Multi-Robot LTL Planning Under Uncertainty

Anonymous Author(s)

ABSTRACT

Robot applications are increasingly based on *teams* of autonomous robots that collaborate and interact to perform a desired mission. Such applications ask for decentralized techniques that allow for tractable automated planning. Another aspect that state-of-the-art robot applications must consider is *partial knowledge* about the environment in which the robots are operating and the associated uncertainty with the outcome of the robots' actions. This occurs, for example, when robots navigate in environments affected by natural disasters, where the movement between locations may be impossible due to structural collapses, flooding, etc.

Current planning techniques used for teams of robots that should perform complex missions do not systematically address these challenges: they are either based on centralized solutions and hence not scalable, they consider rather simple missions, such as A-to-B travel, or do not work in partially known environments. We present a planning solution that decomposes the team of robots into subclasses, considers complex high-level 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.

ACM Reference format:
Anonymous Author(s). 2018. Multi-Robot LTL Planning Under Uncertainty. In *Proceedings of 40th International Conference on Software Engineering, Gothenburg, Sweden, May 27–June 3, 2018 (ICSE 2018),* 13 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Helping designers to engineer robotic systems is one of the timely application areas of software engineering. Designing robotic systems requires solving numerous problems [38], such as the selection of the types of the robot to be employed (e.g., humanoid, robotic arm), the analysis of the required robot properties (e.g., emergence, emotional), the analysis of the place in which the robot operates (e.g., on the bus, in a birthday party), the activity the robot has to perform (e.g., move object, reach a location), and the users it has to serve (e.g., taxi driver, rock star). These aspects are then used by designers in the selection of appropriate planners.

A *planner* is a software component that receives as input a model of the robotic application and computes a set of actions (a *plan*) that, if performed, allows the achievement of a desired mission [34]. As done in some recent works in the robotics community (see for example [4, 5, 19, 20, 28, 30, 60]), in this work we assume that a robot application is defined using finite transition systems and each robot of the team has to achieve a mission, indicated as *local mission*, that is 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 problem formulations in robot task planning, LTL allows us to specify a rich class of temporal goals that include e.g., surveillance, sequencing, safety, or reachability. Several works studied centralized planners that are able to manage *teams* of robots that collaborate to achieve a certain goal (a global mission) [29, 39, 49]. Others studied how to decompose a global mission into a set of local missions to be achieved by each robot of the team [24, 24, 52, 55]. These local missions have been recently exploited by *decentralized* planners [55], i.e., planners that instead of evaluating the global mission over the whole team of robots, analyze the satisfaction of local missions inside a subset of the team of robots. In this way, the problem of finding a collective team behavior is decomposed into sub-problems that avoid the expensive fully centralized planning.

Another aspect that current planners must consider is partial knowledge about the environment in which the robots should operate. Partial knowledge in software development has been strongly studied by the software engineering community. For example, partial models have been used to support requirement analysis and elicitation [36, 42, 43], to help designers in producing a model of the system that satisfies a set of desired properties [2, 21, 56, 57] and to verify whether already designed models possess some properties of interest [7, 10, 41]. However, most of the existing planners assume that the environment in which the robots are deployed is known [14]. This assumption does not usually hold in real word scenarios [33]. In real world applications it is usually the case that only *partial knowledge* about the environment in which the robots are operating is present. This occurs, for example, when the robots navigate in environments affected by natural disasters, where the movement between locations or the execution of specific actions may be impossible due to structural collapses, flooding etc. Several works studied planners that work when only partial information about the environment in which the robots operate is available (e.g., [15, 18, 50]). However, literature considering *decentralized* planners with only partial knowledge about the robot application and temporal logic goals is rather limited [24].

Contribution. This work presents MAPmAKER (Multi-robot plAnner for PArtially Known EnviRonments), a *novel decentralized* planner for partially known environments. 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. A *definitive plan* is a sequence of actions that ensure the satisfaction of the local mission for each robot. A *possible plan* is a sequence of actions that may satisfy the local mission due to some unknown information about the

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ICSE 2018, Gothenburg, Sweden
© 2018 ACM. 978-x-xxxx-xxxx-x/YY/MM...$15.00
DOI: 10.1145/nnnnnnn.nnnnnnn

model of the robots or the environment in which they are deployed. MAPmAKER chooses the plan that allows the achievement of the mission by performing the lower number of actions, but other policies can also be used.

Specific contributions. Specific contributions are detailed in the following: (1) we define the concept of *partial robot model*, which allows the description of the behavior of the robots and its environment when only partial information is available. Specifically, a partial robot model allows considering three types of partial information: partial knowledge about the execution of transitions (possibility of changing the robot location), on service provision (whether the execution of an action succeed in providing a service) and on the meeting capabilities (whether a robot can meet with another); (2) we define the concept of local mission satisfaction for partial robot models; (3) we define the distributed planning problem for partially specified robots; (4) we propose a distributed planning algorithm and we proved its correctness; (5) we evaluate the proposed algorithm on a robot application obtained from the RobotCup Logistics League competition [26] and on a robotic application working in an apartment of about 80 m^2, which is part of a large residential facility for senior citizens [53]. The results show the effectiveness of the proposed algorithm.

Organization. Section 2 presents our running example. Section 3 describes the problem and Section 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

A set $R = \{r_1, r_2, r_3\}$ of robots is deployed in the environment graphically described in Fig. 1. This environment represents a building made by four rooms $L = \{l_1, l_2, l_3, l_4\}$, which has been affected by an earthquake. The environment is further partitioned in cells, each labeled with an identifier in $c_1, c_2, \ldots, c_{30}$. 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*. In Fig. 1 the cells in which a robot r can perform an action 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 *ud*1 and *ud*2. Actions *ud*1 and *ud*2 use different actuators. Specifically, action *ud*1 uses a gripper while action *ud*2 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 $r_3(sp)$ are used in Fig. 1 to mark the regions where actions can be executed by the robots, while movement actions are not reported for graphical reasons. Each action may be associated with a service, which is a high-level functionality provided by the robot when an action is performed. For example, actions *ld*, *rd*, *tp*, and *sp* are associated with the services *load_carrier*, *detect_load*, *take_snapshot*, and *send_info*, respectively. Actions *ud*1 and *ud*2 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

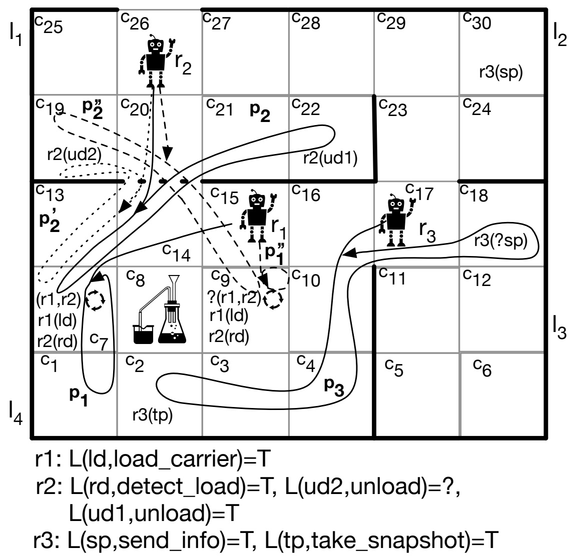

Figure 1: An example showing the model of the robots and their environment. Plans computed by MAPmAKER are represented by trajectories marked with arrows.

α. Robots must meet and synchronously execute actions. In this example, robots r_1 and r_2 must meet in cell c_7 and synchronously execute actions *ld* and *rd*, respectively. The cells where meeting is requested are marked with rotating arrows marked with the identifiers of the robots that must meet, meaning that, in order to meet, the robots must be on the same cell to meet.

The *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 Linear Time Temporal Logic (LTL). An LTL formula is obtained by composing actions with standard LTL operators: X (next), F (eventually), G (always) and U (until) [48]. In our example the mission can be specified by means of the following local missions: $\phi_1 = \mathsf{G}(\mathsf{F}(load\_carrier))$, $\phi_2 = \mathsf{G}(\mathsf{F}(detect\_load \wedge \mathsf{F}(unload)))$, $\phi_3 = \mathsf{G}(\mathsf{F}(take\_snapshot \wedge \mathsf{F}(send\_info)))$, which are assigned to robot r_1, r_2 and r_3, respectively. The formulae specify that periodically robot r_1 loads debris on r_2 (by performing action *load_carrier*), robot r_2 receives debris (when action *detect_ load* occurs) and brings them to an appropriate unload area (by performing action *unload*), and robot r_3 continuously takes pictures (by performing action *take_snapshot*) and sends them using the communication network (by performing action *send_info*). 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 by the container.

The presence of partial knowledge about the robots and their environment is described in the following.

Partial knowledge about the actions execution. The robots can move between cells separated by grey lines, while they cannot cross black bold lines. 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 in location l_3 and specifically in cell c_{18}, i.e., whether action s_p can be performed. Locations of the environment where it is unknown if an action can be provided 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 $ud2$ will always be able to provide the *unload* service, while it is unknown whether $ud1$ is actually able to provide this service since its effectiveness depends on the size of the collected debris. In Fig. 1, the label $L(ud1, unload) =?$ indicates that there is partial knowledge about the provision of the *unload* service when action $ud1$ 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 services ld and rd, i.e., 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 PROBLEM FORMULATION

This section describes how we extend the model of a robotic application presented in [55] to represent robotic applications in which partial knowledge about the action execution, the service provisioning, and the robot meeting capabilities is present. Then, we formulate the decentralized planning problem for robot applications with partial knowledge.

Modeling robot applications with partial knowledge. A partial robot model is defined as follows.

Definition 3.1. Consider a set of robots $\mathcal{R}$. A *partial robot model* of a single robot is represented by a tuple $r = (S, init, A, \Pi, T, T_p, Meet, Meet_p, L)$, where

- S is a finite set of states;
- $init \in S$ 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 $T_p(s, \alpha) = s'$;
- $Meet, Meet_p : S \rightarrow (\wp(\mathcal{R}) \cup \{\#\})$ are functions ensuring:
  (1) for all $s \in S$ either $Meet(s) \subseteq \wp(\mathcal{R})$ or $Meet(s) = \{\#\}$;
  (2) for all $s \in S$ such that $Meet(s) \neq \emptyset$, $Meet_p(s) = Meet(s)$;
- $L : A \times \Pi \rightarrow \{\top, \bot, ?\}$ is the service labeling.

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 $s \overset{\alpha}{\dashrightarrow} s'$. 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 can move 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*. When a robot r executes a maybe transition $s \overset{\alpha}{\dashrightarrow} s'$, either it reaches state s', when the transition is executable, or it remains in s, if the transition is not executable. A transition $T_p(s, \alpha) = s'$, such that $T(s, \alpha) = s'$, is a *definitive transition* and is indicated as $s \xrightarrow{\alpha} s'$. When a robot r executes a transition $s \xrightarrow{\alpha} s'$ it reaches state 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 $L(\alpha, \pi) = \bot$, 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 by the robot 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 $Meet(s) = \emptyset$ and $Meet_p(s) \neq \emptyset$, it is uncertain whether the robots in $Meet_p(s)$ can meet. Otherwise $Meet(s) = Meet_p(s)$, i.e, the meeting capabilities are definitive. When $Meet(s) = \{\#\} = Meet_p(s)$ the meeting is not possible.

A definitive robot model is a partial robot model that does not contain partial information about the transition relation, the service provisioning, and the meeting capabilities. Formally:

Definition 3.2. A *definitive robot model* $r = (S, init, A, \Pi, T, Meet, L)$ is a *partial robot model* where $T = T_p$, $Meet = Meet_p$ and $L(\alpha, \pi) \neq ?$ for all (α, π) such that $\alpha \in A$ and $\pi \in \Pi$.

A plan describes a set of states a robot has to traverse and action to be performed in these states.

Definition 3.3 (Definitive and possible plan). Given a partial robot model $r = (S, init, A, \Pi, T, T_p, Meet, Meet_p, L)$, a *definitive plan* of r is an infinite alternating sequence of states and actions $\beta = s_1, \alpha_1, s_2, \alpha_2, \ldots$, such that $s_1 = init$, for all $i \geq 1$ and $\pi \in \Pi$, $s_i \xrightarrow{\alpha_i} s_{i+1}$, $Meet(s_i) = Meet_p(s_i)$, $Meet(s_i) \neq \{\#\} \neq Meet_p(s_i)$ and $L(\alpha_i, \pi) \neq ?$. A *possible plan* of r is infinite alternating sequence of states and actions $\beta = s_1, \alpha_1, s_2, \alpha_2, \ldots$, such that $s_1 = init$, and for all $i \geq 1$, $s_i \overset{\alpha_i}{\dashrightarrow} s_{i+1}$ and $Meet(s_i) \neq \{\#\} \neq Meet_p(s_i)$.

Running example. *The plan $c_{17}, mov, c_{23}, mov, c_{29}, mov, c_{30}, (sp, c_{30})^\omega$ is a definitive plan for robot r_3 since all the transitions, service provisioning and meeting capabilities are definitive. The plan $c_{26}, mov, c_{20}, mov, c_{14}, (mov, c_{14})^\omega$ for robot r_1 is a possible plan since $c_{20} \overset{mov}{\dashrightarrow} c_{14}$ is a maybe transition.*

We use the notation $\mathcal{B}_d(r)$ and $\mathcal{B}_p(r)$ to indicate the set of all definitive and possible plans of a robot r, respectively. Note that $\mathcal{B}_d(r) \subseteq \mathcal{B}_p(r)$, i.e., a definitive plan is also a possible plan.

Definition 3.4 (Refinement). Given two partial robot models r and r', the partial robot model r' *refines* r, i.e., $r \preceq r'$, iff $\mathcal{B}_d(r) \subseteq \mathcal{B}_d(r')$ and $\mathcal{B}_p(r') \subseteq \mathcal{B}_p(r)$.

A model r' refines r if it contains more definitive plans, i.e., definitive plans of $\mathcal{B}_d(r)$ are a subset (or equal) of definitive plans of r' and if the partial information about r is reduced, i.e., possible plans of $\mathcal{B}_p(r')$ are a subset (or equal) of possible plans of r.
If the obtained partial robot model r' is a definitive robot model, it is a completion of a partial robot model r.

Definition 3.5 (Completion). Given a definitive robot model r' and a partial robot model r, r' is a *completion* of r iff $r \preceq r'$.

We define $C(r)$ as the set of all the completions of a partial robot model r.

A plan is *executed* by a robot. When a plan is executed, the states traversed by the robot and the executed actions are recorded. An execution e of a plan is an infinite sequence $s_1, \alpha_1, s_2, \alpha_2, \ldots$ of states and actions. A robot starts executing a definitive or possible plan starting from its initial state *init* i.e., the initial state s_1 of the execution is *init*. Let us first consider definitive plans and assume that a robot r performs a definitive transition in the form $s \xrightarrow{\alpha} s'$ at step i. Then, state s_i of the execution is s, s' and α are assigned to s_{i+1} and α_i. When possible plans are considered, the following rules are applied: (1) when a robot r attempts to execute a maybe transition $s \overset{\alpha}{\dashrightarrow} s'$, either it detects that the transition is executable and it reaches state s' or, if it is not, it remains in s. In the first case, the state s_{i+1} of the execution is equal to s' and α_i is equal to α. In the second, the state s_{i+1} of the execution is equal to s and α_i is equal to τ. Furthermore, the partial model r of the robot is updated: if the transition is executable, it is added to T, otherwise it is removed from T_p; (2) when a robot r executes a maybe transition $s \overset{\alpha}{\dashrightarrow} s'$ or a definitive transition $s \xrightarrow{\alpha} s'$ and the state s' is reached, if a service π is such that $L(\alpha, \pi) =?$, either the service is provided or it is not. In the first case, robot r is updated by assigning value $\top$ to $L(\alpha, \pi)$, in the second robot r is updated by assigning value $\bot$ to $L(\alpha, \pi)$; (3) when a robot r executes a maybe transition $s \overset{\alpha}{\dashrightarrow} s'$ or a transition $s \xrightarrow{\alpha} s'$ such that $Meet_p(s') \neq \emptyset$ and $Meet(s') = \emptyset$, either the meeting is possible or it is not. In the first case, $Meet(s') = Meet_p(s')$, the state s_{i+1} of the execution is equal to s' and α_i is equal to α. In the second case, the set $\{\#\}$ is assigned to $Meet(s')$ and $Meet_p(s')$ meaning that no meeting is possible. Furthermore, the state s_{i+1} of the execution is then equal to s and α_i is equal to τ.

Executing plans allows discovering information about action execution, service provisioning, and meeting capabilities. When one of the previous rules is applied we say that new information about the partial robot model 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.

THEOREM 3.6. *Let us consider a partial robot model r. As the robot executes a plan, the model r' obtained by updating model r as specified in the previous rules is a refinement of r.*

PROOF. When rules (1), (2) and (3) are applied and true evidence about the partial information is detected, the transition is transformed into a definitive transition, the service is provided or the meeting is possible, $\mathcal{B}_d(r) \subseteq \mathcal{B}_d(r')$ and $\mathcal{B}_p(r') \subseteq \mathcal{B}_p(r)$. By Definition 3.3 every plan that was involving the transition, the services or the meeting capability was a possible plan. Since true evidence is detected, possible plans are not removed from the partial model, i.e., $\mathcal{B}_p(r') = \mathcal{B}_p(r)$. Since partial knowledge about a transition or a service or a meeting capability is transformed into definitive knowledge, plans that were possible may now be also definitive, i.e., additional plans are added to the set of definitive plans of r', i.e., $\mathcal{B}_d(r) \subseteq \mathcal{B}_d(r')$.
When rules (1), (2) and (3) are applied and false evidence about the partial information is detected, the transition is removed, the service is not provided or the meeting is not possible. Since the plan was involving partial information, by Definition 3.3 it was a possible plan. Furthermore, by Definition 3.3, any plan that involved the removed transition, the not provided service or the meeting capability was a possible plan. Thus, rules (1), (2) and (3) do not modify the set of definitive plans of r when false evidence about partial information is detected, i.e., $\mathcal{B}_d(r) = \mathcal{B}_d(r')$. On the contrary, by removing the transition, by not providing the service or by avoid meeting the set of possible plans is reduced, i.e., $\mathcal{B}_p(r') \subseteq \mathcal{B}_p(r)$. □

THEOREM 3.7. *Let us consider a partial robot model r. As the robot executes a plan, if the model r' obtained by updating model r as specified in the previous rules is a definitive model it is a completion of model r.*

PROOF. Theorem 3.6 proved that as the robot executes a plan, refinements of the model r is generated. If the obtained model is a definitive model, by Definition 3.5 it is a completion of r. □

Definition 3.8. A *definitive (partial) robot application $\mathcal{H}$* is a set of definitive (partial) robot models $\{r_1, r_2, \ldots, r_N\}$ such that for every couple of definitive (partial) robot models $r_n, r_m \in \mathcal{H}$, where r_n=(S_n, $init_n$, A_n, Π_n, T_n, $T_{p,n}$, $Meet_n$, $Meet_{p,n}$, L_n) and r_m=(S_m, $init_m$, A_m, Π_m, T_m, $T_{p,m}$, $Meet_m$, $Meet_{p,m}$, L_m), the following is satisfied $\Pi_n \cap \Pi_m = \emptyset$ and $A_n \cap A_m = \emptyset$.

Definition 3.9. A partial robot application $\mathcal{H}$' defined over the set of robots $\{r'_1, r'_2, \ldots, r'_N\}$ is a refinement of a partial robot application $\mathcal{H}$ defined over the set of robots $\{r_1, r_2, \ldots, r_N\}$ if and only if $r_n \preceq r'_n$ holds for every robot $r_n \in \{r_1, r_2, \ldots, r_N\}$.

Definition 3.10. A definitive robot application $\mathcal{H}$' is a completion of a partial robot application $\mathcal{H}$ if and only if $\mathcal{H}$' refines $\mathcal{H}$.

A definitive robot application is obtained by refining robots of a partial robot application.

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

The notion of plan, however, should reflect the meeting scheme. Let us consider a partial or definitive robot application $\mathcal{H}$ and a set $\mathcal{B} = \{\beta_1, \beta_2, \ldots, \beta_N\}$ of definitive or possible plans, where β_n is the plan for the robot r_n in $\mathcal{H}$. Let us consider the set $\mathcal{E} = \{e_1, e_2, \ldots, e_N\}$ of the corresponding executions, where we use the notation $e_n = s_{n,1}, \alpha_{n,1}, s_{n,2}, \alpha_{n,2}, \ldots$ to identify the execution of the plan β_n. The plans in $\mathcal{B}$ must ensure a meeting paradigm. The plans that satisfy the meeting paradigms are *compatible plans*.

Definition 3.11 (Definitively and possibly compatible plans). 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 execution $e_n = s_{n,1}, \alpha_{n,1}, s_{n,2}, \alpha_{n,2}, \ldots$ of each robot r_n the following must hold:

- $Meet(s_{n,j}) \neq \{\#\}$;

- if $s_{n,j-1} \neq s_{n,j}$ and $Meet(s_{n,j}) \neq \emptyset$ then (1) $s_{n,j} = s_{n,j+1}$; (2) for all $r_m \in Meet_n(s_{n,j})$, it holds that $s_{m,j} = s_{n,j}$ and $s_{m,j+1} = s_{n,j}$.

The first condition ensures that meeting is always possible. The second condition enforces 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 $Meet_n(s_{n,j})$ reaches state $s_{n,j}$ at step j and remains in $s_{n,j}$ at step $j+1$.

Mission specification. Given a *robot application* $\mathcal{H} = \{r_1, r_2, \ldots, r_N\}$, 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 $\Pi = \Pi_1 \cup \Pi_2 \cup \ldots \cup \Pi_N$. We assume that the mission assigned to the team of robots is made by a set $\Phi = \{\phi_1, \phi_2, \ldots, \phi_N\}$ of local missions such that each mission $\phi_n \in \Phi$ is assigned to the robot r_n of the team. 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 the team. When we consider definitive robot applications we use the classical LTL semantics over infinite words $w = \varpi_1\varpi_2\ldots$. Given a local mission ϕ_n assigned to a robot r_n, we indicate as $\mathcal{B}_n$ the Büchi automaton obtained from ϕ_n. Further details can be found in [55].

Definition 3.12 (Local LTL satisfaction). Let r_n be a robot and D a set of robots of a definitive robotic application $\mathcal{H}$ such that $\{r_n\} \subseteq D \subseteq \mathcal{H}$ and let β_n be a plan of r_n. 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 $w_m = \varpi_{m,1}\varpi_{m,2}\ldots$. The *word* produced for set D is $w_{\mathfrak{B}} = \omega_1\omega_2\ldots$, where for all $j \geq 1$, for all $\pi \in \Pi$, $\omega_j(\pi) = \bigvee_{r_m \in D} \varpi_{m,j}(\pi)$. The set of compatible plans $\mathfrak{B}$ *locally satisfies* ϕ_n for the robot n, i.e., $\mathfrak{B} \models \phi_n$, iff $w_{\mathfrak{B}} \models \phi_n$.

Given a set of compatible plans $\mathfrak{B}$ that locally satisfies ϕ_n, the execution τ_m associated with each plan $\beta_m \in \mathfrak{B}$, is such that $e_m = s_{1,m}\alpha_{1,m}\, s_{2,m}\ldots(s_{i,m}\,\alpha_{i,m}\, s_{i+1,m}\,\alpha_{i+1,m}\ldots s_{k-1,m}\,\alpha_{k-1,m}s_{k,m})^{\omega}$. Intuitively, the execution is made by two parts: a prefix and a suffix which occur infinitely many often. This is a consequence of a well known property of LTL that ensures the following: if a word satisfies an LTL property, it can be decomposed into a prefix and a suffix which occurs infinitely many often [13]. The length $length(e_m)$ of the execution e_m is k. Given a set of compatible plans $\mathfrak{B}$ satisfying ϕ_n, the length of the set of plans $\mathfrak{B}$ is the maximum between the lengths of the executions of the plans in $\mathfrak{B}$.

Planning. The planning goal is to identify a set of definitive plans (one for each robot of the team) that ensure the satisfaction of local missions.

Problem 1 (Planning). *Consider a partial robot application $\mathcal{H}$ defined over the set of partial robot models $\{r_1, r_2, \ldots, r_N\}$. Given a set of missions $\{\phi_1, \phi_2, \ldots, \phi_n\}$, one for each robot $r_n \in \mathcal{H}$ find a set of plans $\mathfrak{B} = \{\beta_1, \beta_2, \ldots, \beta_N\}$ that (1) are compatible and (2) $\mathfrak{B}$ locally satisfies each ϕ_n.*

4 PLANNING WITH PARTIAL KNOWLEDGE

Solving the planning problem requires defining what it means that a possible plan satisfies a mission. This section provides different definitions of this concept, on the top of which our planning algorithm is developed. Specifically, the three-valued LTL semantics and the thorough LTL semantics are defined. 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 thorough LTL semantics allows returning a precise result but requires implementing a checking procedure that is more computationally expensive.

Consider for a moment a single partial robot $r = (S, init, A, \Pi, T, T_p, Meet, Meet_p, L)$, and a definitive (possible) plan $\beta = s_1\alpha_1 s_2\alpha_2\ldots$. The *definitive (possible) word* associated with β is $w_\tau = \varpi_1\varpi_2\ldots$, 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 $\{\top, \bot\}$, while possible words w_τ associate services with values in the set $\{\top, ?, \bot\}$.

The three-valued LTL semantics is based on the ordering relation $>$ between values $\{\top, ?, \bot\}$ such that $\top > ? > \bot$. Based on this ordering, the following functions are defined as usual [7]: (1) complement (*comp*) such that $comp(\top) = \bot$, $comp(\bot) = \top$ and $comp(?) =?$; (2) minimum (min) such that $\min(\top, \bot) = \bot$, $\min(\top, ?) = ?$ and $\min(?, \bot) = \bot$ and (3) maximum (max) such that $\max(\top, \bot) = \top$, $\max(\top, ?) = \top$ and $\max(?, \bot) = ?$.

Definition 4.1 (Three-valued LTL semantics). Let us consider a property ϕ and a word $w_\tau = \varpi_1\varpi_2\ldots$. The three valued semantics $\overset{3}{=}$ associates to a word w_τ a value in the set $\{\top, ?, \bot\}$ as follows:

$$[\varpi_i \models \pi] \overset{3}{=} \varpi_i(\pi)$$
$$[\varpi_i \models \neg\phi] \overset{3}{=} \text{comp}([\varpi_i \models \phi])$$
$$[\varpi_i \models \phi_1 \wedge \phi_2] \overset{3}{=} \min([\varpi_i \models \phi_1], [\varpi_i \models \phi_2])$$
$$[\varpi_i \models X\phi] \overset{3}{=} [\varpi_{i+1} \models \phi]$$
$$[\varpi_i \models \phi_1 U \phi_2] \overset{3}{=} \max_{j \geq 0}(\min(\{[\varpi_k \models \phi_1] | i \leq k < j\} \cup \{[\varpi_j \models \phi_2]\}))$$

The three-valued LTL semantics 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 [7]. However, while a formula is evaluated as $\top$ and $\bot$ reflecting the natural intuition, it has been shown [8] 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 $\bot$, such that w'_τ satisfies ϕ and w''_τ does not satisfy ϕ. This statement is not always satisfied. For example, consider the formula $a \wedge \neg a$ and a word w_τ such that for all $i \geq 1, \varpi_i(a) =?$, the formula is evaluated to ? but it is trivially unsatisfiable. For this reason the thorough LTL semantics has been proposed [8].

The thorough LTL semantics is usually defined considering partial models [8]. 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 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.

Definition 4.2 (Word refinement). Given two possible words $w_\tau = \varpi_1\varpi_2\ldots$ and $w'_\tau = \varpi'_1\varpi'_2\ldots$ such that for all $\pi \in \Pi$ and $i \in \{1, 2\ldots\}$, $\varpi_i(\pi) \in \{\top, \bot, ?\}$ and $\varpi'_i(\pi) \in \{\top, \bot, ?\}$, the (possible) word

w'_τ is a refinement of a possible word w_τ, indicated as $w_\tau \preceq w'_\tau$ if and only if the following conditions hold (1) for all $i \geq 0$, $\pi \in \Pi$, if $w_i(\pi) = \top \rightarrow w'_i(\pi) = \top$; (2) for all $i \geq 0, \pi \in \Pi$, if $w_i(\pi) = \bot \rightarrow w'_i(\pi) = \bot$;

Intuitively, w'_τ can be obtained from w_τ by assigning to some $w_i(\pi)$ such that $w_i(\pi) =?$ a $\top$ or a $\bot$ value.

Definition 4.3 (Word completion). A definitive word $w'_\tau = \varpi'_1\varpi'_2\ldots$ is a completion of a possible word w_τ if and only if w'_τ refines w_τ.

Definition 4.4 (Thorough LTL word semantics). Given a word w_τ and a property ϕ, the thorough LTL semantics $\overset{T}{=} \top$ associates the word w_τ with a value in the set $\{\top, ?, \bot\}$ as follows (1) $[w_\tau \models \phi] \overset{T}{=} \top$ iff all the word completions w'_τ of w_τ are such that $[w'_\tau \models \phi] = \top$; (2) $[w_\tau \models \phi] \overset{T}{=} \bot$ iff all the word completions w'_τ of w_τ are such that $[w'_\tau \models \phi] = \bot$; (3) $[w_\tau \models \phi] \overset{T}{=}?$ otherwise.

Note that, 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. Thus, the followings lemmas hold.

Lemma 4.5. *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) $[w_\tau \models \phi] \overset{3}{=} \top \Rightarrow [w'_\tau \models \phi] \overset{T}{=} \top$; (2) $[w_\tau \models \phi] \overset{3}{=} \bot \Rightarrow [w'_\tau \models \phi] \overset{T}{=} \bot$.*

Proof. Lemma 4.5 has been proved to hold for partial models, i.e., it has been proved to hold for partial Kripke structures (PKSs) [8]. A word $w_\tau = \varpi_1\varpi_2\ldots$ can be considered as a PKS obtained as follows:

1. a new state s_i of the PKS is created for every $\varpi_i \in w_\tau$;
2. the initial state of the PKS is state s_0;
3. each service π is associated with a proposition a of the PKS;
4. for each state s_i of the PKS the function L associates to each proposition a in the set of proposition of the PKS the value $\varpi_i(\pi)$;
5. for each s_i, s_{i+1} a transition is added from s_i to s_{i+1}.

Thus, since words w_τ and w'_τ can be considered as PKS, it follows that Lemma 4.5 also holds for words. □

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 [23].

Since word w_τ can be considered as a PKS M obtained using the procedure previously specified, the model checking procedure can encode the word in a PKS which is then checked against property ϕ. Checking whether a PKS satisfies a property ϕ w.r.t. the thorough LTL semantics can be done in polynomial time in the length of w_τ and double exponential time in the size of ϕ [23]

There exists a subset of LTL formulae, known in literature as *self-minimizing* formulae, such that given a word the three valued and the thorough semantics coincide [22]. More precisely, given a word w_τ and a self-minimizing LTL property ϕ, then $[w_\tau \models \phi] \overset{T}{=} x \overset{3}{=} [w_\tau \models \phi]$ where $x \in \{\bot, \top, ?\}$. It has been observed that most practically useful LTL formulae, such as absence, universality, existence, response and response chain, are self-minimizing [22]. 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 [8].

The previous definitions allow us to define the notion of local definitive and possible LTL satisfaction.

Definition 4.6 (Local LTL definitive and possible satisfaction). Let $\overset{X}{=}$ be a semantics in $\{\overset{3}{=}, \overset{T}{=}\}$. 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 $w_m = \varpi_{m,1}\varpi_{m,2}\ldots$. The *word* produced by a set of definitive (resp. possible) compatible plans $\mathfrak{B} = \{\beta_m \mid r_m \in D\}$ is $w_\mathfrak{B} = \omega_1\omega_2\ldots$, where for all $j \geq 1$, 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 $[w_\mathfrak{B} \models \phi_n] \overset{X}{=} \top$ ($[w_\mathfrak{B} \models \phi_n] \overset{X}{=}?$).

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 $\bot$ to ? such that it is possible to obtain a word that satisfies ϕ and another that does not satisfy ϕ. This ensures that there is a chance that, if executed, the plan satisfies a mission ϕ. This is not true for the three-valued semantics. If this semantics is considered, it can be the case that a plan possibly satisfies a mission ϕ, but the mission is trivially unsatisfiable.

Based on these definitions, Problem 1 is reformulated as follows:

Problem 2 (Planning). Consider *a partial robot application $\mathcal{H}$ defined over the set of partial robot models $\{r'_1, r'_2, \ldots, r'_N\}$ and a semantics $\overset{X}{=}$ in $\{\overset{3}{=}, \overset{T}{=}\}$. Given a set of missions $\{\phi_1, \phi_2, \ldots, \phi_n\}$, one for each robot $r_n \in \mathcal{H}$ find a set of plans $\mathfrak{B} = \{\beta_1, \beta_2, \ldots, \beta_N\}$ that (1) are compatible and (2)* $\mathfrak{B}$ locally definitively (resp. possibly) satisfies *each ϕ_n w.r.t. the given semantics.*

5 ALGORITHMS

We discuss how MAPmAKER solves Problem 1 by implementing a decentralized planning with partial knowledge. The algorithm is decentralized since it tries to decompose the robots within the robotic application into dependency classes. A dependency class $dp_n = r_1, r_2, \ldots, r_n$ is a subset of robots that depends on each other for achieving their missions. Two robots r_n=(S_n, $init_n$, A_n, Π_n, T_n, $T_{p,n}$, $Meet_n$, $Meet_{p,n}$, L_n) and r_m=(S_m, $init_m$, A_m, Π_m, T_m, $T_{p,m}$, $Meet_m$, $Meet_{p,m}$, L_m) with mission 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.

We first briefly overview the steps performed by MAPmAKER which are presented in Algorithm 1. Then, we will discuss how dependency classes are computed and how to plan with partial knowledge. MAPmAKER gets as input a partial robot application $\mathcal{H}$ and a set of missions Φ, i.e., one for each robot in the partial robot application. First (Line 3), MAPmAKER creates a set of empty plans, one for each robot of the partial robot application. Then (Line 4), the set of robots in the partial robot application is partitioned into dependency classes based on the missions the robots have to achieve. Each dependency class $dp_n \in \{dp_1, dp_2, \ldots, dp_M\}$ is a partial robot application, each $\Psi_n \in \{\Psi_1, \Psi_2, \ldots, \Psi_M\}$ contains a set of missions for the robots in the partial robot application dp_n. Then, each dependency class dp_n is analyzed (Lines 5-17). The variable *disc* (Line 6) is used to track whether new information about the partial robot models in dp_n has been discovered while plans are executed. Then the planner for partial robot application is executed on dp_n (Line 9). If the variable *pp* is equal to the value *false*, no plan is available and the value *NO_PLAN* is returned (Lines 10-??). Otherwise (Lines 11-15), the plans returned by the planner are executed. The algorithm executes one action for each robot per time, by calling function *EX* until the maximum length of the plans is reached. Variable *i* keeps the index of the actions to be executed next (Line 13). Note that, as specified in Section 3 when actions are executed by robots the models of the robots are updated. If one of the robots in the partial robot application detects a *false evidence* about partial information, value 0 is returned and assigned to variable *disc*. Otherwise, value 1 is assigned to variable *disc*. When *false evidence* about partial information is detected, the loop of Line 12 is left and the planner for partial robot applications (Line 9) is re-executed.

Algorithmus 1 The MAPmAKER main loop.

```
1:  Input a partial robot application $\mathcal{H}$ and a set of missions $\Phi$
2:  Output a set of definitive plans (if they exist)
3:  $\{\beta_1, \beta_2, \ldots, \beta_N\} = CREATE\_PLANS(\mathcal{H})$;
4:  $\{dp_1, dp_2, \ldots, dp_M, \Psi_1, \Psi_2, \ldots \Psi_M\} = DEP\_CLASSES(\mathcal{H}, \Phi)$;
5:  for $dp_n \in \{dp_1, dp_2, \ldots, dp_M\}$ do
6:      $disc = 1$
7:      while $disc == 1$ do
8:          $disc = 0$;
9:          $[pp, \{pd_1, pd_2, \ldots, pd_M\}] = PARTIAL\_PLAN(dp_n, \Psi_n)$;
10:         if $pp = false$ then return $NO\_PLAN$; end if
11:         $i = 1$;
12:         while $i \leq LENGTH(\{dp_1, dp_2, \ldots, dp_M\})$ & $disc == 1$ do
13:             $disc = EX(\{dp_1, dp_2, \ldots, dp_M\}, i, \{\beta_1, \beta_2, \ldots, \beta_N\})$;
14:             $i = i + 1$;
15:         end while
16:     end while
17: end for
18: return $\{\beta_1, \beta_2, \ldots, \beta_N\}$
```

Compute the dependency classes To compute dependency classes the *DEP_CLASSES* function iteratively applies the following rule [24]: 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.

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 2 allows planning with partial knowledge. The algorithm receives a partial robot application and a set of properties and computes a definitive plan (or a possible plan) that ensures the property satisfaction. We discuss how the different types of partial information are handled by the planning Algorithm 2. This is done by incrementally enabling the proposed planning algorithm to handle types of partial information. Identifiers A1, A2, and A3 are used in Algorithm 2 to 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 $\mathcal{H} = \{r_1, r_2, \ldots, r_N\}$ where each robot $r_n = (S_n, init_n, A_n, \Pi_n, T_n, T_{p,n}, Meet_n, Meet_{p,n}, L_n)$ is such that $Meet_n = Meet_{p,n}$ and for each service $\pi \in \Pi_n$ and action $\alpha \in A_n$, $L_n(\alpha, \pi) \in \{\top, \bot\}$. In this case, partial information only refers to the presence of maybe transitions, i.e., transitions $s_n \overset{\alpha}{\dashrightarrow} s'_n$ such that $s_n \xrightarrow{\alpha} s'_n \notin T_n$. Lines marked with the identifier A1 allow Algorithm 2 to handle this case.

The algorithm works in two steps. *Step 1.* For each robot $r_n \in \mathcal{H}$ it removes the transitions $s_n \overset{\alpha}{\dashrightarrow} s'_n$ such that $s_n \xrightarrow{\alpha} s'_n \notin T_n$ (Line 3) and applies a classical decentralized planning algorithm for definitive models (Line 8), such as the one proposed in [55]. Variable *pd* contains whether the planner had found a plan, $\{pd_1, pd_2, \ldots, pd_N\}$ contains the plans if they are founded. If a plan is found, *pd* is assigned to *true* and the definitive plans for each of the robots are stored in variables $\{pd_1, pd_2, \ldots, pd_N\}$. 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 robots) are stored in $\{pp_1, pp_2, \ldots, pp_N\}$. *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 and a *NO_PLAN_AVAILABLE* value is returned. If *pd* is *false* while *pp* is not, then only possible plans are available and they are returned as output. Otherwise, an appropriate policy is used to choose between $\{pd_1, pd_2, \ldots, pd_N\}$ and $\{pp_1, pp_2, \ldots, pp_N\}$.

Correctness. If a set of plans is found in Step 1, they are definitive plans and the set $\{pd_1, pd_2, \ldots, pd_N\}$ locally satisfies ϕ_n for all $r_n \in \mathcal{H}$ due to the correctness of the algorithm invoked by the *DEC_PLANNER* function. The obtained plans also locally *definitively* satisfy ϕ_n since the obtained plans are definitive, i.e., they are obtained by traces that do not include any maybe transition. If a plan is found in Step 2, and no plan was found in Step 1, the obtained plans *possibly* satisfy ϕ_n. Indeed, the obtained plans must include some maybe transition. □

Remark 1. *This work does not discuss how to choose between possible and definitive plans. Possible plans can be computed in cases in which definitive plans are not present or they can always be computed and appropriate policies can be used to select between definitive and possible plans. A policy may choose for example the plan with the shortest length, or it may consider non-functional aspects of the plans e.g., time to perform certain actions, or likelihood of detecting true or false evidence about partial information. In the following we assume that the planner always chooses the shortests between the possible and the definitive plan*[1]*. This policy may for example reduce energy consumption, since every action performed by the robots may consume energy. Thus, shorter plans require less energy.*

Running example. *Let us assume that no meeting primitive is specified in cell c_9, actions ld and rd cannot be performed by robots*

[1]Note that, in this case, it is still necessary to execute Step 1 of Algorithm 1 since Step 2 may return a possible plan while a definitive plan of the same length is present.

Algorithmus 2 The *PARTIAL_PLAN* function.

```
1:  Input a partial robot application $\mathcal{H}$ and a set of missions $\Phi$
2:  Output a definitive or a possible plan (if they exist)
3:  For each robot $r_n$ remove the transitions $s_n \overset{\alpha}{\dashrightarrow} s'_n$ such that $s_n \xrightarrow{\alpha} s'_n \notin T_n$ (A1)
4:  Put each formula $\phi_n \in \Phi$ in its negation normal form (A2)
5:  Compute the complement closed model $r'_n$ of each $r_n \in \mathcal{H}$ (A2)
6:  For $r'_n$ and $s \in S_n$ if $Meet_{n,p}(s) \neq Meet_n(s)$ then $Meet_{n,p}(s) = \{\#\}$ (A3)
7:  Construct the pessimistic approximation of each $r_n \in \mathcal{H}$ (A2)
8:  $[pd, \{pd_1, pd_2, \ldots, pd_N\}]$=DEC_PLANNER ($\{r'_1, r'_2, \ldots, r'_N\}, \Phi$)
9:  For each robot $r_n$ insert the transitions $s_n \overset{\alpha}{\dashrightarrow} s'_n$ such that $s_n \xrightarrow{\alpha} s'_n \notin T_n$ (A1)
10: Construct the optimistic approximation of each $r_n \in \mathcal{H}$ (A2)
11: For $r'_n$ consider all the meeting requests in $Meet_{n,p}$ (A3)
12: $[pp, \{pp_1, pp_2, \ldots, pp_N\}]$=DEC_PLANNER ($\{r'_1, r'_2 \ldots, r'_N\}, \Phi$)
13: if $pd = false$ and $pp = false$ then return NO_PLAN_AVAILABLE
14: end if
15: if $pd = false$ then return $\{pp_1, pp_2, \ldots, pp_N\}$;
16: else return choose($\{pd_1, pd_2, \ldots, pd_N\}, \{pp_1, pp_2, \ldots, pp_N\}$).
17: end if
```

r_1 and r_2 in c_9 and that 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 a possible plan since it is not known whether robot r_1 can move from cell c_8 to cell c_{14}. Plan p_3 is also 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 [7], specified by Lines marked with the identifier A2.

• *Step 1.* For each r_n, remove all the transitions $s_n \overset{\alpha}{\dashrightarrow} s'_n$ 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).
• *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 $\overline{\pi}$, called complement-closed service, such that $L_n(\alpha, \overline{\pi}) = comp(L_n(\alpha, \pi))$ (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 $L_n(\alpha, \pi) = ?$, $L_{n,pes}(\alpha, \pi) = \bot$ 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 $\{pd_1, pd_2, \ldots, pd_N\}$ otherwise a *false* value is associated to pd.
• *Step 6.* For each r_n insert all the transitions $s_n \overset{\alpha}{\dashrightarrow} s'_n$ 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 $\{pp_1, pp_2, \ldots, pp_N\}$ 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, thus Algorithm 2 returns the special value *NO_PLAN_AVAILABLE*. If pd is *false* while pp is not (Line 15), then no definitive plans are available while a possible plan has been found. The possible plan is returned as output. Otherwise (Line 16), an appropriate policy can be used to choose between $\{pp_1, pp_2, \ldots, pp_N\}$ and $\{pd_1, pd_2, \ldots, pd_N\}$.

Correctness. If a plan is found in Step 5, it is a definitive plan. The plan is obtained by only executing definitive transitions. Furthermore, the considered model is obtained putting the formula in negation normal form and setting the values of the services in the complement closed structure to $\bot$, meaning that the services are not provided. If a mission is found it definitely satisfies the property of interest w.r.t. the three valued semantics. If properties are self-minimizing, by Lemma 4.5, this result also holds considering the thorough semantics.
If a plan is found in Step 8, it is a possible plan. The plan is obtained by executing only definitive and possible transitions. Furthermore, the plans are obtained putting the formula in negational normal form and setting the values of the services in the complement closed structure to $\top$. If a mission is found it possibly satisfies the property of interest w.r.t. the three valued semantics. By Lemma 4.5 this result also holds considering the thorough semantics if properties are self-minimizing. □

Running example. *Let us assume that no meeting is required in cell c_9, that actions ld and rd cannot be performed by robots r_1 and r_2 in c_9 and that it is not known whether ud_2 provides 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 a possible plan since it is unknown whether the execution of action ud_2 provides the unload service.*

Managing uncertainties in the meeting capabilities. Partial knowledge in the meeting capabilities can be handled considering lines marked with identifier A3. These lines ensure that before searching for a definitive plan, all the meeting requests that are only 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.

Correctness. If a plan is found in Line 8 it is a definitive plan, since it only exploits meeting capabilities that are present in the real robot application. If a plan is found in Line 12 it is a possible plan, since it may be using meeting capabilities that are only possible. □

Running example. *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.*

Overall correctness and complexity. If a set of plans $\mathfrak{B} = \{\beta_1, \beta_2, \ldots, \beta_N\}$ that (1) are compatible and (2) $\mathfrak{B}$ locally definitively satisfies each ϕ_n, exist in the real robot application they are returned from Algorithm 2 if the partial robot application satisfies the following, if a movement is possible from cell c_i to c_j, then a movement is also possible from c_j to c_i. In this case, as Algorithm 1 is performed, new knowledge about the real robot application is added to the partial robot application, until (in the worst case) the model of the partial robot application corresponds to the real robot application. If a set of plans that are compatible and and $\mathfrak{B}$ locally definitively satisfies each ϕ_n exist, in the worst case, it must be returned when Algorithm 2 is performed on the model of the real robot application. Assume now that a movement is possible from cell c_i to c_j, but not from c_j to c_i and that the set of plans $\mathfrak{B}$ that are compatible and such that $\mathfrak{B}$ locally definitively satisfies each ϕ_n, do not require one of the robots to move from c_i to c_j. The robot may move from c_i to c_j while it is following a possible plan. However, after reaching c_j and performing the rest of the possible plan it may detect false evidence about the partial information. At this point, it may be impossible to came back to c_i and to compute the set of plans $\mathfrak{B}$.

Algorithm 2 uses the three-valued LTL semantics. Thus, for general LTL properties, if a possible plan is returned it may be spurious considering the thorough semantics. If the property of interest is self-minimizing the computed plans are correct considering both the semantics.

Algorithm 2 relies on calling twice a classical decentralized model checking algorithm. This algorithm is called by Algorithm 1 every time false evidence about partial information is detected.

6 EVALUATION

This section reports on our experience evaluating MAPmAKER. Specifically, we aim to answer the following research questions: **RQ1:** *How does MAPmAKER help planning in partially known environments?* **RQ2:** *How does the employed decentralized algorithm help in plan computation?*

Implementation. As a proof of concepts we implemented MAPmAKER as a Matlab application. It takes as input the model of the robot application to be considered and runs the procedure described in Section 5. The source code of MAPmAKER, a complete replication package and a set of video showing MAPmAKER in action can be found at https://goo.gl/GY7ZrG.

RQ1. To answer RQ1 we analyzed the behavior of MAPmAKER on a set of *simulated* models.

Methodology. We had considered a set of existing examples proposed in literature and simulated the presence of partial knowledge about the robot application. We performed different experiments in which we evaluated the impact of partial information about the execution of transitions (*Experiment 1*), services provisioning (*Experiment 2*) and meeting capabilities (*Experiment 3*) on the planning procedure. In each of these experiments we considered a partial robot application obtained from models of real robot applications, i.e., the RoboCup Logistics League competition [26] and an apartment of a large residential facility for senior citizens [53]. 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 a possible plan that is executed by the robots. As the robots explore the environment in which they operate (by following possible plans), *true* or *false* evidence about partial information is detected meaning that a transition, service, and meeting capability is detected to be firable, provided, and possible, respectively. Whenever 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 a *definitive* plan.

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 returns a *definitive* plan. 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 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. 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.

Experimental inputs. We have chosen two different examples proposed in literature to evaluate similar algorithms and then abstracted them to obtain partial robot applications.

Example 1. The model of the robot application is obtained from the one used in the RoboCup Logistics League competition [26]. We considered the same map used in the competition. The map has 169 cells and 4 rooms.

Example 2. The model of the robot application is defined on an apartment of about 80 m^2, which is part of a large residential facility for senior citizens [53]. This model has been already used to evaluate a planning algorithm for a single robot based on information contained in RFID tags present in the environment [27].

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 but 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. To simulate the presence of partial information we constructed a partial robot application that conforms with Definition 3.1. To analyze partial information in the execution of transitions (*Experiment 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 (*Experiment 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

			Example 1												Example 2											
			Experiment 1				Experiment 2				Experiment 3				Experiment 1				Experiment 2				Experiment 3			
ID	*I*	*C*	*F*	*T*	$\mathcal{T}_r$	$\mathcal{L}_r$	*F*	*T*	$\mathcal{T}_r$	$\mathcal{L}_r$	*F*	*T*	$\mathcal{T}_r$	$\mathcal{L}_r$	*F*	*T*	$\mathcal{T}_r$	$\mathcal{L}_r$	*F*	*T*	$\mathcal{T}_r$	$\mathcal{L}_r$	*F*	*T*	$\mathcal{T}_r$	$\mathcal{L}_r$
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

Table 1: Results of Experiments 1, 2, and 3 for Examples 1 and 2. Each row contains a different configuration. Columns *F* and *T* contains the number of time false and true evidence about partial knowledge is detected. Columns $\mathcal{T}_r$ and $\mathcal{L}_r$ contain the ratio between the time and the length of the plans computed in steps 1 and 2.

procedure. To simulate partial information about the meeting capabilities (*Experiment 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. Specifically, the following missions were considered: (1) robot r_1 must achieve the mission $\mathsf{F}(s_1 \wedge (\mathsf{F}(s_2 \vee s_3)))$. It had to reach a predefined destination where service s_1 is provided, then to perform either service s_2 or service s_3. This is a variation of the sequencing pattern [31, 61], which requires to first visit a location where the service s_1 is provided and then visit a location where the service s_2 or s_3 is provided. (2) robot r_2 must achieve the following mission $\mathsf{G}(\mathsf{F}(s_4 \vee s_5))$. 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. This is a variation of the surveillance pattern [31, 61], which requires to keep visiting locations where service s_4 or s_5 is provided. The transitions labeled with the actions associated with the added services have been positioned randomly. Meeting between robots r_1 and r_2 is placed in the locations where service s_1 is provided.

Our simulation scenarios were obtained by considering 3 different initial conditions (indicated as I_1, I_2, and I_3) where robots were initially located in different cells. We considered three different models (indicated as C_1, C_2, and C_3) of the partial robot application obtained by making different choices about partial information. Each model analyzed in our experiments is associated with an identifier (*ID*) and is obtained by considering a model of the partial robot application in one of the initial conditions.

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.

Example 1. Four cases are identified. For configurations with ID 2 for Experiment 1, with IDs 1, 4, 5 and 6 for Experiment 2 and with IDs 2, 5, 6, 8 and 9 for Experiment 3 the plans computed in Step 1 were shorter than the one computed in Step 2 (Case 1). For configurations with IDs 1, 3, 4 and 6 for Experiment 1, with IDs 3 for Experiment 2 and with ID 3 for Experiment 3 the plans computed in Step 1 were longer than the one computed in Step 2 (Case 2). For configurations with IDs 7 and 9 for Experiment 1, with IDs 2, 7 and 8 for Experiment 2 and with IDs 1, 4, 7 and 9 for Experiment 3 the plans computed in Step 1 correspond with the one computed in Step 2 (Case 3). Finally, for configurations with IDs 5 and 8 for Experiment 1 plans were found using the procedure outlined in Step 1, while no plans were obtained in Step 2 (Case 4). For this reason they are marked with a – since no comparison was possible.

Example 2. Four cases are identified. For configurations with ID 9 for Experiment 1, with IDs 3, 4, 8 and 9 for Experiment 2 and with IDs 8 and 9 for Experiment 3 the plans computed in Step 1 were shorter than the one computed in Step 2 (Case 1). For configurations with IDs 3 and 6 for Experiment 1, with IDs 1, 6 and 7 for Experiment 2 and with IDs 1 for Experiment 3 the plans computed in Step 1 were longer than the one computed in Step 2 (Case 2). For configurations with ID 2 for Experiment 1, with IDs 2 and 5 for Experiment 2 and with IDs 2, 3, 4, 5, 6 and 7 for Experiment 3 the plans computed in Step 1 correspond with the one computed in Step 2 (Case 3). For configurations with IDs 1, 4, 5, 7 and 8 for Experiment 1 plans were found using the procedure outlined in Step 1, while no plans were obtained in Step 2 (Case 4).

Discussion. The results show that MAPmAKER is effective whenever it computes a possible plan, and during its execution a *true* evidence about partial information is detected (Case 1). In several cases, the generated plans had the same length. This occurred when no partial information was involved in the plans computed by MAPmAKER (Case 3). Furthermore, 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. If *true* evidence about partial information is detected during the plan execution, the mission is achieved. Finally, the detection of a *false* evidence causes a decrease in the effectiveness of MAPmAKER

(Case 2). It happens due to the need of recomputing the plans to be followed by the robots. Thus, the plan executed in Step 1 has a higher length than the one executed in Step 2. This is a price that needs to be paid to have the benefits described in Cases 1 and 4. To alleviate this problem, more complex procedures can be employed. These procedures may consider for example the likelihood of a transition, a service, or a meeting capability to be actually present in the real model. Note that 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 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. More efficient planners, which rely on symbolic techniques ([47]) and abstraction ([12]) can be used to increase performance.

RQ2. To answer RQ2 we analyzed the behavior of the decentralized procedure.

Methodology and experimental inputs. We had considered the set of partial models previously described. We added an additional robot, i.e., robot r_3, which must achieve the following mission $\mathsf{G}(\mathsf{F}(s_6 \vee s_7))$ 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. We analyzed three threats to validity: construct validity, internal validity, and external validity. The random identification of elements that are considered uncertain in the partial robot application models is a threat to *construct validity* since it may generate not realistic models. To mitigate this threat we have constrained how partial information is randomly generated, e.g., by ensuring that partial information about transitions is added in correspondence with an exit and a wall. Biases in the creation of models is threat to *internal validity*. To avoid this threat we had considered real models used in the RoboCup Logistics League competition [26] and representing a part of a large residential facility for senior citizens [53]. The limited number of examples that have been analyzed are a threat to *external validity*. To mitigate this threat, we verify that as possible plans were executed, both true and false evidence about partial information was detected.

7 RELATED WORK

Decentralized solutions. Decentralized planning problem has been studied for known environments [24, 52, 55]. However, planners for partially known environments do not usually employ decentralized solutions [15, 18, 50].

Dealing with partial knowledge in planning. Planning in partially known environments is handled in different ways. (1) Several works (e.g., [3, 6, 11, 16, 18, 32, 44, 46, 51, 58]) consider probabilities within the planning algorithm. Most of these works treat partial information by modeling the robotic application using some form of *Markov decision processes* (MDP). In some of these works [11, 16] transitions of the robots are associated with probabilities which indicate the probability of reaching the destination of the transition given that an action is performed. In other works [58], transition probabilities are not exactly known but are known to belong to a given uncertainty sets. Finally, several works [32, 51] consider partially observable Markov decision processes. All these approaches generally generate plans that maximize the worst-case probability of satisfying a mission. Differently, our work does not consider probabilities. (2) Several works (e.g, [14, 33, 35, 37, 45]) studied how to change the planned trajectories when unknown obstacles are detected or when obstacles move in a unpredictable way. In this case, the used underlying model is some sort of *hybrid model*, i.e., models in which finite state machines are combined with differential equations. In [33], to plan trajectories the authors use a high-level planner that exploits an abstraction of the hybrid system and the mission to compute high-level plans. The low-level planner uses the dynamics of the hybrid system and the suggested high-level plans to explore the state space for feasible solutions. Every time an unknown obstacle is encountered, the high-level planner modifies the coarse high-level plan online by accounting for the geometry of the discovered obstacle. Within this framework, MAPmAKER can be considered as a high-level planner that is able to use an abstraction of the hybrid system that contains partial information, i.e., encode unknown obstacles. (3) Some approaches analyzed how to update plans when new information about known model of a robotic application is detected (e.g., [24]). Differently, in our approach portions of the model of the robotic application are partially known, partial knowledge is reduced as true and false evidence about partial information is detected. Other works (e.g., [1]), aim at detecting how to explore totally unknown environments. (4) Plan synthesis is a particular instance of controller synthesis. Controller synthesis (e.g., [9, 17]) aims at finding a component, usually indicated as controller or supervisor, that ensures property satisfaction for all the possible system executions. Differently, plan synthesis aims at finding a single execution, i.e., a plan that ensures property satisfaction. The controller synthesis is usually ([11, 25, 31, 37, 59]) performed by solving a two player game between robots and their environment. The goal is to find a strategy the robots can use that allows always winning the game. Differently, in our case the planning algorithm ensures that there is a way of completing the *single* (possible) plan that satisfies the property of interest. (5) MAPmAKER can be classified on the boundary between reactive synthesis [11, 37, 54] techniques and iterative planning [25, 40]. As reactive synthesis techniques, MAPmAKER constructs a control strategy that accounts for every possible variation in the environment, but the computed plan does not allow always winning the *two player game* between the robots and their environment. As iterative planning, a new plan is computed on-the-fly when new information is available.

8 CONCLUSIONS

This work presented MAPmAKER, a novel decentralized planner for partially known environments. MAPmAKER solves the decentralized planning problem when partial robot applications are analyzed. We evaluated MAPmAKER by considering the robot application model of the RoboCup Logistics League competition [26] and an apartment of about 80 m^2, which is part of a large residential facility for senior citizens [53]. The results show that the effectiveness of MAPmAKER is triggered when the computed possible plans are actually executable in the real model of the robotic application. Furthermore, in several cases, MAPmAKER was able to achieve missions that could not be completed by classical planners.

Future work and research directions include (1) the study of appropriate policies to select between definitive and possible plans, (2) the use of more efficient planners to speed up plan computation. These may be based for example on symbolic techniques.

REFERENCES

[1] B. C. Akdeniz and H. I. Bozma. 2015. Exploration and topological map building in unknown environments. In *IEEE International Conference on Robotics and Automation (ICRA)*. 1079–1084.

[2] Aws Albarghouthi, Arie Gurfinkel, and Marsha Chechik. 2012. From under-approximations to over-approximations and back. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 157–172.

[3] C. Amato, G. Konidaris, G. Cruz, C. A. Maynor, J. P. How, and L. P. Kaelbling. 2015. Planning for decentralized control of multiple robots under uncertainty. In *IEEE International Conference on Robotics and Automation (ICRA)*. 1241–1248.

[4] Amit Bhatia, Lydia E Kavraki, and Moshe Y Vardi. 2010. Motion planning with hybrid dynamics and temporal goals. In *Conference on Decision and Control (CDC)*. IEEE, 1108–1115.

[5] Amit Bhatia, Lydia E Kavraki, and Moshe Y Vardi. 2010. Sampling-based motion planning with temporal goals. In *International Conference on Robotics and Automation (ICRA)*. IEEE, 2689–2696.

[6] S. Bhattacharya, R. Ghrist, and V. Kumar. 2015. Persistent Homology for Path Planning in Uncertain Environments. *IEEE Transactions on Robotics* 31, 3 (2015), 578–590.

[7] Glenn Bruns and Patrice Godefroid. 1999. Model checking partial state spaces with 3-valued temporal logics. In *International Conference on Computer Aided Verification*. Springer, 274–287.

[8] Glenn Bruns and Patrice Godefroid. 2000. Generalized Model Checking: Reasoning About Partial State Spaces. In *International Conference on Concurrency Theory*. Springer, 168–182.

[9] Christos G Cassandras and Stephane Lafortune. 2009. *Introduction to discrete event systems*. Springer Science & Business Media.

[10] Marsha Chechik, Benet Devereux, Steve Easterbrook, and Arie Gurfinkel. 2004. Multi-valued symbolic model-checking. *ACM Transactions on Software Engineering and Methodology* 12, 4 (2004), 1–38.

[11] Yushan Chen, Jana Tumová, and Calin Belta. 2012. LTL robot motion control based on automata learning of environmental dynamics. In *International Conference on Robotics and Automation (ICRA)*. IEEE, 5177–5182.

[12] Jens Christensen. 1991. Automatic abstraction in planning. (1991).

[13] Edmund M Clarke, Orna Grumberg, and Doron Peled. 1999. *Model checking*. MIT press.

[14] A. G. Cunningham, E. Galceran, R. M. Eustice, and E. Olson. 2015. MPDM: Multipolicy decision-making in dynamic, uncertain environments for autonomous driving. In *International Conference on Robotics and Automation (ICRA)*. 1670–1677.

[15] Jonathan F Diaz, Alexander Stoytchev, and Ronald C Arkin. 2001. Exploring Unknown Structured Environments.. In *FLAIRS Conference*. AAAI Press, 145–149.

[16] Xu Chu Dennis Ding, Stephen L Smith, Calin Belta, and Daniela Rus. 2011. LTL control in uncertain environments with probabilistic satisfaction guarantees. *IFAC Proceedings Volumes* 44, 1 (2011), 3515–3520.

[17] Nicolás D'ippolito, Victor Braberman, Nir Piterman, and Sebastián Uchitel. 2013. Synthesizing Nonanomalous Event-based Controllers for Liveness Goals. *ACM Trans. Softw. Eng. Methodol.* 22, 1 (2013).

[18] Noel E Du Toit and Joel W Burdick. 2012. Robot motion planning in dynamic, uncertain environments. *IEEE Transactions on Robotics* 28, 1 (2012), 101–115.

[19] Georgios E Fainekos, Antoine Girard, Hadas Kress-Gazit, and George J Pappas. 2009. Temporal logic motion planning for dynamic robots. *Automatica* 45, 2 (2009), 343–352.

[20] Georgios E Fainekos, Hadas Kress-Gazit, and George J Pappas. 2005. Temporal logic motion planning for mobile robots. In *International Conference on Robotics and Automation (ICRA)*. IEEE, 2020–2025.

[21] Michais Famelis, Rick Salay, and Marsha Chechik. 2012. Partial models: Towards modeling and reasoning with uncertainty. In *International Conference on Software Engineering*. IEEE, 573–583.

[22] P. Godefroid and M. Huth. 2005. Model checking vs. generalized model checking: semantic minimizations for temporal logics. In *Logic in Computer Science*. IEEE Computer Society, 158–167.

[23] Patrice Godefroid and Nir Piterman. 2011. LTL generalized model checking revisited. *International journal on software tools for technology transfer* 13, 6 (2011), 571–584.

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

[25] Meng Guo, Karl H Johansson, and Dimos V Dimarogonas. 2013. Revising motion planning under linear temporal logic specifications in partially known workspaces. In *International Conference on Robotics and Automation (ICRA)*. IEEE, 5025–5032.

[26] Christian Deppe Ulrich Karras, Tobias Neumann, Tim Niemueller Alain Rohr, Wataru Uemura, Daniel Ewert, Nils Harder, Sören Jentzsch, Nicolas Meier, and Sebastianyear Reuter. 2016. RoboCup Logistics League Rules and Regulations. (2016).

[27] Ali Abdul Khaliq and Alessandro Saffiotti. 2015. Stigmergy at work: Planning and navigation for a service robot on an RFID floor. In *International Conference on Robotics and Automation (ICRA)*. IEEE, 1085–1092.

[28] Marius Kloetzer and Calin Belta. 2008. A fully automated framework for control of linear systems from temporal logic specifications. *IEEE Trans. Automat. Control* 53, 1 (2008), 287–297.

[29] Marius Kloetzer, Xu Chu Ding, and Calin Belta. 2011. Multi-robot deployment from LTL specifications with reduced communication. In *Conference on Decision and Control and European Control Conference (CDC-ECC)*. IEEE, 4867–4872.

[30] Hadas Kress-Gazit, Georgios E Fainekos, and George J Pappas. 2007. Where's waldo? sensor-based temporal logic motion planning. In *International Conference on Robotics and Automation*. IEEE, 3116–3121.

[31] Hadas Kress-Gazit, Georgios E Fainekos, and George J Pappas. 2009. Temporal-logic-based reactive mission and motion planning. *IEEE transactions on robotics* 25, 6 (2009), 1370–1381.

[32] Hanna Kurniawati, Yanzhu Du, David Hsu, and Wee Sun Lee. 2011. Motion planning under uncertainty for robotic tasks with long time horizons. *The International Journal of Robotics Research* 30, 3 (2011), 308–323.

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

[34] Jean-Claude Latombe. 2012. *Robot motion planning*. Vol. 124. Springer.

[35] Bruno L'Espérance and Kamal Gupta. 2014. Safety hierarchy for planning with time constraints in unknown dynamic environments. *Transactions on Robotics* 30, 6 (2014), 1398–1411.

[36] Emmanuel Letier, Jeff Kramer, Jeff Magee, and Sebastian Uchitel. 2008. Deriving event-based transition systems from goal-oriented requirements models. *Automated Software Engineering* 15, 2, 175–206.

[37] Scott C Livingston, Richard M Murray, and Joel W Burdick. 2012. Backtracking temporal logic synthesis for uncertain environments. In *International Conference on Robotics and Automation (ICRA)*. IEEE, 5163–5170.

[38] Sara Ljungblad and Lars Erik Holmquist. 2005. Designing robot applications for everyday environments. In *Joint conference on Smart objects and ambient intelligence: innovative context-aware services: usages and technologies*. ACM, 65–68.

[39] Savvas G Loizou and Kostas J Kyriakopoulos. 2005. Automated planning of motion tasks for multi-robot systems. In *Conference on Decision and Control and European Control Conference (CDC-ECC)*. IEEE, 78–83.

[40] Matthew R Maly, Morteza Lahijanian, Lydia E Kavraki, Hadas Kress-Gazit, and Moshe Y Vardi. 2013. Iterative temporal motion planning for hybrid systems in partially unknown environments. In *International conference on Hybrid systems: computation and control*. ACM, 353–362.

[41] Claudio Menghi, Paola Spoletini, and Carlo Ghezzi. 2016. Dealing with Incompleteness in Automata-Based Model Checking. In *Formal Methods*, Vol. 9995. Springer, 531–550.

[42] Claudio Menghi, Paola Spoletini, and Carlo Ghezzi. 2017. COVER: Change-based Goal Verifier and Reasoner.. In *REFSQ Workshops*. Springer.

[43] Claudio Menghi, Paola Spoletini, and Carlo Ghezzi. 2017. Integrating Goal Model Analysis with Iterative Design. In *International Working Conference on Requirements Engineering: Foundation for Software Quality*. Springer, 112–128.

[44] Venkatraman Narayanan and Maxim Likhachev. 2015. Task-oriented planning for manipulating articulated mechanisms under model uncertainty. In *International Conference on Robotics and Automation (ICRA)*. IEEE, 3095–3101.

[45] Xinkun Nie, Lawson LS Wong, and Leslie Pack Kaelbling. 2016. Searching for physical objects in partially known environments. In *International Conference on Robotics and Automation (ICRA)*. IEEE, 5403–5410.
[46] Alexandros Nikou, Jana Tumova, and Dimos D Dimarogonas. 2017. Probabilistic Plan Synthesis for Coupled Multi-Agent Systems. *arXiv preprint arXiv:1704.01432* (2017).
[47] Fabio Patrizi, Nir Lipovetzky, Giuseppe De Giacomo, and Hector Geffner. 2011. