

Multi-Robot LTL Planning Under Uncertainty

Claudio Menghi¹[0000-0001-5303-8481], Sergio Garcia¹[0000-0001-7369-2480],
Patrizio Pelliccione¹[0000-0002-5438-2281], and Jana
Tumova²[0000-0003-4173-2593]

¹ Chalmers — University of Gothenburg, Sweden

{claudio.menghi,sergio.garcia,patrizio.pelliccione}@gu.se

² Royal Institute of Technology (KTH), Sweden

tumova@kth.se

Abstract. Robot applications are increasingly based on teams of robots that collaborate to perform a desired mission. Such applications ask for *decentralized* techniques that allow for tractable automated planning. Another aspect that current robot applications must consider is *partial knowledge* about the environment in which the robots are operating and the uncertainty associated with the outcome of the robots' actions.

Current planning techniques used for teams of robots that perform complex missions do not systematically address these challenges: (1) they are either based on centralized solutions and hence not scalable, (2) they consider rather simple missions, such as A-to-B travel, (3) they do not work in partially known environments. We present a planning solution that decomposes the team of robots into subclasses, considers missions given in temporal logic, and at the same time works when only partial knowledge of the environment is available. We prove the correctness of the solution and evaluate its effectiveness on a set of realistic examples.

1 Introduction

A *planner* is a software component that receives as input a model of the robotic application and computes a *plan* that, if performed, allows the achievement of a desired mission [26]. As done in some recent works in robotics (e.g., [4],[3]), we assume that a robot application is defined using finite transition systems and each robot has to achieve a mission, indicated as *local mission*, specified as an LTL property. As opposed to more traditional specification means, such as consensus or trajectory tracking in robot control, A-to-B travel in robot motion planning, or STRIPS or PDDL in robot task planning, LTL allows the specification of a rich class of temporal goals that include surveillance, sequencing, safety, or reachability [8]. LTL has also been recently considered as a reference logic for the specification of patterns for robotic mission, in which LTL template solutions are provided to recurrent specification problems [33].

Several works studied centralized planners that are able to manage *teams* of robots that collaborate to achieve a global mission (e.g., [20],[28],[34]), and how to decompose a global mission into a set of local missions (e.g., [36],[16],[16],[38]).

Local missions have been recently exploited by *decentralized* planners [38], which avoid the expensive centralized planning by analyzing the satisfaction of local missions inside subsets of the robots.

Another aspect that current planners must consider is partial knowledge about the environment in which the robots operate. Partial knowledge has been strongly studied by the software engineering and formal methods communities. Partial models have been used to support requirement analysis and elicitation [32],[31],[27], to help designers in producing a model of the system that satisfies a set of desired properties [29],[40],[39], and to verify whether already designed models possess some properties of interest [2],[30],[5],[7]. However, most of the existing planners assume that the environment in which the robots are deployed is known [9]. Literature considering planners that work in partially specified environments is more limited and usually rely on techniques based on probabilistic models (e.g., [35],[12],[10]). Furthermore, decentralized planners are rarely applied when partial knowledge about the robot application is present [16].

Contribution. This work presents MAPmAKER (Multi-robot pLAnner for Partially Known EnviRonments), a *novel decentralized* planner for *partially* known robotic applications. Given a team of robots and a local mission for each robot, MAPmAKER partitions the set of robots into classes based on dependencies dictated by the local missions of each robot. For each of these classes, it explores the state space of the environment and the models of the robot searching for definitive and possible plans. Definitive plans ensure the satisfaction of the local mission for each robot. Possible plans may satisfy the local mission due to some unknown information in the robotic application. MAPmAKER chooses the plan to be executed among definitive and possible plans that allow the achievement of the mission.

Specific novel contributions. The specific contributions are:

- (1) We define the concept of *partial robot model*. This definition customizes partial models (e.g., PKS [5] and MTS [24]) in a robotic domain context [38] allowing the description of the robots and their environments when partial information is available. A partial robot model allows considering three types of partial information: partial knowledge about the execution of transitions (possibility of changing the robot location), the service provision (whether the execution of an action succeed in providing a service) and the meeting capabilities (whether a robot can meet with another).
- (2) We define the concept of local mission satisfaction for partial robot models and the thorough LTL *word* semantics. This semantics extends the well known thorough LTL semantics [6] for partial models and allows the thorough evaluation of an LTL formula over words. This definition is needed to define when an LTL formula is satisfied or possibly satisfied on a given plan.
- (3) We define the distributed planning problem for partially specified robots.
- (4) We prove that under certain assumptions the planning problem can be solved by relying on two calls of a “classical” planner.
- (5) We propose a distributed planning algorithm and we prove its correctness. The distributed algorithm enables a tractable planning.

- (6) We evaluate the proposed algorithm on a robot application obtained from the RoboCup Logistics League competition [18] and on a robotic application working in an apartment which is part of a large residential facility for senior citizens [1]. The results show the effectiveness of MAPmAKER.

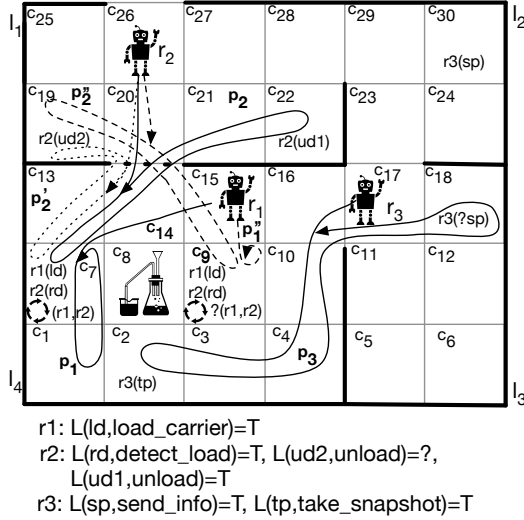
Organization. Section 2 presents our running example. Section 3 describes the problem and Sect. 4 describes how MAPmAKER supports partial models. Section 5 presents the proposed planning algorithm. Section 6 evaluates the approach. Section 7 presents related work. Section 8 concludes with final remarks and future research directions.

2 Running Example

Robots r_1, r_2 , and r_3 are deployed in the environment graphically described in Fig. 1. This environment represents a building made up of rooms l_1, l_2, l_3 , and l_4 . The environment is partitioned in cells, each labeled with an identifier in c_1, c_2, \dots, c_{30} . Robots r_1, r_2 , and r_3 are placed in their initial locations. Each robot is able to move from one cell to another, by performing action *mov*. The robots are also able to perform the following actions. Robot r_1 is able to load debris of the building by performing action *ld*. Given a robot r and a generic action α , in Fig. 1 the cells in which r can perform α are marked with the label $r(\alpha)$. Robot r_2 can wait until another robot loads debris on it by performing action *rd* and can unload debris by performing one of the two actions *ud1* and *ud2*. Actions *ud1* and *ud2* use different actuators. Specifically, action *ud1* uses a gripper while action *ud2* exploits a dump mechanism. Robot r_3 is able to take pictures by performing action *tp* and send them using a communication network through the execution of action *sp*. Symbols $r_1(ld)$, $r_2(rd)$, $r_2(ud1)$, $r_2(ud2)$, $r_3(tp)$, and $r_3(sp)$ mark the regions where actions can be executed by the robots, while movement actions are not reported for graphical reasons. Actions *ld*, *rd*, *tp*, and *sp* are associated with the services *load_carrier*, *detect_load*, *take_snapshot*, and *send_info*, respectively, which are high-level functionalities provided by the robot when actions are performed. Actions *ud1* and *ud2* are associated with service *unload*. The labels $L(\pi, \alpha) = \top$ below Fig. 1 are used to indicate that a service π is associated with action α . Robots must meet and synchronously execute actions. Robots r_1 and r_2 must meet in cell c_7 and synchronously execute actions *ld* and *rd*, respectively. Rotating arrows marked with robots identifiers, are used to indicate where robots must meet.

