks $\mathcal{T}_i$ and σ_i denote any sequence such that $\sigma_i \vDash \phi^{(i)}$. These tasks are called a decomposition of the mission $\mathcal{M}$ with the finite LTL specification ϕ if and only if:*

$$\sigma_{j_1} ... \sigma_{j_i} ... \sigma_{j_n} \vDash \phi$$

for all permutations of $j_i \in \{1, ..., n\}$ and all respective sequences σ_i.

The decomposition properties can then be aligned with Definition 4 as follows.

Lemma 1. Decomposition Properties. *Tasks $\mathcal{T}_1, \ldots, \mathcal{T}_n$ denoting a decomposition according to Definition 4 fulfill the decomposition properties* independence *and* completeness *regarding $\mathcal{M}$.*

Proof. Assume there would be a task $\mathcal{T}_i$ depending on a previous execution of $\mathcal{T}_j$, then the condition in Definition 4 would be violated and $\mathcal{T}_i$ would not be part of a decomposition. The same holds true when $\mathcal{T}_i$ depends on non-execution of $\mathcal{T}_j$. Similarly, if execution of all tasks $\mathcal{T}_1, \ldots, \mathcal{T}_n$ would not fulfill $\mathcal{M}$, then the sequence $\sigma_1 \ldots \sigma_n$ would not satisfy ϕ as required by Definition 4. ■

In order to construct the decomposition set D, it is possible to find so-called *essential sequences*, basically the sequences in the NFA with minimal propositions, and relate states $q \in Q$ to decomposition choices according to Definition 4.

Definition 5. Essential Sequence (Schillinger et al. (2016a)). *A sequence σ is called essential for an NFA $\mathcal{F}$ if and only if it describes a run ρ in $\mathcal{F}$ and $\sigma(t) \setminus \{\pi\} \not\vDash \delta(\rho(t-1), \rho(t))$ for all t and propositions $\pi \in \sigma(t)$, i.e., σ contains only the required propositions.*

In practice, it is rather intuitive to construct an essential sequence, especially if representing the transition conditions δ in the NFA in disjunctive normal form (DNF). For example considering the NFA in Figure 1, only the self-transition at state q_1 is not in DNF and can be represented equivalently as $cbin \wedge \neg def \vee cbin \wedge \neg disp$. If transition conditions are given in DNF, an essential sequence can be constructed by adding all propositions of one conjunctive clause to the sequence. For example, an essential sequence from q_0 to q_1 would be $\sigma = \{def, desk\}$.

Based on the notion of an essential sequence, the decomposition set D can finally be constructed as shown by the following Theorem.

Theorem 1. Decomposability (Schillinger et al. (2016a)). *Let $q \in Q$ be a state in the NFA $\mathcal{F}$ constructed from $\mathcal{M}$, and $\sigma = \sigma_1\sigma_2$ be an essential sequence such that σ_1 describes a run from an initial state to q and σ_2 describes a run from q to an accepting state of $\mathcal{F}$. Then, $q \in D$ if and only if $\widehat{\sigma} = \sigma_2\sigma_1$ describes an accepting run in $\mathcal{F}$.*

Note that initial and accepting states are trivially contained in the decomposition set. This reflects the decomposition into one part consisting of doing nothing and the other part of doing the whole mission, indicating that the mission is not split. In the following, we denote these trivial states by $Q_{\text{triv}} := Q_0 \cup F$ and say that a mission is *decomposable* only if $D \setminus Q_{\text{triv}} \neq \emptyset$.

Example 4 (Non-Decomposability). First, only assume the part ϕ_1, as defined in Example 2, of the specification ϕ. It covers the part where a bin should be picked up and emptied. Although this task covers first picking up the bin and second, emptying it, these two parts cannot be separated into independent tasks because it is not possible to empty the bin without picking it up before.

In order to check this intuition with Theorem 1 and determine the decomposition set, we need to construct an essential sequence for each of the states in the NFA of ϕ_1 (see Figure 1) and expect that $D \setminus Q_{\text{triv}} = \emptyset$.

q_1: The essential sequence $\sigma = \sigma_1\sigma_2$ is given by $\sigma_1 = \{def, desk\}$ to reach q_1 and $\sigma_2 = \{disp, def\}$ to continue from q_1. The permuted sequence $\sigma_2\sigma_1$ gives the run $q_0q_0q_1$, resulting in $\sigma_2\sigma_1 \not\vDash \phi_1$. Consequently, q_1 is not in D.

This process is repeated for the two remaining states.

q_3: $\sigma = \sigma_1\sigma_2$ with $\sigma_1 = \{def, desk\}\{cbin, def, \neg disp\}$ and $\sigma_2 = \{disp\}$. $\sigma_2\sigma_1 \not\vDash \phi_1$ with run $q_0q_0q_1q_3$.

q_4: $\sigma = \sigma_1\sigma_2$ with $\sigma_1 = \{def, desk\}\{disp, \neg def\}$ and $\sigma_2 = \{def\}$. $\sigma_2\sigma_1 \not\vDash \phi_1$ with run $q_0q_0q_1q_4$.

Consequently, we get that $D \setminus Q_{\text{triv}} = \emptyset$, i.e., the specification ϕ_1 is not decomposable. ◇

In contrast, the whole mission specification ϕ is expected to be decomposable as discussed previously in Example 3. The following example shows a case where it is possible to decompose ϕ.

Example 5 (Decomposability). For space reasons, we skip a visualization of the full NFA of ϕ, but consider the following case for a possible decomposition at a state q. The essential sequence $\sigma = \sigma_1\sigma_2$ with $\sigma_1 = \{def, desk, \neg cbin\}\{\neg cbin, disp\}\{def, \neg cbin\}$ and $\sigma_2 = \{desk, ebin, \neg cbin\}\{def, desk, \neg cbin\}$ describes an accepting run in the NFA and also $\sigma_2\sigma_1 \vDash \phi$. Consequently, a decomposition is possible at the respective state and we get $q \in D$. Illustratively speaking, a decomposition at this specific q resembles the behavior discussed in Example 3 where one robot empties the full bin and another one brings a new one independently. ◇

The way how exactly a goal specification is formulated can influence the decomposition set and consequently, the tasks which are identified to be independent. While we defer a discussion of this aspect to the end of this paper, note here that the decomposition approach presented above operates on the automaton representation of the formula. This relaxes the dependency on the specific formulation as the automaton corresponds more closely to the semantics of the goal.

4 Model Construction

Given an automaton which represents the mission specification and where each state indicates if a split is possible there as discussed in the previous section, we can construct a full model of the system. In addition to the mission, this model captures capabilities of the agents and thus, describes possible action sequences to execute the mission.

4.1 System Model

First, we start with modeling the individual components of the system before we describe how to combine them to a complete model. Since the LTL mission can be represented as an automaton, it appears natural to model an agent as an automaton as well. It is also often intuitive to model the internal state and the actions of an agent as a state machine or an abstraction of places in the environment as a topological map.

Definition 6. Agent Model. *An agent model is given as the automaton* $\mathcal{A} := (S_\mathcal{A}, s_{0,\mathcal{A}}, A_\mathcal{A}, \Pi, \lambda)$ *consisting of*
(1) *a set of states* $S_\mathcal{A}$,
(2) *an initial state* $s_{0,\mathcal{A}} \in S_\mathcal{A}$,
(3) *a set of actions* $A_\mathcal{A} \subseteq S_\mathcal{A} \times S_\mathcal{A}$,
(4) *a set of propositions* Π,
(5) *a labeling function* $\lambda\colon S_\mathcal{A} \to 2^\Pi$.

Then, by combining this agent model automaton with the NFA of the mission, a model can be constructed to capture both the agent capabilities encoded in $\mathcal{A}$ and the mission specification encoded in the NFA $\mathcal{F}$.

Definition 7. Product Model. *A product model is given by* $\mathcal{P} = \mathcal{F} \otimes \mathcal{A} := (S_\mathcal{P}, S_{0,\mathcal{P}}, A_\mathcal{P})$ *consisting of*
(1) *a set of states* $S_\mathcal{P} = Q \times S_\mathcal{A}$,
(2) *a set of initial states* $S_{0,\mathcal{P}} = Q_0 \times \{s_{0,\mathcal{A}}\}$,
(3) *a set of actions* $A_\mathcal{P} = \{((q_\mathrm{s}, s_\mathrm{s}), (q_\mathrm{t}, s_\mathrm{t})) \in S_\mathcal{P} \times S_\mathcal{P}\colon (s_\mathrm{s}, s_\mathrm{t}) \in A_\mathcal{A} \wedge \lambda(s_\mathrm{s}) \vDash \delta(q_\mathrm{s}, q_\mathrm{t})\}$.

The agent models can be different, each representing the capabilities of the respective agent, while the NFA is determined by a particular mission specification. As such, $\mathcal{P}$ describes for each of the different agents how a given mission can be executed by this agent or indicates that the mission cannot be satisfied.

In order to combine multiple agents, a team model can be constructed from the individual product models. While there are different ways of defining this combination, we choose an approach based on the decomposition set discussed in the previous section. Individual agents are assumed to act independently and based on the decomposition set, special transitions indicate the options to split a mission at some states and allocate the rest to a different agent. Consequently, the resulting team model can be used for simultaneous task allocation and planning.

Definition 8. Team Model. *The team model automaton* $\mathcal{G}$ *is a union of the* N *local product models* $\mathcal{P}^{(r)}$ *with* $r \in \{1, ..., N\}$ *and given by* $\mathcal{G} := (S_\mathcal{G}, S_{0,\mathcal{G}}, F_\mathcal{G}, A_\mathcal{G})$ *consisting of*
(1) *a set of states* $S_\mathcal{G} = \{(r, q, s)\colon r \in \{1, ..., N\}, (q, s) \in S_\mathcal{P}^{(r)}\}$,
(2) *a set of initial states* $S_{0,\mathcal{G}} = \{(r, q, s) \in S_\mathcal{G}\colon r = 1\}$
(3) *a set of final states* $F_\mathcal{G} = \{(r, q, s) \in S_\mathcal{G}\colon q \in F\}$,
(4) *a set of actions* $A_\mathcal{G} = \bigcup_r A_\mathcal{P}^{(r)} \cup \zeta$ *which include switch transitions* ζ *as defined below.*

In Definition 8, $\mathcal{P}^{(r)}$ is the product automaton corresponding to agent r according to Definition 7. A switch transition is only present at a state $s \in S_\mathcal{G}$ if $\mathcal{M}$ can be decomposed at s according to the following conditions.

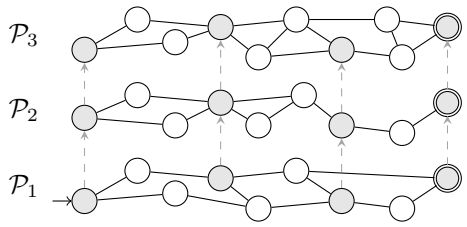

Figure 2. Structure of $\mathcal{G}$ for three agents. It has one initial state (bottom left corner) and three final states (right side). Between the agent automata, directed *switch transitions* to the next agent connect states of the *decomposition set*.

Definition 9. Switch Transition. *The set of switch transitions in* $\mathcal{G}$ *is given by* $\zeta \subset S_\mathcal{G} \times S_\mathcal{G}$. *A transition* $\varsigma = ((r_\mathrm{s}, q_\mathrm{s}, s_\mathrm{s}), (r_\mathrm{t}, q_\mathrm{t}, s_\mathrm{t}))$ *belongs to* ζ *if and only if it*
(i) *connects different agents:* $r_\mathrm{s} \neq r_\mathrm{t}$,
(ii) *preserves the NFA progress:* $q_\mathrm{s} = q_\mathrm{t}$,
(iii) *points to the next agent:* $r_\mathrm{t} = r_\mathrm{s} + 1$,
(iv) *points to an initial agent state:* $s_\mathrm{t} = s_{0,\mathcal{A}}^{(r_\mathrm{t})}$,
(v) *represents a decomposition choice:* $q_\mathrm{s} \in D$.

Note that staying in one state of the NFA as specified in condition (ii) requires that q_s has a self-transition and that the respective condition is fulfilled. An example for the structure of the team model $\mathcal{G}$ is depicted in Figure 2.

Finally, an action sequence $\beta := s_0 a_1 s_1 \dots a_n s_n$ is defined as a run in $\mathcal{G}$ such that $s_i \in S_\mathcal{G}$, and $a_i \in A_\mathcal{G}$. In order to distribute β among the involved agents, $\beta_\mathcal{P}^{(r)}$ for agent r can be formed by projecting β onto $\mathcal{P}^{(r)}$. Note that, due to the characteristics of $\mathcal{G}$, subsequent actions in β either refer to the same agent or are separated by a switch transition $a_i = \varsigma_i \in \zeta$. In the second case, s_{i-1} is the final state of one agent and s_i the initial state of another one. These properties of $\mathcal{G}$ are more closely discussed in the following.

4.2 Model Properties

The team model $\mathcal{G}$ as defined above has a set of relevant properties which are discussed in the following. Specifically, these properties are required to ensure that the planning algorithm, presented in the next section, is guaranteed to find the optimal allocation of the given LTL mission specification and the optimal sequence of actions for each agent to fulfill the specification.

Lemma 2. Correctness. *If there exists an action sequence* $\beta = s_0 a_1 s_1 \dots a_n s_n$ *in* $\mathcal{G}$ *such that* $s_0 \in S_{0,\mathcal{G}}$ *and* $s_n \in F_\mathcal{G}$, *the corresponding mission specification* ϕ *is satisfied by* β.

Proof. Due to the construction of $\mathcal{G}$, every state $s = (r, q, s_\mathcal{A}) \in S_\mathcal{G}$ has a component $q \in Q$ in the NFA $\mathcal{F}$ constructed from ϕ. Besides the switch transitions, which do not change the NFA component, but only model considering a different agent, no new transitions are added to $\mathcal{G}$. Consequently, s_0 is projected onto $q_0 \in Q_0$, s_n onto $q_n \in F$, and any $a_i \in \beta$ is projected onto an existing transition in $\mathcal{F}$. Thus, the projected β forms an accepting run in $\mathcal{F}$ and consequently, satisfies the mission specification ϕ. ■

In the case that Lemma 2 holds for an action sequence β, we call β a satisfying action sequence or, more specifically, a solution to the mission $\mathcal{M}$ which given by the mission specification ϕ. As will be shown in the next Lemma, such a solution can be distributed to different agents.

Lemma 3. Independence. *Assume that the solution $\beta = \beta^{(1)}{}_{\varsigma}\beta^{(2)}$ in $\mathcal{G}$ to a decomposable mission $\phi^{(1)} \wedge \phi^{(2)} := \phi$ is allocated to two different agents. Then, both agents can execute $\beta^{(1)}$, $\beta^{(2)}$ independently from each other.*

Proof. The switch transition $\varsigma = ((r_\mathrm{s}, q_\mathrm{s}, s_\mathrm{s}), (r_\mathrm{t}, q_\mathrm{t}, s_\mathrm{t}))$ can only connect $\beta^{(1)}$ and $\beta^{(2)}$ if the respective NFA state $q_\mathrm{s} = q_\mathrm{t} \in D$ is in the decomposition set D. As shown for the respective construction rules of the decomposition set in Section 3, only states indicating a split into two independent parts are in D and thus, ϕ can be written as $\phi^{(1)} \wedge \phi^{(2)}$. ■

By introducing a cost for actions to express the desirability of a solution, it is possible to show that any preferred solution can be constructed from the model $\mathcal{G}$ resulting from the independence shown above.

Lemma 4. Completeness. *Given any action sequence β_opt with minimal costs c^*, which satisfies an LTL mission ϕ with respect to the system model and where actions $a_i \in \beta_\mathrm{opt}$ are in an arbitrary permutation regarding the agents. Then, there exists a β with ordered actions, i.e. $r_i \leq r_{i+1}, r \in \{1, \dots, N\}$ for r_i referring to one of the N agents executing action a_i, such that its costs $c_\beta = c^*$ are optimal as well.*

Proof. As shown in Lemma 3, different agents can execute their parts independently from each other. Consequently, actions $a_i, a_{i+1} \in \beta_\mathrm{opt}$ with $r_i \neq r_{i+1}$ can be swapped without affecting the cost or correctness. Thus, it is possible to swap actions in β_opt with $r_i > r_{i+1}$ until the condition $r_i \leq r_{i+1}$ is satisfied for all actions. ■

Furthermore, the team model has additional properties which can be utilized for more efficient planning. In the following, we discuss the most important one in preparation for the planning algorithm presented in the next section.

Lemma 5. Ordered Sequence. *For any action sequence β with actions $a_i, a_{i+1} \in \beta_\mathrm{opt}$ resulting from the team model $\mathcal{G}$, it holds that $r_i \leq r_{i+1}, r \in \{1, \dots, N\}$ for r_i referring to one of the N agents executing action a_i.*

Proof. According to condition (iii) in Definition 9, it holds that $r_\mathrm{s} < r_\mathrm{t}$ for all switch transitions $\varsigma = ((r_\mathrm{s}, q_\mathrm{s}, s_\mathrm{s}), (r_\mathrm{t}, q_\mathrm{t}, s_\mathrm{t})) \in \zeta$. Similarly, according to the construction of $\mathcal{G}$ in Definition 8, all actions $a \in \bigcup_r A_\mathcal{P}^{(r)}$ are associated with a single agent r. Consequently, $r_i \leq r_{i+1}$ holds for all transitions $A_\mathcal{G} = \bigcup_r A_\mathcal{P}^{(r)} \cup \zeta$. ■

Specifically, note that an immediate consequence of Lemma 5 is that, starting in a state associated with agent r, no state associated with an agent $r' < r$ can be reached by any path in $\mathcal{G}$.

4.3 Cost Representation

In order to select the optimal satisfying action sequence, actions $A_\mathcal{G}$ are associated with non-negative costs $C\colon A_\mathcal{G} \to \mathbb{R}_{\geq 0}$. A special case are the switch transitions $\zeta \subset A_\mathcal{G}$ and we define $C\colon \varsigma \mapsto 0$ for all $\varsigma \in \zeta$. This reflects the fact that switch transitions are purely virtual and will not appear in any of the action sequences $\beta_\mathcal{P}^{(r)}$ executed by the agents.

For modeling the multi-agent character of a cost, one can think of extending $C(a)$ for an action $a \in A_\mathcal{G}$ to an N dimensional vector $c_a \in \mathbb{R}_{\geq 0}^N$ where each agent $r \in \{1, \dots, N\}$ represents one dimension. Since each action with non-zero cost is associated with a particular agent by the fact that $A_\mathcal{G} \setminus \zeta = \bigcup_r A_\mathcal{P}^{(r)}$, we can define for each $a \in A_\mathcal{G}$

$$c_{a,i} = \begin{cases} C(a) & \text{if } i = r \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

and $c_\zeta = 0$. Consequently, the costs $c_\beta \in \mathbb{R}_{\geq 0}^N$ of an action sequence β are given by $c_\beta = \sum_{a \in \beta} c_a$.

The Pareto front of all cost vectors c_β for satisfying action sequences then forms the set of potential optimal solutions. In order to prioritize these solutions, we define the overall team cost $\kappa\colon \mathbb{R}_{\geq 0}^N \to \mathbb{R}_{\geq 0}$ as

$$\kappa(c_\beta) = (1 - \epsilon) \cdot \|c_\beta\|_\infty + \epsilon \cdot \|c_\beta\|_1. \tag{5}$$

with $\epsilon \in (0, 1]$. This reflects the objective to minimize the maximal agent cost $\|c_\beta\|_\infty$, e.g., minimizing the completion time of the mission, with a regularization term $\|c_\beta\|_1$ to avoid unnecessary actions of the agents.

However, based on Lemma 5 and given the definition of κ in Equation (5), it is possible to choose a more compact representation for c_β, in the following denoted by $\widehat{c}_\beta$. Specifically, we utilize the fact that given a particular agent r, the team cost κ of the action sequence β can already be evaluated for all agents $r' < r$ since no action associated with any r' will occur in a continuation of β. Consequently, it is sufficient to define $\widehat{c}_\beta \in \mathbb{R}_{\geq 0}^3$ irrespectively of the team size N as a three-dimensional cost vector such that

$$\widehat{c}_\beta = \begin{pmatrix} \|(c_{\beta,1}, \dots, c_{\beta,r-1})^T\|_\infty \\ \|(c_{\beta,1}, \dots, c_{\beta,r-1})^T\|_1 \\ c_{\beta,r} \end{pmatrix}. \tag{6}$$

Accordingly, the team cost $\widehat{\kappa}\colon \mathbb{R}_{\geq 0}^3 \to \mathbb{R}_{\geq 0}$ is adjusted such that

$$\begin{aligned} \widehat{\kappa}(\widehat{c}_\beta) =& (1 - \epsilon) \cdot \|(\widehat{c}_{\beta,1}, \widehat{c}_{\beta,3})^T\|_\infty \\ &+ \epsilon \cdot \|(\widehat{c}_{\beta,2}, \widehat{c}_{\beta,3})^T\|_1 \end{aligned} \tag{7}$$

with $\widehat{c}_{\beta,i}$ denoting the i-th component of $\widehat{c}_\beta$. Not only does this compact representation remove a dependency of c_β on the team size N, it is also more efficient to use this representation during planning. The reason for this efficiency gain is that additional cost vectors are Pareto-dominated and can thus be eliminated from the set of potential solutions much earlier in the planning process.

4.4 Resource Representation

In addition to the LTL mission specification, which models discrete constraints, it is often required to also consider constraints of the agents in continuous domains, in the following referred to as resource constraints. As resources, we capture numerical domains which are not explicitly modeled in the discrete state space $S_\mathcal{G}$ of the system, but might change as the consequence of executing an action $a \in A_\mathcal{G}$. This change is denoted by $\Gamma\colon A_\mathcal{G} \to \mathbb{R}^M$ where M is the total amount of resource dimensions. In contrast to costs, which can only be increased, resources can be modified in both directions.

For an action sequence β, the resulting status of resources $\gamma_\beta \in \mathbb{R}^M$ is given by

$$\gamma_\beta = \gamma_0 + \sum_{a \in \beta} \Gamma(a) \tag{8}$$

where $\gamma_0 \in \mathbb{R}^M_{\geq 0}$ denotes the initial resources. We constrain the set of satisfying action sequences to sequences β such that at any state $s_i \in \beta$ it holds for $\beta' = s_0 a_1 \dots a_i s_i$ that $\gamma_{\beta',i} > 0$ for each component $i \in \{1, \dots, M\}$.

Example 6 (Limited Battery). The assumption that the robots only have a limited battery capacity can be modeled as one resource constraint per robot. $\gamma_{\beta,r}$ provides a measure of how much battery power is left for robot r and consequently, $\gamma_{\beta,r} > 0$ guarantees that the robots do not get depleted at any time during the mission. Each robot has an initial battery level $\gamma_{0,r}$ and each action a affects the battery level by $\Gamma(a)$. In particular, we set $\Gamma(a) < 0$ for all actions which consume battery and $\Gamma(a) > 0$ for charging actions. ◇

Note that modeling agent-specific resources as described in the above example, while intuitive, again implies a dependency of the resource vector γ_β on the team size $N = M$. Similar to changing the cost representation, we can also model agent-specific resources in a more compact way given Lemma 5. More precisely, it suffices to have a single dimension $\gamma_{\beta,i}$ of γ_β tracking the resource status. Whenever a switch transitions $\varsigma = ((r_\mathrm{s}, q_\mathrm{s}, s_\mathrm{s}), (r_\mathrm{t}, q_\mathrm{t}, s_\mathrm{t}))$ is taken, $\gamma_{\beta,i}$ is set to the initial resource status $\gamma_{0,i}$ of agent r_t to which ς leads.

As mentioned before, we assume that resource constraints are of the form $\gamma_{\beta,i} > 0$. In the rest of this section, we present several other types of constraints which can be reformulated as $\gamma_{\beta,i} > 0$ as well.

First, for completeness, recall the trivial relationships

$$\gamma_{\beta,i} > c \Leftrightarrow \gamma_{\beta,i} - c > 0$$
$$\gamma_{\beta,i} < 0 \Leftrightarrow -\gamma_{\beta,i} > 0.$$

Also constraints of the form $\gamma_{\beta,i} \geq 0$ can be expressed. To include the case of equality, one can add a small offset ϵ to the resources and formulate it as strict inequality constraint. A sufficiently small ϵ is derived as follows. Given by the finiteness of the set of actions, there is a smallest possible change of $\gamma_{\beta,i}$ for each resource, given by

$$\gamma_{\Delta,i} = \min_{(a_j, a_k)} |\Gamma(a_j)_i - \Gamma(a_k)_i| \tag{9}$$

for actions $a_j, a_k \in A_\mathcal{G}$ and resource modifiers Γ. Thus, it suffices that $\epsilon \leq \gamma_{\Delta,i}$ and the constraint $\gamma_{\beta,i} \geq 0$ is implied by $\gamma_{\beta,i} + \epsilon > 0$.

Interval constraints of the form $\gamma_{\beta,i} \in \mathcal{I} = (c, C)$ can be captured by the set of two inequality constraints $\gamma_{\beta,i} > c$ and $\gamma_{\beta,i} < C$. However, this would be inefficient in practice as it introduces a potentially large set of Pareto optimal labels. Instead, it is usually appropriate to prefer solutions further away from the interval boundaries, given equally high costs, as they are more likely to satisfy the interval constraint in the future. Thus, interval constraints can be expressed as

$$\gamma_{\beta,i} \in \mathcal{I} \Leftrightarrow \gamma_{\beta,\mathcal{I}} - \frac{C - c}{2} > 0$$

where $\gamma_{\beta,\mathcal{I}} = \left\| \frac{C-c}{2} + c - \gamma_{\beta,i} \right\|$ denotes a distance measure of $\gamma_{\beta,i}$ from the center of the interval $\mathcal{I}$.

5 Action Planning

In this section, we present an approach for *Simultaneous Task Allocation and Planning (STAP)* for the previously defined model. Specifically, the goal is to solve the following planning problem.

Problem 1. Simultaneous Task Allocation and Planning. *Let the models $\mathcal{A}^{(r)}$ with $r \in R$ of a multi-robot system $R = \{1, \dots, N\}$, a cost function C, initial resources $\gamma_0 \in \mathbb{R}^M_{\geq 0}$, and a goal specification ϕ to be achieved by the team R. Find an action sequence $\beta^{(r)}$ for all $r \in R$ such that ϕ is satisfied and the team cost $\widehat{\kappa}$ is minimized.*

We represent ϕ as an LTL formula and construct the team model $\mathcal{G}$ as defined in Section 4.1. In particular, the underlying decomposition set D, which captures the distributable tasks, is automatically determined as discussed in Section 3.2. The team cost $\widehat{\kappa}$ is defined according to the compact representation in Equation (7). It remains to find an action sequence β in $\mathcal{G}$ which can be decomposed into $\beta^{(r)}$ and distributed to all $r \in R$. Then, each robot r can execute its respective $\beta^{(r)}$ independently of the others, as given by Lemma 3.

The proposed planning algorithm is an extension of the approach described in our previous work in Schillinger et al. (2017), which is a label-setting approach based on the Martins' algorithm in Martins (1984). We extend our previous work by explicitly considering the more efficient cost representation as discussed in Section 4.3, i.e., we integrate our particular formulation of the team model $\mathcal{G}$, and extend the expressiveness of resource constraints as presented in the following Section.

A label-setting search algorithm can be thought of as a multi-criteria generalization of the Dijkstra shortest path search. But instead of operating on states with associated costs, a label-setting algorithm constructs a set of labels for each state. Each label of a state $s \in S_\mathcal{G}$ is given by

$$l = (\widehat{c}_\beta, \gamma_\beta, v, i_v) \tag{10}$$

and depends on the action sequence β leading to it where $\widehat{c}_\beta$ denotes its cost as defined in Equation (6) and γ_β the respective resource status as defined in Equation (8). Integration of γ_β into the labels is an extension of the standard label-setting approach, but required to handle resource constraints. $v \in S_\mathcal{G}$ is the predecessor state in β and $i_v \in \mathbb{N}$ the index of the respective predecessor label.

The construction of a set of labels for each state $s \in S_\mathcal{G}$ is basically an extension of $S_\mathcal{G}$ to a higher dimensional, infinitely large label space $\mathcal{L}_\mathcal{G}$. Each label $l \in L_{\mathcal{G},s} \subset \mathcal{L}_\mathcal{G}$ of state s instantiates one possible resource configuration in the continuous domain of γ and transitions between the labels are described by their predecessor relations. In the following, we denote by $L_{\mathcal{G},s}$ the set of instantiated, i.e., feasible, labels at state s and by $L_\mathcal{G} = \bigcup_{s \in S_\mathcal{G}} L_{\mathcal{G},s} \subset \mathcal{L}_\mathcal{G}$ the set of all feasible labels.

5.1 Resource Propositions

Considering the extension of the state space $S_\mathcal{G}$ to the label space $L_\mathcal{G}$ opens a more expressive perspective on the resource constraints defined in Section 4.4. In fact, it is possible to model a resource constraint as a proposition

$$\pi_i := \gamma_{\beta,i} > 0. \tag{11}$$

In the state space $S_\mathcal{G}$, the constraint $\gamma_{\beta,i} > 0$ depends on the complete action sequence β, which specifies the

resource modifications following from the included actions. In contrast, this constraint only depends on a single label l in the label space $L_\mathcal{G}$ with a specific $\gamma_{l,i} = \gamma_{\beta,i}$. Thus, the resource proposition π_i is either true or false for a certain element of the space $L_\mathcal{G}$.

Following this idea, the resource constraints can directly be included in the LTL mission specification by the extended formulation $\phi \wedge \square\,(\bigwedge_i \pi_i)$, stating that all constraints always have to hold. However, based on conventional LTL semantics as summarized in Section 2.1 now interpreted over the label space $L_\mathcal{G}$, resource propositions π_i can be used as part of the mission specification ϕ the same way as atomic propositions $\pi \in \Pi$. In this way, complex resource constraints can be expressed. Some particularly useful examples are $\lozenge\square\pi_i$ ("at some point, reach and keep a certain threshold for resource i"), $\neg\pi_i \implies \lozenge\phi_j$ ("violation of resource constraint i requires to execute ϕ_j as compensation"), or $\pi_i \,\mathcal{U}\, \phi_j$ ("respect resource constraint i until satisfaction of ϕ_j").

Remark. In the latter case of the above examples, $\pi_i := t_i - \gamma_{l,i} > 0$ could represent a resource like time. Then, the expression can be interpreted as "satisfy ϕ_j before time t_i". This highlights the close connection of interpreting LTL over a more complex space like $L_\mathcal{G}$ with existing temporal logics like MITL, see Alur et al. (1996), and STL, see Maler and Nickovic (2004), over a state space $S_\mathcal{G}$. However, a more extensive comparison with these logics is out of the scope of this paper and will be subject to future work.

Until now, only agent-specific resources have been considered. However, for some scenarios, it is helpful to also model agent-independent, *global* resource constraints. Consider for example the scenario that some supplies need to be refilled. The most straight-forward way would be to specify $\phi = \lozenge\pi_i$ with $\pi_i := \gamma_{\beta,i} - c_\text{thresh} \geq 0$ for some resource threshold c_thresh. In this case, $\gamma_{\beta,i}$ is a global resource.

Since, by construction of the team model $\mathcal{G}$, we assume the agents to act independently, it is not immediately clear how to consider global resources. Nevertheless, it is possible to model global resources in the following special cases and we limit usage of global resources to these cases.

Lemma 6. Global Resources. *Consider the global resource $\gamma_{\beta,i}$ and assume that all of its modifications $\Gamma(a)_i$ for $a \in A_\mathcal{G}$ are non-positive. Then, the LTL specifications $\lozenge\neg\pi_i$ and $\square\pi_i$, including their Boolean combinations, can be expressed for $\pi_i := \gamma_{\beta,i} > 0$.*

Proof. For monotonically decreasing resources $\gamma_{\beta,i}$, it is apparent that the proposition $\gamma_{\beta,i} > 0$ either does not change its truth value, or there exists exactly one transition from true to false. Consequently, $\square\pi_i$ is fulfilled in the former case, $\lozenge\neg\pi_i$ in the latter case. In the case that there is a transition, this transition exists independently of the order in which the resource modifications $\Gamma(a)_i$ are applied due to monotonicity. ■

5.2 Planning Algorithm

An outline of our proposed approach for constrained optimal STAP is given by Algorithm 1. The purpose of the algorithm is to find the label l_fin of an accepting state with minimal team cost $\widehat{\kappa}(\widehat{c}^{(l)})$. Since each label l specifies its predecessor label, the action sequence β leading to l can be reconstructed

Algorithm 1 Constrained Optimal STAP

Input: Team model $\mathcal{G}$, team cost function $\widehat{\kappa}$, resources γ_0
Output: Optimal final label l_fin to construct actions β_fin

Notation Remarks:
$l = (\widehat{c}^{(l)}, \gamma^{(l)}, v, i_v) \in L_\mathcal{G}$ – label, see Equation (10)
$L_{\text{t},s}$ ($L_{\text{p},s}$) – set of temporary (permanent) labels, $s \in S_\mathcal{G}$
$l <_\text{P} \ell$ – short notation for $(\widehat{c}^{(l)}, -\gamma^{(l)}) <_\text{P} (\widehat{c}^{(\ell)}, -\gamma^{(\ell)})$
$\Delta(a, \gamma)$ – indicates if $\gamma > 0$ holds for a, see Equation (14)

1: $L_{\text{t},v} \leftarrow \{(0, \gamma_0, \varnothing, \varnothing)\}, \forall v \in S_{0,\mathcal{G}}$
2: $L_{\text{t},s} \leftarrow \emptyset, \forall s \in S_\mathcal{G} \setminus S_{0,\mathcal{G}}$
3: $L_{\text{p},s} \leftarrow \emptyset, \forall s \in S_\mathcal{G}$
4: **while** $\forall s \in S_\mathcal{G} : L_{\text{t},s} \neq \emptyset$ **do**
▶ Find label l with lowest costs and make it permanent
5: $(s, l) \leftarrow \text{argmin}_{s \in S_\mathcal{G}, l \in L_{\text{t},s}} \{\widehat{\kappa}(\widehat{c}^{(l)})\}$
6: $L_{\text{t},s} \leftarrow L_{\text{t},s} \setminus \{l\}$
7: $L_{\text{p},s} \leftarrow L_{\text{p},s} \cup \{l\}$
▶ Terminate search if any final state is reached
8: **if** $s \in F_\mathcal{G}$ **then return** $l_\text{fin} \leftarrow l$ ▷ Best found first
▶ Calculate labels for all successors v of s
9: **for all** $v \in S_\mathcal{G} : a = (s, v) \in A_\mathcal{G}$ **do**
10: $\widehat{c}_\text{new} \leftarrow \text{updateCost}(a, \widehat{c}^{(l)})$ ▷ Equation (12)
11: $\gamma_\text{new} \leftarrow \text{updateRess}(a, \gamma^{(l)})$ ▷ Equation (13)
12: $\ell \leftarrow (\widehat{c}_\text{new}, \gamma_\text{new}, s, i_s)$ with $i_s = \text{card}(L_{\text{p},s})$
▶ Only add and keep non-dominated temporary labels
13: **if** $\Delta(a, \gamma_\text{new}) \wedge (\nexists l \in L_{\text{t},v} \cup L_{\text{p},v} : l \leq_\text{P} \ell)$ **then**
14: $L_{\text{t},v} \leftarrow L_{\text{t},v} \setminus \{l \in L_{\text{t},v} : \ell <_\text{P} l\}$
15: $L_{\text{t},v} \leftarrow L_{\text{t},v} \cup \{\ell\}$
16: **return** $l_\text{fin} \leftarrow \varnothing$ ▷ No final state reachable

and l denotes the costs and resource status when executing β. Consequently, finding l_fin is equivalent to finding the respective action sequence β_fin which leads to an accepting state and thus, satisfies the given mission.

In order to find l_fin, a reachable set of temporary labels $L_{\text{t},s}$ is constructed for each state and a set of permanent labels $L_{\text{p},s}$ denotes Pareto-optimal labels at s. As usual for a label-setting approach, only optimal labels are made permanent and consequently, the temporary label with minimal team cost is selected in each iteration (line 5). This requires non-negative action costs in order to guarantee that no label with lower team cost will be added to the set of temporary labels in later iterations.

For each selected label l, a set of consecutive labels is constructed by extending the action sequence associated with l by all available actions (line 9) and adding the resulting labels to the reachable set (line 15). To improve efficiency, only actions resulting in Pareto-optimal labels at their target state are added. Since these labels are only in the reachable set instead of the permanent set, it can be the case that a better label is found later during planning before the label is made permanent. Consequently, when adding a new temporary label, existing temporary labels are again checked for Pareto-optimality (line 14).

Depending on whether a certain action $a = ((r_\text{s}, q_\text{s}, s_\text{s}), (r_\text{t}, q_\text{t}, s_\text{t})) \in A_\mathcal{G}$ is a switch transition, the costs of the label $\widehat{c}_\text{new}$ after action execution are updated

based on the costs $\widehat{c}^{(l)}$ of the current label l.

$$\widehat{c}_{\text{new}} := \begin{cases} \begin{pmatrix} \|(\widehat{c}_1^{(l)}, \widehat{c}_3^{(l)})^T\|_\infty \\ \|(\widehat{c}_2^{(l)}, \widehat{c}_3^{(l)})^T\|_1 \\ 0 \end{pmatrix} & \text{if } a \in \zeta \\ \widehat{c}^{(l)} + (0, 0, C(a))^T & \text{otherwise} \end{cases} \tag{12}$$

In the case $a \in \zeta$, i.e., that a is a switch transition, the maximal and summed costs are updated and the new agent r_t is initialized with zero cost by setting the third component $\widehat{c}_{\text{new},3} = 0$.

Similarly, resources γ_new are updated as follows based on $\gamma^{(l)}$.

$$\gamma_{\text{new}} := \begin{cases} \begin{pmatrix} \gamma_{\text{global}}^{(l)} \\ \gamma_{0,r_\text{t}} \end{pmatrix} & \text{if } a \in \zeta \\ \gamma^{(l)} + \Gamma(a) & \text{otherwise} \end{cases} \tag{13}$$

where $\gamma_{\text{global}}^{(l)}$ is the part of $\gamma^{(l)}$ denoting global, i.e., agent-independent, resources and γ_{0,r_t} are the initial resources of agent r_t, the agent associated with the target state of a.

Furthermore, selecting the available actions requires to ensure that the new resource status satisfies all constraints, which can be encoded in the LTL mission specification as discussed above. For this purpose, we define $\Delta\colon A_\mathcal{G} \times \mathbb{R}^M \to \mathbb{B}$ with $\mathbb{B} := \{\top, \bot\}$ as an extension of the transition function δ of an NFA $\mathcal{F}$ such that

$$\Delta\colon (a, \gamma) \mapsto \lambda(s_\text{s}) \cup \Pi_\gamma \vDash \delta(q_\text{s}, q_\text{t}) \tag{14}$$

for $a = ((r_\text{s}, q_\text{s}, s_\text{s}), (r_\text{t}, q_\text{t}, s_\text{t}))$ and Π_γ is the set of all propositions π_i according to Equation (11). Illustratively speaking, Δ extends the set of state propositions as defined by the proposition labeling function λ of the agent model $\mathcal{A}$ by the set of all resource propositions based on γ and only permits a transition if this extended set fulfills the transition condition δ of $\mathcal{F}$.

In addition to not considering actions which would violate the resource constraints, only actions leading to labels with minimal cost will be considered as continuations of an action sequence. The operator $<_\text{P}$ denotes a "less than"-relation in the Pareto sense, i.e., $(a_1, \ldots, a_n)^T <_\text{P} (b_1, \ldots, b_n)^T \Leftrightarrow a \neq b \wedge a_i \leq b_i, \forall i \in \{1, \ldots, n\}$. The operator $\leq_\text{P}$ relaxes this relation and also includes equality $a = b$. Consequently, a label ℓ is only added to the set of temporary labels if it is *non-dominated* (line 13), meaning that there does not exist another label l at the same state such that $(\widehat{c}^{(l)}, -\gamma^{(l)}) \leq_\text{P} (\widehat{c}^{(\ell)}, -\gamma^{(\ell)})$.

5.3 Planning Properties

In the following, we will more closely investigate the required assumptions on the model $\mathcal{G}$ and the costs C for Algorithm 1 to terminate.

First, since Algorithm 1 is a label-setting approach, it is necessary that the temporary labels $l \in L_\text{t} := \bigcup_{sin S_\mathcal{G}} L_{\text{t},s}$ selected in line 5 are traversed in the correct, i.e., non-decreasing, order.

Lemma 7. Label Ordering. *Given any two labels $l, \ell \in L_\mathcal{G}$ such that $\widehat{\kappa}(\widehat{c}^{(l)}) < \widehat{\kappa}(\widehat{c}^{(\ell)})$. Then, $\ell \in L_\text{p} \implies l \in L_\text{p}$ with the set of permanent labels L_p defined as $L_\text{p} := \bigcup_{s \in S_\mathcal{G}} L_{\text{p},s}$.*

We briefly sketch the proof of the Lemma 7 here and refer the interested reader to Schillinger et al. (2017), Lemmas 1 and 5, where formal proofs of the required properties are provided.

Proof. (sketch) It can be shown that the above ordering holds if $\widehat{\kappa}$ fulfills the following two properties: *dominance*, $\widehat{c}^{(l)} <_\text{P} \widehat{c}^{(\ell)} \implies \widehat{\kappa}(\widehat{c}^{(l)}) < \widehat{\kappa}(\widehat{c}^{(\ell)})$, and *monotonicity*, $\widehat{\kappa}(\widehat{c}^{(l)}) \leq \widehat{\kappa}(\widehat{c}^{(\ell)})$. κ, as defined here in Equation (5), fulfills these properties. Finally, it can be shown that $\kappa(c) = \widehat{\kappa}(\widehat{c})$ holds for the team model $\mathcal{G}$. ■

As a consequence of Lemma 7, note that the iterations of the while-loop (line 4) in Algorithm 1 can be seen as increasing a cost threshold, which provides at any time a lower bound on the optimal feasible value $\kappa^* := \widehat{\kappa}(\widehat{c}^{(l^*_\text{fin})})$. This interpretation is expressed by the function $\kappa_\text{LB}\colon \mathbb{N} \to \mathbb{R}$ where $\kappa_\text{LB}(i) := \min_{s \in S_\mathcal{G}, l \in L_{\text{t},s}} \{\widehat{\kappa}(\widehat{c}^{(l)})\}$ denotes the lower cost bound after i iterations (c.f. line 5). Then, Lemma 7 states that κ_LB is non-decreasing.

In order to investigate which assumptions are required to ensure that Algorithm 1 terminates, first assume that an optimal solution β^*_fin with cost κ^* exists, i.e., Problem 1 is solvable. It remains to show that there exists an i^* such that $\kappa_\text{LB}(i^*) = \kappa^*$.

Lemma 8. Zero-Cost Cycles. *Assume there exists a β^*_fin with cost κ^*. Then, $\kappa_\text{LB}(i) < \kappa^*$ for all i only if the planning model $\mathcal{G}$ contains cycles with zero cost.*

Proof. Assume that β^*_fin is not found, i.e., there exists an i such that $\kappa_\text{LB}(j) \leq \kappa_\text{LB}(i) < \kappa^*$ for all $j > i$. Since κ_LB is non-decreasing by Lemma 7, equality $\kappa_\text{LB}(j) = \kappa_\text{LB}(i)$ needs to hold. This means, after i iterations, all actions must have zero costs. Since $j \in \mathbb{N}$ is not bounded above, a cycle formed by actions which all have zero costs needs to exist. ■

Note that the existence of zero-cost cycles is only a necessary, but not a sufficient condition for non-termination. In fact, if a zero-cost cycle is only reachable with a cost $\widehat{\kappa} > \kappa^*$, Algorithm 1 will not reach this cycle before termination. Furthermore, as given in line 13 of Algorithm 1, a zero-cost cycle is only considered by the planner if it increases resources such that the label after traversing the cycle is non-dominated.

Usually, it is not known in advance if a solution β^*_fin exists. Considering now the case that no solution exists for Problem 1, we get from line 4 in Algorithm 1 that the set of temporary labels L_t as defined above needs to become empty in order to terminate the algorithm.

Lemma 9. Bounded Resources. *Assume there exists an upper bound $\gamma_\text{max} \in \mathbb{R}^M_{\geq 0}$ on the resources γ such that $\gamma_i \in [0, \gamma_{\text{max},i}]$ for all resources $i \in \{1, \ldots, M\}$. Then, the set of feasible labels $L_\mathcal{G} \subset \mathcal{L}_\mathcal{G}$ is finite.*

Proof. First, recall that $L_\mathcal{G}$ is defined as $L_\mathcal{G} = \bigcup_{s \in S_\mathcal{G}} L_{\mathcal{G},s}$ where the set of states $S_\mathcal{G}$ is finite. As shown above in Equation (9), given by the finite set of actions $A_\mathcal{G}$ and consequently, the finite amount of possible resource modifiers Γ, there is a minimal resource difference $\gamma_{\Delta,i}$ for each resource dimension $i \in \{1, \ldots, M\}$ with which a state can be reached. Consequently, given an upper bound γ_max on resources, the cardinality of $L_\mathcal{G}$ is bounded by $|L_\mathcal{G}| \leq |S_\mathcal{G}| \prod_{i=1}^{M} \frac{\gamma_{\text{max},i}}{\gamma_{\Delta,i}}$ and thus, $L_\mathcal{G}$ is finite. ■

Bounding the resources is only required in the case that no solution exists since otherwise, the solution will eventually be found. However, requiring such an upper bound, which is usually not a problem in practice, also has implications in the case that a solution exists. As discussed before, a zero-cost cycle is only considered by the planner if it improves resources. Now, given by Lemma 9, resources can only be improved a finite amount of times. Thus, $\kappa_{\mathrm{LB}}(i)$ will always converge to κ^* if the resources are bounded and κ^*, denoting the lowest cost of any feasible solution, exists.

The above results are summarized in the following proposition.

Proposition 1. *Algorithm 1 solves Problem 1 in the sense that it provides the optimal solution β^*_{fin} if one exists and terminates with an empty result otherwise under the assumption that*
(1) *all action costs are non-negative: $C(a) > 0, \forall a \in A_{\mathcal{G}}$,*
(2) *there exists an upper bound $\gamma_{\max} \in \mathbb{R}^M_{\geq 0}$ on resources: $\gamma_i \in [0, \gamma_{\max,i}], \forall i \in \{1, \dots, M\}$.*

5.4 Planning Complexity

As shown in the proof of Lemma 9, an upper bound on cardinality of the label space $L_{\mathcal{G}}$ is given by

$$|L_{\mathcal{G}}| \leq |S_{\mathcal{G}}| \prod_{i=1}^{M} \frac{\gamma_{\max,i}}{\gamma_{\Delta,i}} \tag{15}$$

with $|S_{\mathcal{G}}| = \sum_{r=1}^{N} |S^{(r)}_{\mathcal{P}}| = |Q| \sum_{r=1}^{N} |S^{(r)}_{\mathcal{A}}|$ where N is the number of agents, Q the state space of the NFA and $S^{(r)}_{\mathcal{A}}$ the state space of the respective agent model of agent r. This is only an upper bound on $|L_{\mathcal{G}}|$ since not all theoretically possible labels are necessarily reachable depending on the available actions.

The amount of actions in the team model $\mathcal{G}$ including switch transitions is given by

$$|A_{\mathcal{G}}| = |\zeta| + \sum_{r=1}^{N} |A^{(r)}_{\mathcal{P}}| \tag{16}$$

with $|\zeta| = (N-1) \cdot |D|$ since for each state in the decomposition set D, one switch transition to the initial state of the next agent is constructed. The set of actions $A^{(r)}_{\mathcal{P}}$ of each agent r is bounded by $|A^{(r)}_{\mathcal{P}}| \leq |Q|^2 \cdot |A^{(r)}_{\mathcal{A}}|$. The factor $|Q|^2$ results from the fact that each action in the mission-independent agent model $\mathcal{A}$ can, in theory, connect any two states of the mission NFA. However, this is only an upper bound since numerous of the theoretically possible transitions need not to be considered, as they are constantly false or would violate the constraints.

6 System Implementation

In the following, we explain a ROS implementation of our presented STAP approach. The framework accepts a mission specification as LTL formula and automates all required steps to fulfill the given specification. This includes not only mission decomposition, model construction, and action planning, but also autonomous execution of the generated actions by all involved robots.

6.1 Model Definition

Before being able to accept an LTL formula, the software framework requires a model of the system for which a plan should be derived. As opposed to a specific LTL mission, such a model definition is specified once before deploying the robots and then remains unchanged. The current state of the robots in this model and their available resources are determined online by the system and are updated appropriately.

In our implementation, the required model definitions are given as text files in the YAML[1] format and are manually defined, for example by using available tools like a map editor. The following models are required for a specific system before being deployed.

Topological map – Associates regions and points of interest (together referred to as *nodes*) with location information and discretizes the environment. Each node is labeled with a set of propositions which hold true there. For example, desks are labeled to identify to whom they belong. Furthermore, each node defines a list of edges to indicate possible navigation actions to neighboring nodes.

Robot models – Similar to the topological map, a transition system models the robot's capabilities such that transitions from a state describe possible actions. For example, a bin can be picked up in the *default* state of the robot and results in changing the robot state to *carrying*.

Resource definitions – Defines which resource variables γ_0 are available, for example robot-specific resources such as their battery level or robot-independent resources such as paper supplies in the printer room or coffee in the kitchen. In addition, technical properties such as a dependency on action costs can be specified. For example, it can be defined that the battery level decreases depending on the costs of an action.

Both the topological map, which can be different for each robot, and one of the robot models form the agent model $\mathcal{A}$ of the respective robot as given in Definition 6 and provide actions costs as required for Problem 1. $\mathcal{A}$ is given by the product of the two transition systems with the additional constraints that each action can specify conditions for being applicable. For example, a charging action can be restricted to locations labeled as charging station.

Together with an LTL formula as mission specification, of which many different ones can be received after the system has been deployed, the provided model description forms the required input to Problem 1.

6.2 Software Framework

The approach described in this paper has been implemented in ROS for both evaluating it in simulation and running it on a real multi-robot system. Robots register themselves at the central synthesis server when available for executing missions and provide their agent model based on the above mentioned YAML files.

The synthesis server listens to a ROS topic for new mission requests in the form of a string message containing the LTL formula. When a new mission specification is received, the server plans for the currently available team and distributes the resulting action sequences to all involved robots. We implemented the ROS system as a multi-master setup where each robot runs on its own ROS network and

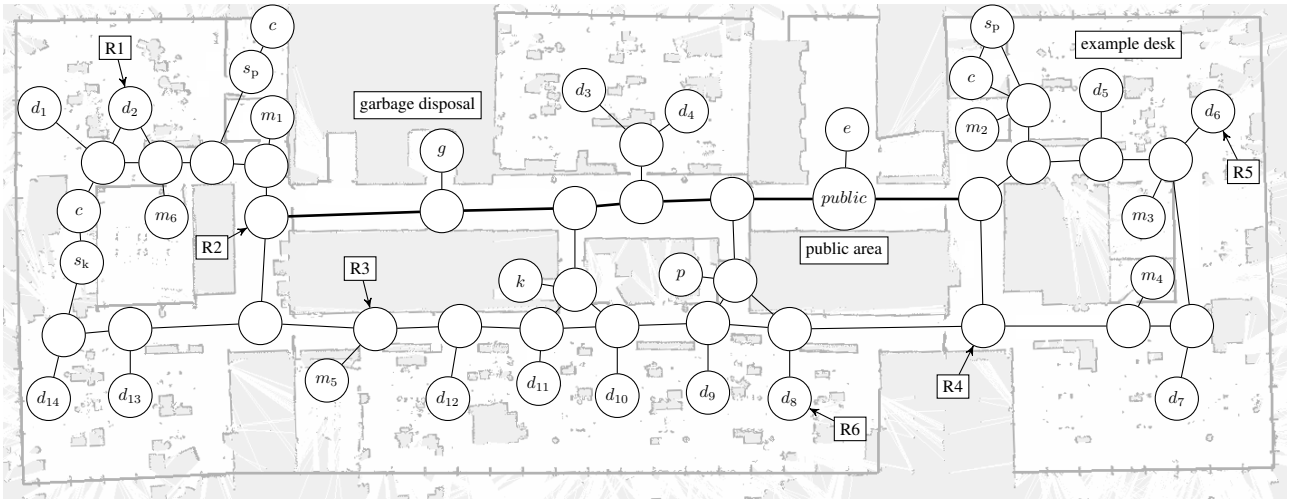

Figure 3. Topological map of the environment to represent navigation actions, while nodes are labeled with atomic propositions. Six robots R1,...,R6 are available in the marked locations. The background shows the map of an existing Bosch office environment, recorded with a usual SLAM procedure. Annotations refer to the case study scenario.

only communicates with the synthesis server to notify it about availability and to receive its generated action plan.

For translating the given LTL formula into an automaton, we use the tool *Spot*[2] Duret-Lutz et al. (2016). To integrate it with the planning framework, we added a ROS action interface which accepts an LTL formula string as input and returns a description of the automaton in the standard HOA[3] format. Consequently, also other LTL to NFA translators can be used if they offer such an interface.

In order to execute the action sequence found by the planner, we use the behavior framework *FlexBE*[4] Schillinger et al. (2016b). FlexBE's graphical editor can be used to define the specific robot capabilities, such as navigation to a waypoint or picking up a bin. These capabilities, including their parameterizations, are then annotated to transitions in the topological map and the robot model in order to provide the implementation for these abstract actions. Since also the FlexBE editor itself provides an interface for behavior synthesis, we provide an offline synthesis option to integrate with the editor and automate development of more complex behaviors, which can in turn be manually adjusted and used itself as capabilities.

7 Evaluation

We focus the evaluation on investigating different aspects of our proposed STAP approach in order to better conclude on its planning properties. After defining the system we use throughout this evaluation, a case study scenario is presented in detail before discussing further exemplary scenarios. The system model is defined broadly enough to incorporate all of the four discussed scenarios in order to represent a realistic use case in the demonstrated environment. This certainly increases planning complexity for the individual scenarios, but illustrates the flexibility of executing various types of missions as instructed by different LTL specifications.

7.1 System Definition

In order to illustrate the application of our proposed planning framework, we define an environment based on an existing Bosch office. The topological map of the office environment is depicted in Figure 3 and is the same for all robots. Costs are defined as the approximate travel times with the only exception that the costs for edges drawn bold are lower by a certain factor, i.e., navigation in the corridor is preferred. The basis for this map is an office floor accommodating around one hundred employees and including several meeting rooms. Two of the meeting rooms, located at the top right and top left of the map, have been declared as storage rooms including charging stations for the robots.

Points of interest in the map are labeled with atomic propositions according to Table 2. The desk spaces $d_1, \ldots, d_{14}$ indicate the particular locations of all desk areas. Located near the center of the map are a kitchen and a printer room, which will be used more extensively in some of the evaluation scenarios.

Figure 4 illustrates the robot capabilities as defined in the robot model. For simplicity, we assume for now that all robots have the same model, although this is not required. The central state in the model is the default state. From there, the robot can switch to its different operation modes such as

Proposition	Description
$d_1, \ldots, d_{14}$	Desk areas 1 to 14.
$m_1, \ldots, m_6$	Meeting rooms 1 to 6.
c	Locations of charging stations.
s_{k}	Storage for kitchen supplies.
s_{p}	Storage for office/printer supplies.
k	Coffee kitchen.
p	Printer room.
g	Garbage room, $g := service \wedge storage$.
e	Elevators.
$guiding$	The robot guides a person.
$carrying$	A document or anything similar is carried.
$docked$	The robot is docked in a charging station.
$camera$	The camera of the robot is turned on.

Table 2. Overview of propositions used in the given scenario. This list complements the propositions defined in Table 1.

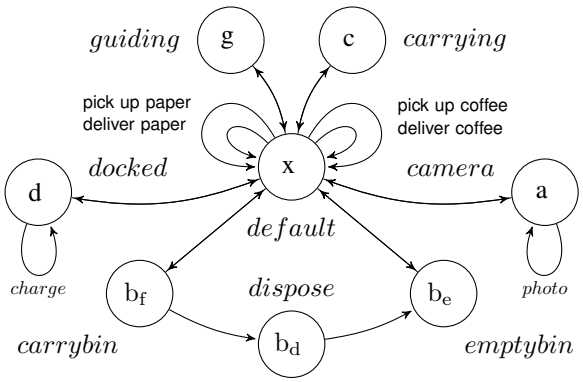

Figure 4. Location-independent states of a robot. Self-transitions at states denote actions which do not change the state of the robot. However, actions can still have effects on resources (effects are not depicted here).

visitor guidance or floor cleaning. If docked in a charging station, an action to charge the battery of the robot becomes available.

Of particular interest for the case study is the lower part of Figure 4, illustrating the capability of the robot to carry and empty paper bins. Specifically, the robot cannot be in multiple states at the same time and consequently, by labeling the states with their respective propositions as listed in Tables 1 and 2, it is specified that $default$ and $carrybin$ cannot be true at the same time.

While in the default state, the robot can pick up or deliver supplies like paper or coffee. This does not change the operation mode under the assumption that the robot can carry multiple supplies, but affects the robot-specific resource status. For example, picking up a paper pack increases the number of packs carried by the robot by one and is only possible below a certain capacity. Similarly, delivering this paper at the printer room decreases the resources of the robot, but increases the paper supplies of the printer.

Finally, a summary of the resource definitions is given by Table 3. For each of the resource dimensions, we prefer larger values in order to reflect the benefit of spending the costs for, e.g., picking up an object. Furthermore, we specify a proportional cost dependency of γ_{battery}. In addition, battery consumption of actions can also be defined individually as for any other resource. For example, the charging action increases the battery level despite the fact that it has non-negative costs. Also, all of the resource dimensions have an upper bound as given by Table 3. On the one hand, this reflects a limited capacity of these resources and on the other hand, this ensures termination of the planning process as given by Lemma 9.

Resource	Description
$\gamma_{\text{coffee}} \in [0, 1]$	Amount of coffee carried.
$\gamma_{\text{paper}} \in [0, 2]$	Amount of paper packs carried.
$\gamma_{\text{battery}} \in [0, 100]$	Battery level (in percent).
$\gamma_{\text{kitchen}} \in [0, 1]$	Amount of coffee in the kitchen.
$\gamma_{\text{printer}} \in [0, 3]$	Amount of paper packs at the printer.

Table 3. Resource variables used for planning. The first three rows are robot-specific, the rest are global resources.

7.2 Scenarios

Based on the defined environment, the system is ready to accept goal specifications. The following exemplary scenario definitions specify what is required as an input to the robot system after it has been configured as described in the previous subsection. In particular, the goal needs to be stated as an LTL formula to resemble a desired result of the robot operation for which a behavior should be planned. Also, the initial configuration of the system needs to be determined in order to correctly initialize the planner.

We restate Equation (2) again for reference as the first scenario and set $desk := d_5$.

Scenario 1 (Case Study). *"Empty a paper bin at d_5. Avoid the public area while carrying a bin."*

$$\begin{aligned}\phi_{\text{bin}} = \Diamond(&desk \wedge default \\ &\wedge \bigcirc((carrybin \,\mathcal{U}\, dispose) \wedge \Diamond default)) \\ &\wedge \Diamond(desk \wedge emptybin \wedge \bigcirc(desk \wedge default)) \\ &\wedge \Box(carrybin \implies \neg public)\end{aligned}$$

Initial configuration:

- $\mathcal{S}_{1a}$ – All six robots available, no resources considered.
- $\mathcal{S}_{1b}$ – Only R5 is available, no resources considered.
- $\mathcal{S}_{1c}$ – Only R1 is available, no resources considered.

Note that we select d_5 in this example as the most illustrative desk since, from looking at the map in Figure 3, the shortest direct path between d_5 and the garbage disposal location g would go through the public area. Consequently, we expect the robots to take a detour while carrying a full bin in order to satisfy the additional constraint. In contrast,

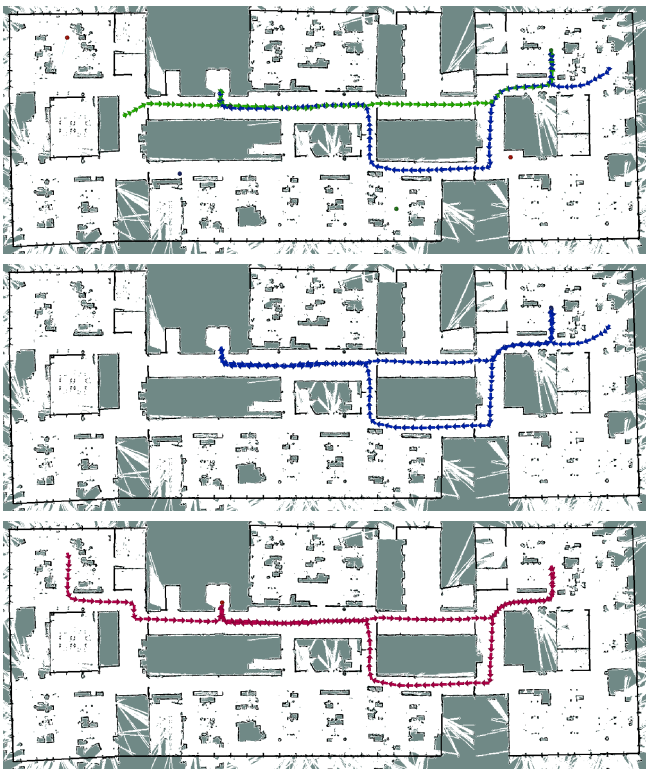

Figure 5. Localization recordings of simulating ϕ_{bin} with **(top)** all robots, **(mid)** only R5, **(bottom)** only R1 available.

we would expect the robots to cross this public area when providing the empty bin to d_5 since then, it is on the shortest path and not restricted.

Figure 5 (top) shows the robot paths for executing the resulting plan for satisfying the mission in scenario configuration $\mathcal{S}_{1a}$. The specification ϕ_{bin} is decomposed by the STAP planner and allocated to two different robots. R5, which is close to the desk d_5, picks up the full bin while at the same time R2 already delivers a new empty bin to the desk. In contrast, when only R5 would be available as depicted in Figure 5 (mid), the robot needs to return the empty bin to the desk by itself. As a third variation shown in Figure 5 (bottom), we assume that only R1 is available. In this case, R1 first delivers the empty bin before it picks up the full bin and finally disposes the garbage.

For a more detailed evaluation of the planning performance, specification expressiveness, and the flexibility in terms of scenario variations, we additionally use the following mission specifications. These specifications are chosen to represent missions which will typically be executed by a fleet of indoor service robots.

Scenario 2 (Supplies). *"Refill supplies at the printer room and the kitchen, ensure that sufficient battery is available."*

$$\begin{aligned}\phi_{\text{supplies}} = \Diamond(\gamma_{\text{printer}} \geq 2) \wedge \Diamond(\gamma_{\text{kitchen}} \geq 1) \\ \wedge \Box(\gamma_{\text{battery}} > 20)\end{aligned}$$

Initial configuration:

- $\mathcal{S}_{2a}$ – All six robots available, each robot has full battery. Supplies are both empty.
- $\mathcal{S}_{2b}$ – Like $\mathcal{S}_{2a}$, but R5 starts with reduced battery $\gamma_{0,\text{battery}} = 60$.
- $\mathcal{S}_{2c}$ – Like $\mathcal{S}_{2b}$, but R2 also starts with reduced battery $\gamma_{0,\text{battery}} = 70$.
- $\mathcal{S}_{2d}$ – Like $\mathcal{S}_{2c}$, but R4 also starts with reduced battery $\gamma_{0,\text{battery}} = 60$.

This scenario represents a common use case where the robots are required to refill some supplies. The additional battery constraint ensures long-term usage and forces the robots to charge if they run out of battery. For better illustration, we assume that actions have a higher battery consumption than usual.

Scenario 3 (Printer). *"Distribute printed copies of a document to the desks d_{10}, d_7, d_5, and avoid public areas while carrying the document. Make sure that there is sufficient paper left at the printer."*

$$\begin{aligned}\phi_{\text{printer}} = \Diamond(p \wedge carry\, \mathcal{U}\ (d_{10} \wedge \bigcirc\neg carry)) \\ \wedge \Diamond(p \wedge carry\, \mathcal{U}\ (d_7 \wedge \bigcirc\neg carry)) \\ \wedge \Diamond(p \wedge carry\, \mathcal{U}\ (d_5 \wedge \bigcirc\neg carry)) \\ \wedge \Box(carry \implies \neg public) \\ \wedge \Diamond(\gamma_{\text{printer}} > 0)\end{aligned}$$

Initial configuration:

- $\mathcal{S}_{3a}$ – All six robots available, no paper left at the printer.

We assume that each robot can only serve one desk at a time, which is defined by the requirement that the carry state needs to be left at each desk. This scenario is primarily used for comparison between STAP and a classic allocation approach for which the costs of task combinations need to be calculated first.

Scenario 4 (Video). *"Take a photo in the meeting rooms m_1, m_4, and m_6. Furthermore, deliver a document from desk d_5 to d_3 and guide a person waiting at desk d_{11} to meeting room m_6. The camera has to be turned off for privacy reasons while not in meeting rooms and the document is internal such that it should not be delivered through any public areas."*

$$\begin{aligned}\phi_{\text{video}} = \Diamond(m_1 \wedge photo) \wedge \Diamond(m_4 \wedge photo) \\ \wedge \Diamond(m_6 \wedge photo) \\ \wedge \Box(\neg meeting \implies \neg camera) \\ \wedge \Diamond(d_5 \wedge carry\, \mathcal{U}\ (d_3 \wedge \bigcirc\neg carry)) \\ \wedge \Box(carry \implies \neg public) \\ \wedge \Diamond(d_{11} \wedge guide\, \mathcal{U}\ (m_6 \wedge \bigcirc\neg guide)) \\ meeting := m_1 \vee \ldots \vee m_6\end{aligned}$$

Initial configuration:

- $\mathcal{S}_{4a}$ – All six robots available, no resources considered.
- $\mathcal{S}_{4b}$ – Only R1 and R2 available, no resources considered.

The last scenario consists of a composition of different tasks to represent a more complex mission. For further illustration, we provide a video showing the execution of configuration $\mathcal{S}_{4b}$ of this scenario on the real system given by Extension 1.

7.3 Discussion

Table 4 provides an overview of the complexity of the planning algorithm for some of the scenario variations. For all of the considered scenarios, the amount of explored labels is orders of magnitude less than the worst-case label space complexity. Consequently, these problems can be solved in reasonable time. For example, scenario $\mathcal{S}_{2a}$ only requires approximately 3.1×10^4 iterations as opposed to

	$\mathcal{S}_{1a}$	$\mathcal{S}_{2a}$	$\mathcal{S}_{4a}$	$\mathcal{S}_{4b}$
explored labels	7519	30 835	47 233	36 530
label cardinality	3.2×10^4	5.6×10^9	3.7×10^5	1.2×10^5
non-dom. actions	8264	40 919	54 099	40 408
dominated actions	17 032	71 278	95 078	55 753
switch transitions	3872	24 192	27 651	2892
planning time (s)	2.7	16.5	194.7	113.1

Table 4. Overview of labels explored before finding the solution (required iterations of Algorithm 1), worst-case cardinality of the label space (given by Equation 15), considered non-dominated actions (passing line 13 of Algorithm 1), considered dominated actions (not added due to line 13 of Algorithm 1), considered switch transitions (both added and dominated), approximate planning time in seconds (measured on an Intel® Xeon® E5-1620 v3, single-thread).

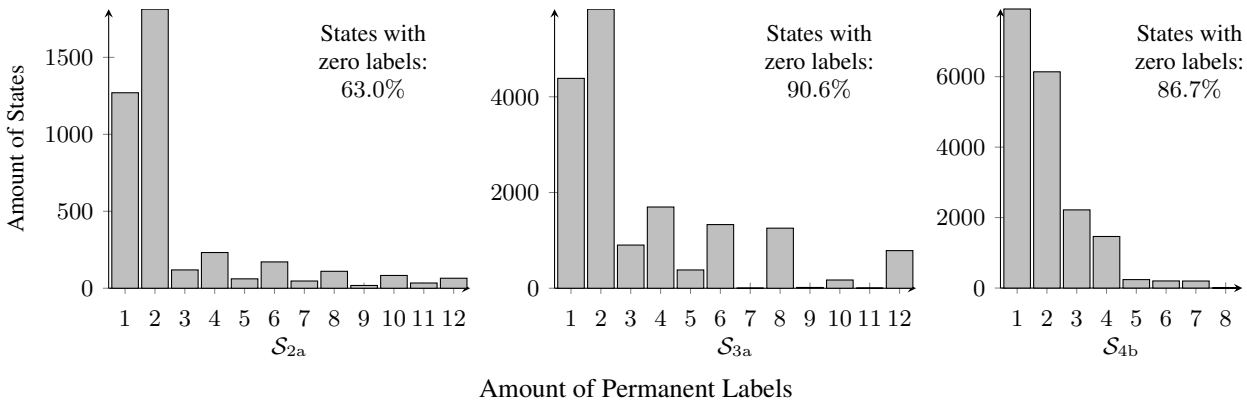

Figure 6. Distribution of the states having a certain amount of permanent labels after planning. Scenario $\mathcal{S}_{2a}$ has additional 769 states with more than 12 labels and at most 68 labels per state, $\mathcal{S}_{3a}$ has 253 states with more than 12 labels and at most 29 labels per state.

around 5.6×10^9 iterations when assuming $\gamma_{\Delta,\text{battery}} = 0.01$, which is the case for the presented experiments. This can be explained by the following two reasons. First, only a small subset of the theoretically possible labels are actually feasible in the model by following the available actions and second, a significant amount of actions is identified as dominated such that no suboptimal labels are added.

Furthermore, note the qualitative difference of the scenario variations $\mathcal{S}_{4a}$ and $\mathcal{S}_{4b}$ regarding the switch transitions. In $\mathcal{S}_{4b}$ with only two robots available, most of the planning time is spent on finding rather long action sequences to satisfy combinations of the tasks. In contrast, when all six robots are available as in $\mathcal{S}_{4a}$, more planning time is used on allocation of the single tasks. In fact, the longest action sequences for individual robots considered during planning of $\mathcal{S}_{4a}$ have less than half of the cost compared to the result of $\mathcal{S}_{4b}$.

Figure 6 shows the distribution of the amount of states which have a certain amount of permanent labels after termination of Algorithm 1. Most of the states only have a few Pareto-optimal labels after planning, mainly resulting from different task allocation options. For example, scenario $\mathcal{S}_{4b}$ does not consider resources, but still has states with more than one label. In contrast, scenario $\mathcal{S}_{2a}$ has a mission which is much simpler, but primarily requires to plan for resources. Taking a closer look at the label distribution, it turns out that especially the states around docking stations where the robots can charge their battery have the most labels. This appears reasonable since charging has costs, but also improves the battery resource, resulting in a more diverse Pareto front. Regardless of the scenario, a significant fraction of states has zero labels, indicating that these states did not need to be considered during planning.

For scenario 2, we vary the initial configuration to investigate how the STAP planner adjusts its solution. The resulting cost vectors are summarized in Table 5. First, in scenario $\mathcal{S}_{2a}$, the mission is solved by R1 refilling the coffee and R4 and R5 each picking up a paper pack at the storage near room m_2. Note that we set the cost of both picking up paper and delivering it to 2.0 each. Consequently, letting either R4 or R5 take both paper packs would result in a higher team cost.

As a first variation, we change the initial battery level of R5 to a value low enough such that it is not sufficient to stay above 20% as required by the constraint. Now, the solution found by the planner uses R2 instead to pick up the second paper pack at the storage near room m_1, which is still cheaper than letting R4 carry the paper alone. For $\mathcal{S}_{2c}$, we also reduce the initial battery of R2 below the level sufficient for this mission and now, R4 is indeed assigned with picking up both paper packs on its own. Finally, also the initial battery for R4 is reduced in $\mathcal{S}_{2d}$. Since R3 and R6 are far away from any storage location, it is now cheaper to let both R4 and R5 briefly charge their battery in order to complete the mission.

In order to compare simultaneous task allocation and planning (STAP) with a classical approach of planning task costs first and then using a task assignment algorithm (COMB), we compare both approaches for scenario 3. For a better comparison, we only consider the planning

	$\mathcal{S}_{2a}$	$\mathcal{S}_{2b}$	$\mathcal{S}_{2c}$	$\mathcal{S}_{2d}$
R1	49.4	49.4	49.4	49.4
R2	0.0	50.9	0.0	0.0
R3	0.0	0.0	0.0	0.0
R4	47.7	47.7	51.7	57.7
R5	48.9	0.0	0.0	58.9
R6	0.0	0.0	0.0	0.0

Table 5. Cost vectors of the optimal solutions for all variations of scenario 2. Zero cost for a robot indicates that this robot does not participate in executing the mission.

	$\mathcal{S}_{3a}$	$\mathcal{S}_{4a}$
STAP for team	61 776	47 233
COMB for R1	351 937	82 315
COMB for R2	273 181	82 232
COMB for R3	228 825	80 224
COMB for R4	164 516	67 838
COMB for R5	193 792	68 032
COMB for R6	139 309	73 186

Table 6. Explored labels (required iterations of Algorithm 1) for a comparison of STAP with planning all task combinations explicitly for the individual robots.

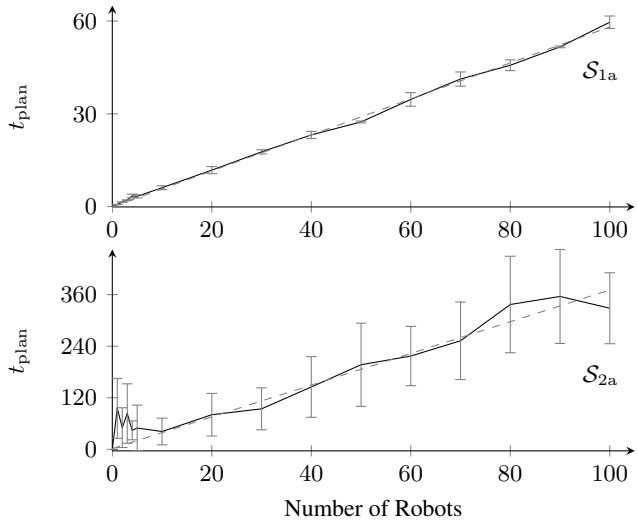

Figure 7. Average planning time (in seconds) for $\mathcal{S}_{1a}$ (top) and $\mathcal{S}_{2a}$ (bottom) with different team sizes. For each planning run, the respective number of robots was instantiated at random positions (uniformly distributed) in the default state and with random battery values (uniformly distributed in the range [50,100]). Error bars indicate the range within one standard deviation. Shown as dashed line is a linear approximation.

part of COMB and assume that the assignment problem afterwards is trivial given its small size. Table 6 provides an overview of the required planner iterations where the number for planning all combinations is given by the sum of the individual planning runs. Even under the assumption that the robots can plan their task costs in parallel, the maximum across the single robots is already significantly higher than STAP for the whole team in both cases.

Figure 7 shows the scalability of STAP to a larger team, up to one hundred robots. As can be seen, the planning time scales roughly linearly in practice with an increase of approximately 0.58 seconds (Scenario $\mathcal{S}_{1a}$) or 3.69 seconds (Scenario $\mathcal{S}_{2a}$) per additional robot. For Scenario $\mathcal{S}_{2a}$, the variability of the required planning time is relatively high, indicating a significant dependency of the planning time on the initial conditions. For example, finding the optimal solution for 50 robots can take longer than for 100 while still being within one standard deviation from the average. A reason for this observation might be that $\mathcal{S}_{2a}$ highly depends on the battery level of the robots, including planning of which to charge, while $\mathcal{S}_{1a}$ does not consider resources.

In addition to the initial conditions of the robots, also other aspects can influence the planning performance. For example, the way how a goal specification is formulated can make a difference. Translation of the specification into an automaton relaxes the dependency on the specific formulation of the goal as long as it expresses an equivalent specification. But still, even semantically different specifications can lead to the same results. This is primarily due to the limitations of the modeled system, introducing additional implicit constraints which are not necessarily reflected in the formal specification.

Another influence factor for the planning performance is the kind of considered resource constraints. For example, the battery level is highly correlated with costs when executing actions which consume battery as well as increase costs. However, it is inversely proportional for charging actions which increase the battery level at the same time as increasing costs, resulting in significantly more Pareto-optimal labels. This has been noted in the discussion of Figure 6, where results show that the states with most labels are close to charging stations.

Finally, we would like to emphasize that planning time and model complexity highly depend on the abstraction level of actions, such as the density of waypoints in the topological map. Also, having a general model of the system such that the model is usable for a large variety of missions, as done in the approach of this paper, is clearly more expressive and complex than having only a subset of actions depending on a specific mission.

8 Conclusion

We introduce a problem category called *Simultaneous Task Allocation and Planning (STAP)* to address a limitation of classical task assignment problems typical for multi-robot systems: when planning the robot behaviors to execute given tasks, each combination of tasks needs to be planned explicitly for each of the robots in order to calculate their costs. Instead, we propose to construct a team model in order to combine planning and allocation in a single step.

Furthermore, we present a specific STAP approach for planning optimal behaviors from finite Linear Temporal Logic specifications under resource constraints. For this purpose, a specific team model is constructed to combine planning of independent actions for the robots with allocating parts of the mission to the individual robots. A multi-objective optimization planner then finds a plan to solve the mission with minimal team costs defined as the maximal robot cost.

The approach is evaluated and discussed based on exemplary scenarios in an existing office environment in order to investigate its practical applicability. Our results show that simultaneous task allocation and planning is more efficient than explicitly calculating the unknown costs of all task combinations in advance when dealing with complex mission goals like temporal logic specifications.

In fact, many service robot systems also in other environments, e.g., hotels or hospitals, and also robot systems in other domains like indoor logistics, manufacturing, or domestic applications can be modeled and instructed in a similar way. The presented STAP approach and the theoretical results are equally well applicable to these other application domains.

Future work is dedicated to extending the idea of STAP to more multi-agent problem classes in order to improve the efficiency with which they can be solved. Especially relaxing the assumption that actions are deterministic leads to further research questions of robustly allocating tasks which are only fulfilled with a certain probability. A generalization of the proposed decomposition approach to infinite LTL specifications is also of great interest.

Funding

This work was supported by the EU H2020 Research and Innovation Programme under GA No. 731869 (Co4Robots). The third author is also supported by the H2020 ERC Starting Grant

BUCOPHSYS, the Swedish Research Council (VR), the Swedish Foundation for Strategic Research (SSF), and the Knut och Alice Wallenberg Foundation (KAW).

Notes

1. The definition for the YAML format can be found at http://www.yaml.org/start.html
2. Spot is a library for LTL and model checking, see http://spot.lrde.epita.fr
3. The definition for the HOA format can be found at http://adl.github.io/hoaf
4. FlexBE is a user-friendly behavior engine for ROS, see http://flexbe.github.io

References

Manoj Agarwal, Naveen Kumar, and Lovekesh Vig. Non-additive multi-objective robot coalition formation. *Expert Systems with Applications*, 41(8):3736–3747, 2014.

Rajeev Alur, Tomás Feder, and Thomas A Henzinger. The benefits of relaxing punctuality. *Journal of the ACM (JACM)*, 43(1): 116–146, 1996.

Christopher Amato, George Konidaris, Gabriel Cruz, Christopher A Maynor, Jonathan P How, and Leslie P Kaelbling. Planning for decentralized control of multiple robots under uncertainty. In *IEEE International Conference on Robotics and Automation*, pages 1241–1248, 2015.

Marco Autili, Lars Grunske, Markus Lumpe, Patrizio Pelliccione, and Antony Tang. Aligning qualitative, real-time, and probabilistic property specification patterns using a structured english grammar. *IEEE Transactions on Software Engineering*, 41(7):620–638, 2015.

Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT press Cambridge, 2008.

Calin Belta, Antonio Bicchi, Magnus Egerstedt, Emilio Frazzoli, Eric Klavins, and George J Pappas. Symbolic planning and control of robot motion [grand challenges of robotics]. *IEEE Robotics & Automation Magazine*, 14(1):61–70, 2007.

Ronen I Brafman and Carmel Domshlak. From One to Many: Planning for Loosely Coupled Multi-Agent Systems. In *International Conference on Automated Planning and Scheduling (ICAPS)*, pages 28–35, 2008.

Rainer E Burkard and Eranda Cela. Linear assignment problems and extensions. In *Handbook of combinatorial optimization*, pages 75–149. Springer, 1999.

Yushan Chen, Xu Chu Ding, Alin Stefanescu, and Calin Belta. Formal approach to the deployment of distributed robotic teams. *IEEE Transactions on Robotics*, 28(1):158–171, 2012.

Giuseppe De Giacomo and Moshe Y Vardi. Linear temporal logic and linear dynamic logic on finite traces. In *International Joint Conference on Artificial Intelligence*, pages 854–860. Association for Computing Machinery, 2013.

Frits de Nijs, Erwin Walraven, Mathijs Michiel de Weerdt, and Matthijs TJ Spaan. Bounding the Probability of Resource Constraint Violations in Multi-Agent MDPs. In *AAAI Conference on Artificial Intelligence*, pages 3562–3568, 2017.

Alexandre Duret-Lutz, Alexandre Lewkowicz, Amaury Fauchille, Thibaud Michaud, Etienne Renault, and Laurent Xu. Spot 2.0 — a framework for LTL and ω-automata manipulation. In *International Symposium on Automated Technology for Verification and Analysis (ATVA)*. Springer, October 2016.

Edmund H Durfee and Shlomo Zilberstein. Multiagent planning, control, and execution. *Multiagent Systems*, pages 485–545, 2013.

Xavier Gandibleux, Frédéric Beugnies, and Sabine Randriamasy. Martins' algorithm revisited for multi-objective shortest path problems with a MaxMin cost function. *4OR*, 4(1):47–59, 2006.

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

Matthew C Gombolay, Ronald Wilcox, and Julie A Shah. Fast Scheduling of Multi-Robot Teams with Temporospatial Constraints. In *Robotics: Science and Systems*, 2013.

Meng Guo and Dimos V Dimarogonas. Multi-agent plan reconfiguration under local LTL specifications. *The International Journal of Robotics Research*, 34(2):218–235, 2015.

Malte Helmert. The fast downward planning system. *Journal of Artificial Intelligence Research (JAIR)*, 26:191–246, 2006.

Stefan Irnich and Guy Desaulniers. Shortest path problems with resource constraints. In *Column generation*, pages 33–65. Springer, 2005.

Sertac Karaman and Emilio Frazzoli. Vehicle routing problem with metric temporal logic specifications. In *Conference on Decision and Control (CDC)*, pages 3953–3958. IEEE, 2008.

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

Hadas Kress-Gazit, Georgios E Fainekos, and George J Pappas. Translating structured english to robot controllers. *Advanced Robotics*, 22(12):1343–1359, 2008.

Hadas Kress-Gazit, Georgios E Fainekos, and George J Pappas. Temporal-logic-based reactive mission and motion planning. *IEEE Transactions on Robotics*, 25(6):1370–1381, 2009.

Orna Kupferman and Moshe Y Vardi. Model checking of safety properties. *Formal Methods in System Design*, 19(3):291–314, 2001.

Bruno Lacerda, David Parker, and Nick Hawes. Optimal and dynamic planning for Markov decision processes with co-safe LTL specifications. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1511–1516. IEEE, 2014.

Morteza Lahijanian, Matthew R Maly, Dror Fried, Lydia E Kavraki, Hadas Kress-Gazit, and Moshe Y Vardi. Iterative temporal planning in uncertain environments with partial satisfaction guarantees. *IEEE Transactions on Robotics*, 32(3):583–599, 2016.

Kevin Leahy, Dingjiang Zhou, Cristian-Ioan Vasile, Konstantinos Oikonomopoulos, Mac Schwager, and Calin Belta. Provably correct persistent surveillance for unmanned aerial vehicles subject to charging constraints. In *Experimental Robotics*, pages 605–619. Springer, 2016.

Hang Ma, Sven Koenig, Nora Ayanian, Liron Cohen, Wolfgang Hönig, TK Kumar, Tansel Uras, Hong Xu, Craig Tovey, and Guni Sharon. Overview: Generalizations of multi-agent path finding to real-world scenarios. *arXiv preprint arXiv:1702.05515*, 2017.

Oded Maler and Dejan Nickovic. Monitoring temporal properties of continuous signals. In *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, pages 152–166.

Springer, 2004.

Ernesto Queiros Vieira Martins. On a multicriteria shortest path problem. *European Journal of Operational Research*, 16(2): 236–245, 1984.

Alexandros Nikou, Jana Tumova, and Dimos V Dimarogonas. Cooperative task planning of multi-agent systems under timed temporal specifications. In *American Control Conference (ACC)*, pages 7104–7109. IEEE, 2016.

Raz Nissim and Ronen I Brafman. Distributed Heuristic Forward Search for Multi-agent Planning. *Journal of Artificial Intelligence Research (JAIR)*, 51:293–332, 2014.

José Manuel Paixão and José Luis Santos. Labeling Methods for the General Case of the Multi-objective Shortest Path Problem–A Computational Study. In *Computational Intelligence and Decision Making*, pages 489–502. Springer, 2013.

David W Pentico. Assignment problems: A golden anniversary survey. *European Journal of Operational Research*, 176(2): 774–793, 2007.

Philipp Schillinger, Mathias Bürger, and Dimos V Dimarogonas. Decomposition of finite LTL specifications for efficient multi-agent planning. In *Distributed autonomous robotic systems (DARS)*. Springer, 2016a.

Philipp Schillinger, Stefan Kohlbrecher, and Oskar von Stryk. Human-Robot Collaborative High-Level Control with Application to Rescue Robotics. In *IEEE International Conference on Robotics and Automation*, Stockholm, Sweden, May 2016b.

Philipp Schillinger, Mathias Bürger, and Dimos V Dimarogonas. Multi-Objective Search for Optimal Multi-Robot Planning with Finite LTL Specifications and Resource Constraints. In *IEEE International Conference on Robotics and Automation*. Singapore, 2017.

Pedro M Shiroma and Mario FM Campos. CoMutaR: A framework for multi-robot coordination and task allocation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4817–4824. IEEE, 2009.

Jana Tumova and Dimos V Dimarogonas. Multi-agent planning under local LTL specifications and event-based synchronization. *Automatica*, 70:239–248, 2016.

Matthew Turpin, Nathan Michael, and Vijay Kumar. An approximation algorithm for time optimal multi-robot routing. In *Algorithmic Foundations of Robotics XI*, pages 627–640. Springer, 2015.

Alphan Ulusoy, Stephen L Smith, Xu Chu Ding, Calin Belta, and Daniela Rus. Optimality and robustness in multi-robot path planning with temporal logic constraints. *The International Journal of Robotics Research*, 32(8):889–911, 2013.

Zhi Yan, Nicolas Jouandeau, and Arab Ali Cherif. A survey and analysis of multi-robot coordination. *International Journal of Advanced Robotic Systems*, 10:399, 2013.

Robert Zlot and Anthony Stentz. Complex task allocation for multiple robots. In *IEEE International Conference on Robotics and Automation*, pages 1515–1522. IEEE, 2005.

A Index to Multimedia Extensions

Videos are available on the IJRR YouTube channel at `http://www.youtube.com/user/ijrrmultimedia`.

Extension	Media Type	Description
1	Video	Example for execution of scenario $\mathcal{S}_{4\mathrm{b}}$ by robots in the real office environment.