The *global mission* the team of robots has to achieve is to check whether toxic chemicals have been released by the container located in l_4 . We assume that the mission is specified through a set of *local missions* assigned to each robot of the team and described in LTL as in Fig. 2. Informally, while r_3 continuously takes pictures and sends them using the communication network, r_1 and r_2 remove debris to allow r_3 having a better view on the container. The pictures allow verifying whether toxic chemicals have been released.

Partial knowledge about the actions execution. The robots can move between cells separated by grey lines, while they cannot cross black bold lines.



$\phi_1 = G(F(load_carrier))$: periodically robot r_1 loads debris on r_2 (by providing service *load_carrier*).
 $\phi_2 = G(F(detect_load \wedge F(unload)))$: robot r_2 receives debris (service *detect_load*) and brings them to an appropriate unload area (service *unload*).
 $\phi_3 = G(F(take_snapshot \wedge F(send_info)))$: robot r_3 repeatedly takes pictures (service *take_snapshot*) and sends them using the communication network (service *send_info*).

Fig. 1. An example showing the model of the robots and their environment. Plans are represented by trajectories marked with arrows.

Fig. 2. The local missions assigned to each robot.

It is unknown whether it is possible to move between cells c_{14} and c_{20} since the structure may have been affected by collapses. This is indicated using a dashed black bold line. It is also unknown whether robot r_3 can send pictures using a communication network, performing action s_p in location l_3 and specifically in cell c_{18} . Locations of the environment where it is unknown if an action can be executed are marked with the name of the action preceded by symbol $?$.

Unknown service provisioning. There are cases in which actions can be executed but there is uncertainty about service provisions. For example, actions $ud1$ and $ud2$ of robot r_2 unload the robot. Action $ud1$ will always be able to provide the *unload* service, while it is unknown whether $ud2$ provides this service since its effectiveness depends on the size of the collected debris. In Fig. 1, the label $L(ud2, unload) = ?$ indicates that there is partial knowledge about the provision of the *unload* service when action $ud2$ is performed.

Unknown meeting capabilities. It is unknown whether robots r_1 and r_2 can meet in one cell of the environment. For example, a collapse in the roof of the building may forbid the two robots to concurrently execute actions ld and rd , e.g., there is not enough space for $r1$ to load $r2$. Unknown meeting capabilities are indicated with rotating arrows labeled with the symbol $?$. For example, in Fig. 1, it is unknown whether robots r_1 and r_2 are able to meet in cell c_9 .

3 Modeling partial knowledge in a robotic application

We extend the model of a robotic application [38] with partial knowledge.

Definition 1. Consider a set of robots \mathcal{R} . A partial robot model of a single robot $r \in \mathcal{R}$ is a tuple $r = (S, \text{init}, A, \Pi, T, T_p, \text{Meet}, \text{Meet}_p, L)$, where S is a finite set of states; $\text{init} \in S$ is the initial state; A is a finite set of actions; Π is a set of services; $T, T_p : S \times A \rightarrow S$ are partial deterministic transition functions such that for all $s, s' \in S$ and $\alpha \in A$, if $T(s, \alpha) = s'$ then $T_p(s, \alpha) = s'$; $L : A \times \Pi \rightarrow \{\top, \perp, ?\}$ is the service labeling; $\text{Meet}, \text{Meet}_p : S \rightarrow (\wp(\mathcal{R}) \cup \{\#\})$ are functions ensuring:

- (1) for all $s \in S$ either $\text{Meet}(s) \subseteq \wp(\mathcal{R})$ or $\text{Meet}(s) = \{\#\}$;
- (2) for all $s \in S$ such that $\text{Meet}(s) \neq \emptyset$, $\text{Meet}_p(s) = \text{Meet}(s)$.

A partial robot model has three sources of partial knowledge:

Partial knowledge about the action execution. A transition $T_p(s, \alpha) = s'$ is called *possible transition* and is indicated as $s \xrightarrow{\alpha} s'$. Given two states s and s' and an action α , if $T_p(s, \alpha) = s'$ and $T(s, \alpha) \neq s'$ (i.e., it is undefined) it is uncertain whether the robot moves from s to s' by performing α . A transition $T_p(s, \alpha) = s'$, such that $T(s, \alpha) \neq s'$ is called *maybe transition*. A transition $T_p(s, \alpha) = s'$, such that $T(s, \alpha) = s'$, is a *definitive transition* and is indicated as $s \xrightarrow{\alpha} s'$.

Partial knowledge about the service provisioning. The service labeling specifies whether a service $\pi \in \Pi$ is provided when an action $\alpha \in A$ is performed. If $L(\alpha, \pi) = \top$, then the service π is provided by the action α , if $L(\alpha, \pi) = \perp$, then the service π is not provided by the action α , and, finally, if $L(\alpha, \pi) = ?$, then it is uncertain whether service π is provided when the action α is executed.

Partial knowledge about robot meeting capabilities. Function Meet_p labels a state s with a set of robots that must meet with r in s . If $\text{Meet}(s) = \emptyset$ and $\text{Meet}_p(s) \neq \emptyset$, it is uncertain whether the robots in $\text{Meet}_p(s)$ can meet. Otherwise $\text{Meet}(s) = \text{Meet}_p(s)$, i.e., the meeting capabilities are definitive. When $\text{Meet}(s) = \{\#\} = \text{Meet}_p(s)$ the meeting is not possible.

A definitive robot model is a partial robot model that does not contain partial information about the action execution, the service provisioning, and the meeting capabilities.

Variations of finite state machines are strongly used by planning algorithms in the robotic community (e.g. [17],[16],[38]). They can be directly computed by abstracting maps of real environments (e.g. [21]) or generated for other types of specifications such as Ambient Calculus (e.g., [37]). Within the context of this work we assume the model of the robotic application is given. Note that the notion of partial robot model does not extends PKS [5] and MTS [24] in terms of expressive power, but allows handling explicitly partial knowledge about the execution of transitions (possibility of changing the robot location), the service provision (whether the execution of an action succeed in providing a service), and the meeting capabilities (whether a robot can meet with another), which are key aspects to be considered in robotic applications [38].

A plan describes the states and actions a robot has to traverse and perform.

Definition 2. Given a partial robot model $r = (S, \text{init}, A, \Pi, T, T_p, \text{Meet}, \text{Meet}_p, L)$, a definitive plan of r is an infinite alternating sequence of states

and actions $\beta = s_1, \alpha_1, s_2, \alpha_2, \dots$, such that $s_1 = \text{init}$, for all $i \geq 1$ and $\pi \in \Pi$, $s_i \xrightarrow{\alpha_i} s_{i+1}$, $\text{Meet}(s_i) = \text{Meet}_p(s_i)$, $\text{Meet}(s_i) \neq \{\#\} \neq \text{Meet}_p(s_i)$ and $L(\alpha_i, \pi) \neq ?$. A possible plan of r is infinite alternating sequence of states and actions $\beta = s_1, \alpha_1, s_2, \alpha_2, \dots$, such that $s_1 = \text{init}$, and for all $i \geq 1$, $s_i \xrightarrow{\alpha_i} s_{i+1}$ and $\text{Meet}(s_i) \neq \{\#\} \neq \text{Meet}_p(s_i)$.

The plan $c_{17}, \text{mov}, c_{23}, \text{mov}, c_{29}, \text{mov}, c_{30}, (sp, c_{30})^\omega$ is a definitive plan for robot r_3 since all the transitions, service provisioning and meeting capabilities are definitive. The plan $c_{26}, \text{mov}, c_{20}, \text{mov}, c_{14}, (\text{mov}, c_{14})^\omega$ for robot r_1 is a possible plan since the transition from cell c_{20} to c_{14} executed by performing action mov is a maybe transition.

Definition 3. A partial robot application \mathcal{H} is a set of partial robot models $\{r_1, r_2, \dots, r_N\}$ such that for every couple of partial robot models $r_n, r_m \in \mathcal{H}$, where $r_n = (S_n, \text{init}_n, A_n, \Pi_n, T_n, T_{p,n}, \text{Meet}_n, \text{Meet}_{p,n}, L_n)$ and $r_m = (S_m, \text{init}_m, A_m, \Pi_m, T_m, T_{p,m}, \text{Meet}_m, \text{Meet}_{p,m}, L_m)$, the following is satisfied $\Pi_n \cap \Pi_m = \emptyset$ and $A_n \cap A_m = \emptyset$.

In Definition 3 and in the rest of the paper we will assume that the sets of services and actions of different robots are disjoint. This is not a strong limitation for usage in real applications, where an action α (resp. service π) shared among two robots r_1 and r_2 can be mapped in two actions α_1 and α_2 (resp. π_1 and π_2) to be used in robots r_1 and r_2 . The local mission should then be changed accordingly, i.e., occurrences of service π must be replaced with the formula $\pi_1 \vee \pi_2$.

Let us consider a partial robot application \mathcal{H} . We use \mathcal{B}_d to indicate a set $\{\beta_1^d, \beta_2^d, \dots, \beta_N^d\}$ of definitive plans, where β_n^d is a definitive plan for the partial robot r_n in \mathcal{H} . We use \mathcal{B}_p to indicate a set $\{\beta_1^p, \beta_2^p, \dots, \beta_N^p\}$ of possible plans, where β_n^p is the possible plan for the partial robot r_n in \mathcal{H} .

The notion of plan should reflect the meeting scheme, meaning that robots should enter and leave locations of the environment depending on the functions Meet and Meet_p . Consider a partial robot application \mathcal{H} and a set $\mathcal{B} = \{\beta_1, \beta_2, \dots, \beta_N\}$ of definitive or possible plans, where β_n is the plan for the robot r_n in \mathcal{H} . We indicate as *compatible plans* the plans in \mathcal{B} that ensure a meeting scheme.

Definition 4. Given a set of definitive (possible) plans \mathcal{B}_d (\mathcal{B}_p), the set \mathcal{B}_d (\mathcal{B}_p) is compatible if the following holds for all $r_n \in \mathcal{H}$, and $j \geq 1$. For each plan $\beta_n = s_{n,1}, \alpha_{n,1}, s_{n,2}, \alpha_{n,2}, \dots$ of each robot r_n if $s_{n,j-1} \neq s_{n,j}$ and $\text{Meet}(s_{n,j}) \neq \emptyset$ then

- (1) $s_{n,j} = s_{n,j+1}$;
- (2) for all $r_m \in \text{Meet}_n(s_{n,j})$, it holds that $s_{m,j} = s_{n,j}$ and $s_{n,j} = s_{m,j+1}$.

The condition enforces the constraint dictated by the Meet function when state $s_{n,j}$ is entered. Specifically, (1) ensures that after a robot r_n meets another in a state $s_{n,j}$, it proceeds with the execution of a transition that has state $s_{n,j}$ as a source and destination and (2) ensures that each robot r_m that belongs to $\text{Meet}_n(s_{n,j})$ reaches state $s_{n,j}$ at step j and remains in $s_{n,j}$ at step $j+1$. Note

that, compatible plans force the robots r_n and r_m to synchronously perform and action among steps j and $j + 1$.

A plan is *executed* by a robot, by performing the actions and transitions specified in the plan. Executing plans allows discovering information about action execution, service provisioning, and meeting capabilities. Every time a transition that is associated with partial information about its action is executed, an action associated with uncertainty in the service provisioning is performed and a state with unknown meeting capabilities is reached, new information is detected. If a transition is detected to be executable, a service to be provided, or a meeting to be possible, we say that a *true evidence* about the partial information is detected. In the opposite case, we say that *false evidence* is detected.

Given a *robot application* $\mathcal{H} = \{r_1, r_2, \dots, r_N\}$, such that $r_n = (S_n, init_n, A_n, \Pi_n, T_n, T_{p,n}, Meet_n, Meet_{p,n}, L_n)$ we define $\Pi = \Pi_1 \cup \Pi_2 \cup \dots \cup \Pi_N$. We assume that the mission assigned to the team of robots is made by a set $\Phi = \{\phi_1, \phi_2, \dots, \phi_N\}$ of local missions such that each mission $\phi_n \in \Phi$ is assigned to the robot r_n of \mathcal{H} . Each local mission ϕ_n is specified as an LTL formula defined over the set of atomic propositions $\Pi_{\phi,n} \subseteq \Pi$, i.e., the mission can also involve services that are not provided by robot r_n , but are provided by other robots of \mathcal{H} .

A dependency class $dp_n = r_1, r_2, \dots, r_n$ is a subset of robots that depends on each other for achieving their missions. Two robots $r_n = (S_n, init_n, A_n, \Pi_n, T_n, T_{p,n}, Meet_n, Meet_{p,n}, L_n)$ and $r_m = (S_m, init_m, A_m, \Pi_m, T_m, T_{p,m}, Meet_m, Meet_{p,m}, L_m)$ with missions defined over $\Pi_{\phi,n}$ and $\Pi_{\phi,m}$, are in dp_n if $\Pi_{\phi,n} \cap \Pi_m \neq \emptyset$, or there exists a state s of r_n such that $r_m \in Meet_n(s) \cup Meet_{p,n}(s)$. In the first case, one of the services of the mission of robot r_n is provided by robot r_m ; in the second, robot r_m must meet robot r_n .

The definitions of a definitive robot model, refinement and completion of partial robot models and execution of possible and definitive plans are in Appendix A1, available at <https://claudiomenghi.github.io/2018FMAappendix.pdf>.

4 Planning with partial knowledge

Solving the planning problem requires defining what it means that a plan satisfies a mission. This section provides two different definitions of this concept through the three-valued and the thorough LTL semantics. The first allows implementing an efficient procedure to check whether a plan satisfies a mission. However, the obtained result does not always reflect the natural intuition. The second allows returning a precise result but requires implementing a checking procedure that is more computationally expensive.

Consider for a moment a definitive (possible) plan $\beta = s_1\alpha_1s_2\alpha_2\dots$. The *definitive (possible) word* associated with β is $w_\tau = \varpi_1\varpi_2\dots$, such that for each $i > 1$ and $\pi \in \Pi$, $\varpi_i(\pi) = L_n(\alpha_i, \pi)$. Intuitively, each element ϖ_i of w_τ maps a service π with the value of function $L_n(\alpha_i, \pi)$. Note that definitive words associate services with values in the set $\{\top, \perp\}$, while possible words w_τ associate services with values in the set $\{\top, ?, \perp\}$.

The three-valued LTL semantics $\stackrel{3}{\models}$ allows efficiently checking a property ϕ and a word w_τ . This can be done at the same computational cost as regular two-valued semantics [5]. While a formula is evaluated as \top and \perp reflecting the natural intuition, it has been shown [6] that the three-valued semantics returns a $?$ value more often than expected. Specifically, it is desirable that a property ϕ is evaluated to $?$ when there are two words w'_τ and w''_τ that can be obtained by replacing $?$ values of w_τ with \top and \perp , such that w'_τ satisfies ϕ and w''_τ does not satisfy ϕ . For this reason, the thorough LTL semantics has been proposed [6].

The thorough LTL semantics is usually defined considering partial models [6]. Given a partial model M and a formula ϕ , the thorough LTL semantics assigns the value $?$ if there exist two completions M' and M'' of M , obtained by assigning \top and \perp to proposition with value $?$, such that M' satisfies ϕ and M'' violates ϕ . However, since our goal is to evaluate the satisfaction of a formula on plans we define a notion of thorough LTL semantics based on words. Given a word w_τ a completion w'_τ of w_τ is obtained from w_τ by assigning \top and \perp values to $\varpi_i(\pi)$ for each $i > 1$ and π such that $\varpi_i(\pi) = ?$

Definition 5. *Given a word w_τ and a property ϕ , the thorough LTL semantics $\stackrel{T}{\models}$ associates the word w_τ with a value in the set $\{\top, ?, \perp\}$ as follows*

- (1) $[w_\tau \models \phi] \stackrel{T}{=} \top$ iff all the completions w'_τ of w_τ are such that $[w'_\tau \models \phi] = \top$;
- (2) $[w_\tau \models \phi] \stackrel{T}{=} \perp$ iff all the completions w'_τ of w_τ are such that $[w'_\tau \models \phi] = \perp$;
- (3) $[w_\tau \models \phi] \stackrel{T}{=} ?$ otherwise.

Since a word is a simple linear model (a sequence of states connected by transitions), the properties that hold for the thorough LTL semantics for partial models also hold for the thorough LTL word semantics.

Lemma 1. *Given two words w_τ and w'_τ , such that $w_\tau \preceq w'_\tau$, and an LTL formula ϕ the following are satisfied: (1) $[w_\tau \models \phi] \stackrel{3}{=} \top \Rightarrow [w'_\tau \models \phi] \stackrel{T}{=} \top$;*
(2) $[w_\tau \models \phi] \stackrel{3}{=} \perp \Rightarrow [w'_\tau \models \phi] \stackrel{T}{=} \perp$.

Checking whether a word w_τ and a property ϕ in the sense of the thorough LTL word semantics levies performance penalties: it can be done in polynomial time in the length of w_τ and double exponential time in the size of ϕ [15].

There exists a subset of LTL formulae, known in literature as *self-minimizing* formulae, such that the three valued and the thorough semantics coincide [14]. This result can also be applied on words. Given a word w_τ and a self-minimizing LTL property ϕ , then $[w_\tau \models \phi] \stackrel{T}{=} x \stackrel{3}{=} [w_\tau \models \phi]$ where $x \in \{\perp, \top, ?\}$. It has been observed that most practically useful LTL formulae, such as absence, universality, existence, response and response chain, are self-minimizing [14]. Furthermore, since for this set the two semantics coincide, the more efficient procedure for checking a property w.r.t. the three-valued semantics can be used [6].

Local definitive and possible LTL satisfaction can be then defined as follows.

Definition 6. Let $\overset{X}{\equiv}$ be a semantics in $\{\overset{3}{\equiv}, \overset{T}{\equiv}\}$. Let r_n and D be a robot and a set of robots, such that $\{r_n\} \subseteq D \subseteq \mathcal{H}$. Let us consider a set of compatible plans \mathcal{B} such that each word w_m of robot r_m is infinite and defined as $w_m = \varpi_{m,1}\varpi_{m,2}\dots$. The word produced by a set of definitive (resp. possible) compatible plans $\mathfrak{B} = \{\beta_m \mid r_m \in D\}$ is $w_{\mathfrak{B}} = \omega_1\omega_2\dots$, where for all $j \geq 1$ and for all $\pi \in \Pi$, $\omega_j(\pi) = \max\{\varpi_{m,j}(\pi) \mid r_m \in D\}$. The set of definitive (resp. possible) compatible plans \mathfrak{B} locally definitively (resp. possibly) satisfies ϕ_n for the agent n , i.e., $\mathfrak{B} \models \phi_n$, iff \mathfrak{B} is valid and $[w_{\mathfrak{B}} \models \phi_n] \overset{X}{\equiv} \top$ ($[w_{\mathfrak{B}} \models \phi_n] \overset{X}{\equiv} ?$).

The thorough semantics ensures that when the word associated with the plan possibly satisfies a mission ϕ , there exists a way to assign \top and \perp to $?$ such that it is possible to obtain a word that satisfies ϕ and another that does not satisfy ϕ . Thus, there is a chance that, if executed, the plan satisfies a mission ϕ . This is not true for the three-valued semantics since it can be the case that a plan possibly satisfies a mission ϕ , but the mission is trivially unsatisfiable. Note that since a plan $w_{\mathfrak{B}}$ is such that $w_{\mathfrak{B}} \models \phi_n$, it can be rewritten using an ω -word notation as $w_{\mathfrak{B}} = \omega_1\omega_2\dots\omega_{n-1}(\omega_n\omega_{n+1}\dots\omega_m)^\omega$ [41], where m is the length of the plan. We use the notation $\mathcal{L}(w_{\mathfrak{B}})$ to indicate length m of the ω -word associated with plan $w_{\mathfrak{B}}$ with minimum length.

In this work we assume local missions are given. We also assume that all the local missions must be satisfied for achieving the global mission. Formally, given a global mission ϕ , the corresponding set of the local missions $\{\phi_1, \phi_2, \dots, \phi_N\}$ (one for each robot in \mathcal{H}) and the word \mathfrak{B} produced by a set of definitive (resp. possible) compatible plans, $[\mathfrak{B} \models \phi] \overset{X}{\equiv} \top$ if for all $1 \leq n \leq N$, $[\mathfrak{B} \models \phi_n] \overset{X}{\equiv} \top$; $[\mathfrak{B} \models \phi] \overset{X}{\equiv} ?$ if for all $1 \leq n \leq N$, $[\mathfrak{B} \models \phi_n] \overset{X}{\equiv} \perp$, where \geq is defined such as $\top > ? > \perp$; $[\mathfrak{B} \models \phi] \overset{X}{\equiv} \perp$ otherwise.

Based on these definitions, the planning problem is formulated as follows:

Problem 1 (Planning). Consider a partial robot application \mathcal{H} defined over the set of partial robot models $\{r'_1, r'_2, \dots, r'_N\}$ and a semantics $\overset{X}{\equiv}$ in $\{\overset{3}{\equiv}, \overset{T}{\equiv}\}$. Given a set of local missions $\{\phi_1, \phi_2, \dots, \phi_N\}$, one for each robot $r_n \in \mathcal{H}$ find a set of plans $\mathfrak{B} = \{\beta_1, \beta_2, \dots, \beta_N\}$ that

- (1) are compatible and
- (2) \mathfrak{B} locally definitively (resp. possibly) satisfies each ϕ_n w.r.t. the given semantics.

Appendix A2 contains the proof of Lemma 1 and additional theorems and proofs.

5 Algorithms

MAPmAKER solves Problem 1 by implementing a decentralized planning with partial knowledge. First, we discuss how robots are partitioned into dependency classes; then, we discuss how planning is performed within each of these classes.

Compute the dependency classes To compute dependency classes the following rule [16] is iteratively applied: a robot r_n assigned to a local mission

ϕ_n defined over the services $\Pi_{\phi,n}$ is in the same dependency class D_i of r_m if and only if (1) the local mission predicates on a service π provided by robot r_m , i.e., $\pi \in \Pi_{\phi,n} \cap \Pi_m$, or (2) r_n and r_m must meet.

In the running example, one dependency class contains robots r_1 and r_2 since these robots must meet in cell c_7 , the other contains robot r_3 .

Planning with partial knowledge Algorithm 1 receives a partial robot application and a set of local missions and computes a set of definitive (or possible) plans that ensures the mission satisfaction. We discuss how the different types of partial information are handled by Algorithm 1 by incrementally enabling Algorithm 1 to handle types of partial information. Identifiers A1, A2, and A3 mark lines that enable handling different types of partial information.

Managing partial information in the transition relation. Let us first assume that we have a partial robot application $\mathcal{H} = \{r_1, r_2, \dots, r_N\}$ where each robot r_n is such that $Meet_n = Meet_{p,n}$ and for each service $\pi \in \Pi_n$ and action $\alpha \in A_n$, $L_n(\alpha, \pi) \in \{\top, \perp\}$. In this case, partial information only refers to the presence of maybe transitions, i.e., transitions $s_n \xrightarrow{\alpha} s'_n$ such that $s_n \xrightarrow{\alpha} s'_n \notin T_n$. Lines marked with the identifier A1 allow Algorithm 1 to handle this case.

The algorithm works in three steps.

Step 1. For each robot $r_n \in \mathcal{H}$ it removes the transitions $s_n \xrightarrow{\alpha} s'_n$ such that $s_n \xrightarrow{\alpha} s'_n \notin T_n$ (Line 3) and applies a classical decentralized planning algorithm (Line 8). Variable pd contains whether the planner had found a plan, $\{pd_1, pd_2, \dots, pd_N\}$ contains the plans if they are found. If a plan is found, pd is assigned to *true* and the definitive plans for each of the robots are stored in variables $\{pd_1, pd_2, \dots, pd_N\}$. Otherwise, pd is assigned to *false*.

Step 2. It considers the original partial robot application (Line 9) and it applies the decentralized planning algorithm (Line 12). If a set of possible plans is found, pp is equal to *true* and the possible plans (one for each robot) are stored in $\{pp_1, pp_2, \dots, pp_N\}$.

Step 3. It analyzes the results contained in pd and pp . If both pd and pp are equal to *false* then no plans are synthesizable (Line 13). If pd is *false* while pp is not, only possible plans are available and they are returned as output (Line 14). Otherwise, a policy is used to choose between $\{pd_1, pd_2, \dots, pd_N\}$ and $\{pp_1, pp_2, \dots, pp_N\}$ (Line 15).

If no meeting primitives are specified in cell c_9 , actions ld and rd cannot be performed by robots r_1 and r_2 in c_9 and ud_2 does not provide service *unload*, plans p_1 , p_2 , and p_3 are returned from robots r_1 , r_2 , and r_3 . Plan p_2 is possible since it is not known whether robot r_1 can move from cell c_8 to c_{14} . Plan p_3 is possible since it is unknown whether robot r_3 can perform action sp in cell c_{18} .

Managing uncertainties in the service provision. Let us assume that we have a partial robot application \mathcal{H} where each robot r_n is such that $Meet_n = Meet_{p,n}$, i.e., there is no partial information about meeting capabilities. We designed an algorithm similar to [5], specified by Lines marked with the identifier A2.

Step 1. For each r_n , remove all the transitions $s_n \xrightarrow{\alpha} s'_n$ such that $s_n \xrightarrow{\alpha} s'_n \notin T_n$ (Line 3).

Step 2. Put each formula $\phi_n \in \Phi$ in its negation normal form (Line 4).

Algorithm 1 The *PARTIAL PLAN* function.

- 1: **Input** a partial robot application \mathcal{H} and a set of missions Φ
 - 2: **Output** a definitive or a possible plan (if they exist)
 - 3: **For each** robot $r_n \in \mathcal{H}$ **remove** transitions $s_n \xrightarrow{\alpha} s'_n$ s.t. $s_n \xrightarrow{\alpha} s'_n \notin T_n$ (**A1**)
 - 4: **Put** each formula $\phi_n \in \Phi$ in its negation normal form and rename the negated propositions as in [5] (**A2**)
 - 5: **Compute** the complement closed model r'_n of each $r_n \in \mathcal{H}$ (**A2**)
 - 6: **For each** r'_n and $s \in S_n$ **if** $Meet_{n,p}(s) \neq Meet_n(s)$ **then** $Meet_{n,p}(s) = \{\#\}$ (**A3**)
 - 7: **For each** model r'_n **construct** the pessimistic approximation $r'_{p,n}$ (**A2**)
 - 8: $[pd, \{pd_1, pd_2, \dots, pd_N\}] = \text{DEC_PLANNER}(\{r'_{p,1}, r'_{p,2}, \dots, r'_{p,N}\}, \Phi)$
 - 9: **For each** robot r_n **insert** the transitions $s_n \xrightarrow{\alpha} s'_n$ s.t. $s_n \xrightarrow{\alpha} s'_n \notin T_n$ (**A1**)
 - 10: **For each** model r'_n **construct** the optimistic approximation $r'_{o,n}$ (**A2**)
 - 11: **For each** r'_n **insert** all the meeting requests in $Meet_{n,p}$ (**A3**)
 - 12: $[pp, \{pp_1, pp_2, \dots, pp_N\}] = \text{DEC_PLANNER}(\{r'_{o,1}, r'_{o,2}, \dots, r'_{o,N}\}, \Phi)$
 - 13: **if** $pd = \text{false}$ **and** $pp = \text{false}$ **then return** NO_PLAN_AVAILABLE
 - 14: **if** $pd = \text{false}$ **then return** $\{pp_1, pp_2, \dots, pp_N\}$;
 - 15: **else return** $\text{choose}(\{pd_1, pd_2, \dots, pd_N\}, \{pp_1, pp_2, \dots, pp_N\})$
-

Step 3. For each r_n , construct a model r'_n called complement-closed, in which for action $\alpha \in A$ and service $\pi \in AP$, there exists a new service $\bar{\pi}$, called complement-closed service, such that $L_n(\alpha, \bar{\pi}) = \text{comp}(L_n(\alpha, \pi))$ (Line 5).

Step 4. Substitute function L_n of each r'_n with its pessimistic approximation $L_{n,pes}$. Specifically, $L_{n,pes}$ is constructed as follows: for each action α and π such that $L_n(\alpha, \pi) = ?$, $L_{n,pes}(\alpha, \pi) = \perp$ otherwise $L_{n,pes}(\alpha, \pi) = L_n(\alpha, \pi)$ (Line 7).

Step 5. Apply the decentralized planning algorithm (Line 8). If a set of plans is found they are definitive plans and stored in $\{pd_1, pd_2, \dots, pd_N\}$ otherwise a *false* value is associated to pd .

Step 6. For each r_n insert all the transitions $s_n \xrightarrow{\alpha} s'_n$ such that $s_n \xrightarrow{\alpha} s'_n \notin T_n$ (Line 9).

Step 7. Construct the optimistic approximation function $L_{n,opt}$ by associating the value \top to each atomic proposition of the complement-closure of r' with value $?$ (Line 10).

Step 8. Apply the decentralized planning algorithm (Line 12). If a set of plans are found they are possible plans and stored in $\{pp_1, pp_2, \dots, pp_N\}$ otherwise a *false* value is associated to pp .

Step 9. Analyzes the results. If both pd and pp are assigned with the *false* value (Line 13), neither a possible nor a definitive plan can be synthesized. If pd is *false* while pp is not (Line 14), then no definitive plans are available while a possible plan has been found. Otherwise (Line 15), an appropriate policy is used to choose between $\{pp_1, pp_2, \dots, pp_N\}$ and $\{pd_1, pd_2, \dots, pd_N\}$.

If no meeting is required in cell c_9 , actions ld and rd cannot be performed by robots r_1 and r_2 in c_9 and it is not known whether ud_2 provides service *unload*, plans p_1 , p'_2 , and p_3 are returned for r_1 , r_2 , and r_3 . Plan p'_2 is possible since it is unknown whether the execution of action ud_2 provides the service *unload*.

Managing uncertainties in the meeting capabilities. Partial knowledge in the meeting capabilities is handled considering lines marked with A3. These lines

ensure that before searching for a definitive plan, the meeting requests that are possible in the partial model of each robot of the robot application are removed (Line 6). These requests are added (Line 11) before searching for possible plans.

If meeting in cell c_{21} is considered, and actions ld and rd can be performed by robots r_1 and r_2 in c_9 , plans p_1'' , p_2'' , and p_3 are returned from robot r_1 , r_2 , and r_3 , respectively. Plans p_1'' and p_2'' are shorter plans than p_2' and p_1 .

Algorithm 1 calls a classical decentralized planning algorithm twice. This algorithm is also re-executed every time during that the plan execution a *false* evidence about partial information is detected.

Theorem 1. *Consider a partial robot application \mathcal{H} , a set of missions Φ (one for each robot) and the three-valued LTL semantics ($\stackrel{3}{=}$). A set of plans \mathfrak{B} that (1) are compatible and (2) \mathfrak{B} locally definitively (resp. possibly) satisfies each ϕ_n w.r.t. the three-valued semantics. is returned from Algorithm 1 if and only if they exist in \mathcal{H} . If formulae are self-minimizing this theorem also applies to the thorough LTL semantics ($\stackrel{T}{=}$).*

Proof of Theorem 1 and additional details of the algorithm are in Appendix A3.

6 Evaluation

This section reports on our experience evaluating MAPmAKER. We considered the following research questions: **RQ1:** *How does MAPmAKER help planning in partially known environments?* **RQ2:** *How does the employed decentralized algorithm help in planning computation?*

Implementation. As a proof of concepts we implemented MAPmAKER as a Matlab application, based on the planner proposed in [38]. The source code, a complete replication package and a set of videos showing MAPmAKER in action can be found at <https://github.com/claudiomenghi/MAPmAKER/>.

RQ1. We analyzed MAPmAKER on a set of *simulated* models.

Methodology. We considered two existing examples proposed in literature.

Example 1. The model of the robot application of the RoboCup Logistics League competition [18] which has a map made by 169 cells and 4 rooms.

Example 2. The model of a robot application deployed in an apartment of about 80 m^2 , which is part of a large residential facility for senior citizens [1]. The map had originally been used to evaluate a planning algorithm for a single robot based on information contained in RFID tags present in the environment [19].

In both the examples, we considered a team of 2 robots (r_1 and r_2), which is the same number of robots used in the RoboCup competition. However, we considered a higher number of services. Specifically, we considered 5 services: services s_1, s_2 , and s_3 for robot r_1 and services s_4 and s_5 for robot r_2 .

We simulated the presence of partial knowledge about the robot application to evaluate the impact of partial information about the execution of transitions (*Exp 1*), services provisioning (*Exp 2*) and meeting capabilities (*Exp 3*) on the planning procedure. To simulate the presence of partial information we

constructed a partial robot application that conforms with Definition 1. To analyze partial information in the execution of transitions (*Exp 1*) we considered 2 rooms (that had multiple exits) and for each of these we added two unknown transitions. Both of these transitions allow leaving the room and are placed in correspondence with an exit and a wall. Thus, they will turn into a *true* and *false* evidence about the partial information when reached by the robot, respectively. To simulate partial information about the service provisioning (*Exp 2*) we assumed that service s_2 is associated with actions a_1 and a_2 . However, there is partial information on whether the execution of a_1 and a_2 actually provides service s_2 . In one case, when action a_1 is executed a *true* evidence on the provision of s_2 is returned. Contrariwise, when action a_2 is executed, a *false* evidence is returned by the dynamic discovering procedure. To simulate partial information about the meeting capabilities (*Exp 3*) we assumed that it is unknown whether robots r_1 and r_2 can meet in two cells when service s_1 is provided. In one of the cells, when meeting is performed, a *true* evidence is returned while in the other case *false* evidence is returned.

We consider different missions since in the RoboCup competition each mission was supposed to be performed in isolation by a single robot, while we aim to evaluate the behavior of the overall team. Our missions were inspired by the one used in the RoboCup competition and formalized as self-minimizing LTL properties and based on well known properties patterns [21],[43]. The following missions were considered: (1) robot r_1 must achieve the mission $F(s_1 \wedge (F(s_2 \vee s_3)))$. It had to reach a predefined destination where service s_1 is provided, and then perform either service s_2 or service s_3 . (2) robot r_2 must achieve the following mission $G(F(s_4 \vee s_5))$. Furthermore, it aims at helping r_1 in providing service s_1 , i.e., robots r_1 and r_2 must meet in cells where service s_1 is provided.

Our simulation scenarios were obtained by considering different initial conditions (indicated as I_1 , I_2 , and I_3), where robots were initially located in different cells, and models of the partial robot application (indicated as C_1 , C_2 , and C_3), obtained by making different choices about partial information. Each simulation scenario is associated with an identifier (*ID*) and is obtained by considering a model of the partial robot application in one of the initial conditions.

Then, we performed the two following steps.

Step 1. We run MAPmAKER by considering the partial model of the robot application. The algorithm iteratively computes possible plans that are executed by the robots. As the robots explore their environment, *true* or *false* evidence about partial information is detected meaning that a transition, service, and meeting capability is detected to be frable, provided, and possible, respectively. If a *false* evidence about a partial information is detected, e.g., a transition of the plan is not executable, MAPmAKER is re-executed to recompute a new possible plan. As all the partial information needed to achieve the mission is turned into a *true* or a *false* evidence, the produced plan is actually *definitive*.

Step 2. We run MAPmAKER on a model of the robotic application obtained by assuming that unknown transitions, services, and meeting capabilities are not executable, not provided, and not possible, respectively. Thus, MAPmAKER

Table 1. Results of Experiments 1, 2, and 3 for Examples 1 and 2.

		Example 1												Example 2																
		Exp 1				Exp 2				Exp 3				Exp 1				Exp 2				Exp 3								
<i>ID</i>	<i>I</i>	<i>C</i>	<i>F</i>	<i>T</i>	\mathcal{T}_r	\mathcal{L}_r	<i>F</i>	<i>T</i>	\mathcal{T}_r	\mathcal{L}_r	<i>F</i>	<i>T</i>	\mathcal{T}_r	\mathcal{L}_r	<i>F</i>	<i>T</i>	\mathcal{T}_r	\mathcal{L}_r	<i>F</i>	<i>T</i>	\mathcal{T}_r	\mathcal{L}_r	<i>F</i>	<i>T</i>	\mathcal{T}_r	\mathcal{L}_r	<i>F</i>	<i>T</i>	\mathcal{T}_r	\mathcal{L}_r
1	I_1	C_1	1	1	4.9	1.3	0	1	1.4	0.6	0	0	2.3	1.0	0	1	-	-	1	1	3.2	1.1	1	0	3.5	1.4				
2	I_1	C_2	0	1	3.6	0.9	0	0	2.1	1.0	0	1	2.3	0.7	0	0	3.4	1.0	0	0	2.0	1.0	0	0	2.0	1.0				
3	I_1	C_3	1	1	5.2	1.3	1	1	3.3	1.1	1	0	4.0	1.4	1	0	6.3	1.9	0	1	1.2	0.6	0	0	2.1	1.0				
4	I_2	C_2	1	1	5.2	1.4	0	1	2.0	0.9	0	0	2.1	1.0	0	2	-	-	0	1	1.8	0.8	1	0	3.0	1.0				
5	I_2	C_2	1	2	-	-	0	1	1.5	0.6	0	1	1.8	0.9	0	1	-	-	0	0	2.0	1.0	0	0	2.0	1.0				
6	I_2	C_2	1	2	7.6	1.1	0	1	1.9	0.9	1	1	3.1	0.8	1	0	3.0	1.2	1	1	3.8	1.3	0	0	2.0	1.0				
7	I_3	C_3	0	0	3.9	1.0	0	0	2.0	1.0	0	0	2.0	1.0	0	1	-	-	1	0	3.6	1.6	0	0	2.0	1.0				
8	I_3	C_3	0	1	-	-	1	0	2.1	1.0	0	1	1.8	0.8	0	2	-	-	0	1	1.8	0.8	0	1	1.5	0.8				
9	I_3	C_3	0	0	3.5	1.0	0	0	1.9	1.0	1	1	3.4	0.9	0	1	1.9	0.8	0	1	1.9	0.9	0	1	1.5	0.8				

returns a *definitive* plan (if present). This model is not the real model of the robot application since some transitions, services, and meeting capabilities may be turned into not firable, not provided and not possible, when they can actually be fired, are provided, and are possible in the real model, respectively.

We measured (1) the *time* \mathcal{T}_1 and \mathcal{T}_2 spent by MAPmAKER in Steps 1 and 2 in computing possible and definitive plans. For Step 1 it also includes the time necessary for synthesizing new plans where a *false* evidence about a partial information is detected. For Step 2 it only includes the time spent by MAPmAKER in computing the definitive plan. (2) the *length* \mathcal{L}_1 and \mathcal{L}_2 of the plans computed by MAPmAKER, in Steps 1 and 2. For Step 1 it is obtained by computing the sum of the length of the portions of the possible plans performed before a *false* evidence about a partial information is detected and the length of the final definitive plan (more details are provided in Appendix A4). For Step 2 it corresponds to the length of the definitive plan. We compared the time spent by MAPmAKER in Steps 1 and 2 and the length of the computed plans.

Results. Table 1 shows the obtained results. Column \mathcal{T}_r contains the ratio between \mathcal{T}_1 and \mathcal{T}_2 , column \mathcal{L}_r contains the ratio between \mathcal{L}_1 and \mathcal{L}_2 . Columns *F* and *T* contain the number of times *true* or *false* evidence about partial information about a transition, service and meeting capability was detected while the plans were executed.

Example 1. Four cases are identified. In ID 2 for Exp 1, IDs 1, 4, 5, 6 for Exp 2 and IDs 2, 5, 6, 8, 9 for Exp 3 the plans computed in Step 1 were shorter than the one computed in Step 2 (Case 1). In IDs 1, 3, 4, 6 for Exp 1, IDs 3 for Exp 2 and ID 3 for Exp 3 the plans computed in Step 1 were longer than the one computed in Step 2 (Case 2). In IDs 7, 9 for Exp 1, IDs 2, 7, 8 for Exp 2 and IDs 1, 4, 7, 9 for Exp 3 the plans computed in Step 1 correspond with the one computed in Step 2 (Case 3). In IDs 5, 8 for Exp 1 plans were found in Step 1, while no plans were obtained in Step 2 (Case 4). Thus, they are marked with a – since no comparison was possible.

Example 2. In ID 9 for Exp 1, IDs 3, 4, 8, 9 for Exp 2 and IDs 8, 9 for Exp 3 the plans computed in Step 1 were shorter than the one computed in Step 2

(Case 1). In IDs 3, 6 for Exp 1, IDs 1, 6, 7 for Exp 2 and IDs 1 for Exp 3 the plans computed in Step 1 were longer than the one computed in Step 2 (Case 2). In ID 2 for Exp 1, IDs 2, 5 for Exp 2 and IDs 2, 3, 4, 5, 6, 7 for Exp 3 the plans computed in Step 1 correspond with the one computed in Step 2 (Case 3). In IDs 1, 4, 5, 7, 8 for Exp 1 plans were found in Step 1, while no plans were obtained in Step 2 (Case 4).

Discussion. MAPmAKER is effective whenever it computes a possible plan, and during its execution a *true* evidence about partial information is detected (Case 1). When no partial information was involved in the plans computed by MAPmAKER, the generated plans had the same length than a classical planner (Case 3). In several configurations MAPmAKER allows the achievement of the mission while a classical procedure is not able to do so (Case 4). Indeed, MAPmAKER computes a possible plan when no definitive plan is available and *true* evidence about partial information is detected during the plan execution. Finally, the detection of a *false* evidence decreases the effectiveness of MAPmAKER (Case 2). It happens due to the need of recomputing the plans to be followed by the robots.

MAPmAKER introduced an overhead in plan computation since it runs two times the decentralized planner. The average, median, minimum, and maximum time required to compute the plans for Step 1 considering all the examples of the previous experiments are 1982.28, 2371.38, 990.76, and 2972.64 seconds respectively; while for Step 2 are 400.24, 387.34, 277.85 and 533.8 seconds respectively. The high computation time is due to the planner on top of which MAPmAKER is developed, which uses an explicit representation of the state space of the robotic application. However, MAPmAKER simply relies on two invocations of a general planner to compute plans, thus more efficient planners can be used.

RQ2. We analyzed the behavior of the decentralized procedure.

Methodology and experimental inputs. We considered the set of partial models previously described. We added an additional robot r_3 which must achieve the mission $\mathbb{G}(F(s_6 \vee s_7))$ and does not meet neither with robot r_1 nor with robot r_2 . We then perform the following steps: Step 1 we run MAPmAKER with the decentralized procedure enabled; Step 2 we run MAPmAKER without the decentralized procedure enabled. For each of the steps, we set a timeout of 1 hour. We recorded the time \mathcal{T}_1 and \mathcal{T}_2 required in Steps 1 and 2.

Results and discussion. In Step 1 MAPmAKER computes two dependency classes; one containing robots r_1 and r_2 and one containing robot r_3 . In Step 2 the team containing robots r_1 , r_2 , and r_3 is analyzed. For all the configurations and experiments, MAPmAKER ends within the timeout for Step 1, while MAPmAKER was not able to find a solution for Step 2.

Threats to validity. The random identification of elements that are considered uncertain is a threat to *construct validity* since it may generate not realistic models. To mitigate this threat we ensured that partial information about transitions is added in correspondence with an exit and a wall. This ensures that both true and false evidence for transition executions can occur while the computed plans are executed. Biases in the creation of models is a threat to *internal*

validity and is mitigated by considering real models. The limited number of examples is a threat to *external validity*. To mitigate this threat, we verified that as possible plans were executed, both true and false evidence about partial information were detected.

7 Related work

Decentralized solutions. The decentralized planning problem has been studied for known environments [36],[16],[38]. However, planners for partially known environments do not usually employ decentralized solutions [35],[12],[10].

Dealing with partial knowledge in planning. Most of the works proposed in literature to plan in partially known environments (see for example [11],[22],[42],[13]) treat partial information by modeling the robotic application using some form of Markov decision processes (MDP). In MDPs, transitions are associated with probabilities indicating the likelihood of reaching the destination state when an action is performed [13]. The planning problem usually requires the actions the robots must perform to reach a set of goal states. In our work, the planning goal is specified in a richer language, i.e., LTL. Planning with LTL specifications has been considered in MDPs (e.g., [23],[11],[25]). However, in MDPs the developer knows the probabilities associated with transitions, while in the formulation proposed in this work this information is not available. Encoding a partial robot model into a MDP by associating a probability of 0.5 to maybe transitions is not correct. Indeed, the obtained MDP would not correctly represent the current scenario in which the probability of firing transitions is unknown.

8 Conclusions

This work presented MAPmAKER, a novel decentralized planner for partially known environments. MAPmAKER solves the decentralized planning problem when partial robot applications made by multiple robots are analyzed and missions are provided through a set of LTL specifications that are assigned to the different robots. The results showed that MAPmAKER was effective in dealing with partially known environments. They evidenced that the number of actions performed by the robots was lower when the computed possible plans were actually executable in the real model of the robotic application. Furthermore, they highlight that MAPmAKER outperformed classical planners by achieving the desired mission when only possible plans were available. Finally, they show that decentralization allows considering partial models of the robotic applications that can not be handled with a classical centralized approaches.

Future work and research directions include (1) studying techniques to support developers in the automatic or manual development of the (partial) model of a robotic application; (2) evaluation of the proposed procedure using robots deployed in real environments; (3) the study of appropriate policies to select between definitive and possible plans; (4) the use of more efficient planners to speed up plan computation. These may be based for example on symbolic techniques.

References

1. The angen research and innovation apartment: official website (2014), <http://angeninnovation.se>
2. Bernasconi, A., Menghi, C., Spoletini, P., Zuck, L.D., Ghezzi, C.: From Model Checking to a Temporal Proof for Partial Models, pp. 54–69. Springer (2017)
3. Bhatia, A., Kavvaki, L.E., Vardi, M.Y.: Motion planning with hybrid dynamics and temporal goals. In: Conference on Decision and Control (CDC). pp. 1108–1115. IEEE (2010)
4. Bhatia, A., Kavvaki, L.E., Vardi, M.Y.: Sampling-based motion planning with temporal goals. In: International Conference on Robotics and Automation (ICRA). pp. 2689–2696. IEEE (2010)
5. Bruns, G., Godefroid, P.: Model checking partial state spaces with 3-valued temporal logics. In: Computer Aided Verification. pp. 274–287. Springer (1999)
6. Bruns, G., Godefroid, P.: Generalized model checking: Reasoning about partial state spaces. In: International Conference on Concurrency Theory. pp. 168–182. Springer (2000)
7. Chechik, M., Devereux, B., Easterbrook, S., Gurfinkel, A.: Multi-valued symbolic model-checking. *ACM Transactions on Software Engineering and Methodology* 12(4), 1–38 (2004)
8. Chen, Y., Tůmová, J., Ulusoy, A., Belta, C.: Temporal logic robot control based on automata learning of environmental dynamics. *The International Journal of Robotics Research* 32(5), 547–565 (2013)
9. Cunningham, A.G., Galceran, E., Eustice, R.M., Olson, E.: Mpdm: Multipolicy decision-making in dynamic, uncertain environments for autonomous driving. In: International Conference on Robotics and Automation (ICRA). pp. 1670–1677 (2015)
10. Diaz, J.F., Stoytchev, A., Arkin, R.C.: Exploring unknown structured environments. In: FLAIRS Conference. pp. 145–149. AAAI Press (2001)
11. Ding, X.C.D., Smith, S.L., Belta, C., Rus, D.: LTL control in uncertain environments with probabilistic satisfaction guarantees*. *IFAC Proceedings Volumes* 44(1), 3515 – 3520 (2011)
12. Du Toit, N.E., Burdick, J.W.: Robot motion planning in dynamic, uncertain environments. *IEEE Transactions on Robotics* 28(1), 101–115 (2012)
13. Ghallab, M., Nau, D., Traverso, P.: *Automated Planning and Acting*. Cambridge University Press, New York, NY, USA, 1st edn. (2016)
14. Godefroid, P., Huth, M.: Model checking vs. generalized model checking: semantic minimizations for temporal logics. In: *Logic in Computer Science*. pp. 158–167. IEEE Computer Society (2005)
15. Godefroid, P., Piterman, N.: LTL generalized model checking revisited. *International journal on software tools for technology transfer* 13(6), 571–584 (2011)
16. Guo, M., Dimarogonas, D.V.: Multi-agent plan reconfiguration under local LTL specifications. *The International Journal of Robotics Research* 34(2), 218–235 (2015)
17. Guo, M., Johansson, K.H., Dimarogonas, D.V.: Revising motion planning under linear temporal logic specifications in partially known workspaces. In: International Conference on Robotics and Automation (ICRA). pp. 5025–5032. IEEE (2013)
18. Karras, C.D.U., Neumann, T., Rohr, T.N.A., Uemura, W., Ewert, D., Harder, N., Jentzsch, S., Meier, N., Reuter, S.: Robocup logistics league rules and regulations (2016)

19. Khaliq, A.A., Saffiotti, A.: Stigmergy at work: Planning and navigation for a service robot on an rfid floor. In: International Conference on Robotics and Automation (ICRA). pp. 1085–1092. IEEE (2015)
20. Kloetzer, M., Ding, X.C., Belta, C.: Multi-robot deployment from LTL specifications with reduced communication. In: Conference on Decision and Control and European Control Conference (CDC-ECC). pp. 4867–4872. IEEE (2011)
21. Kress-Gazit, H., Fainekos, G.E., Pappas, G.J.: Temporal-logic-based reactive mission and motion planning. *IEEE transactions on robotics* 25(6), 1370–1381 (2009)
22. Kurniawati, H., Du, Y., Hsu, D., Lee, W.S.: Motion planning under uncertainty for robotic tasks with long time horizons. *The International Journal of Robotics Research* 30(3), 308–323 (2011)
23. Lacerda, B., Parker, D., Hawes, N.: Optimal and dynamic planning for markov decision processes with co-safe ltl specifications. In: 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems. pp. 1511–1516 (2014)
24. Larsen, K.G., Thomsen, B.: A modal process logic. In: *Logic in Computer Science*. pp. 203–210. IEEE (1988)
25. Lassaigne, R., Peyronnet, S.: Approximate planning and verification for large markov decision processes. *International Journal on Software Tools for Technology Transfer* 17(4), 457–467 (2015)
26. Latombe, J.C.: *Robot motion planning*, vol. 124. Springer (2012)
27. Letier, E., Kramer, J., Magee, J., Uchitel, S.: Deriving event-based transition systems from goal-oriented requirements models. vol. 15, pp. 175–206. Springer (2008)
28. Loizou, S.G., Kyriakopoulos, K.J.: Automated planning of motion tasks for multi-robot systems. In: Conference on Decision and Control and European Control Conference (CDC-ECC). pp. 78–83. IEEE (2005)
29. Menghi, C., Spoletini, P., Chechik, M., Ghezzi, C.: Supporting verification-driven incremental distributed design of components. In: *Fundamental Approaches to Software Engineering*. pp. 169–188. Springer (2018)
30. Menghi, C., Spoletini, P., Ghezzi, C.: Dealing with incompleteness in automata-based model checking. In: *Formal Methods*. vol. 9995, pp. 531–550. Springer (2016)
31. Menghi, C., Spoletini, P., Ghezzi, C.: Cover: Change-based goal verifier and reasoner. In: *REFSQ Workshops*. Springer (2017)
32. Menghi, C., Spoletini, P., Ghezzi, C.: Integrating goal model analysis with iterative design. In: *International Working Conference on Requirements Engineering: Foundation for Software Quality*. pp. 112–128. Springer (2017)
33. Menghi, C., Tsigkanos, C., Berger, T., Pelliccione, P., Ghezzi, C.: Poster: Property specification patterns for robotic missions. In: (To appear) *International Conference on Software Engineering (ICSE)*. Poster Track (2018)
34. Quottrup, M.M., Bak, T., Zamanabadi, R.: Multi-robot planning: A timed automata approach. In: *International Conference on Robotics and Automation*. vol. 5, pp. 4417–4422. IEEE (2004)
35. Roy, N., Gordon, G., Thrun, S.: Planning under uncertainty for reliable health care robotics. In: *Field and Service Robotics*. pp. 417–426. Springer (2006)
36. Schillinger, P., Bürger, M., Dimarogonas, D.: Decomposition of finite LTL specifications for efficient multi-agent planning. In: *International Symposium on Distributed Autonomous Robotic Systems* (2016)
37. Tsigkanos, C., Pasquale, L., Menghi, C., Ghezzi, C., Nuseibeh, B.: Engineering topology aware adaptive security: Preventing requirements violations at runtime. In: *International Requirements Engineering Conference (RE)*. pp. 203–212 (2014)
38. Tumova, J., Dimarogonas, D.V.: Multi-agent planning under local LTL specifications and event-based synchronization. *Automatica* 70, 239–248 (2016)

39. Uchitel, S., Alrajeh, D., Ben-David, S., Braberman, V., Chechik, M., De Caso, G., D'Ippolito, N., Fischbein, D., Garbervetsky, D., Kramer, J., et al.: Supporting incremental behaviour model elaboration. *Computer Science-Research and Development* 28(4), 279–293 (2013)
40. Uchitel, S., Brunet, G., Chechik, M.: Synthesis of partial behavior models from properties and scenarios. *IEEE Transactions on Software Engineering* 35(3), 384–406 (2009)
41. Vardi, M., Wolper, P.: Reasoning about infinite computations. *Information and Computation* 115(1), 1 – 37 (1994)
42. Wolff, E.M., Topcu, U., Murray, R.M.: Robust control of uncertain markov decision processes with temporal logic specifications. In: *Annual Conference on Decision and Control (CDC)*. pp. 3372–3379. IEEE (2012)
43. Yoo, C., Fitch, R., Sukkariéh, S.: Online task planning and control for fuel-constrained aerial robots in wind fields. *The International Journal of Robotics Research* 35(5), 438–453 (2016)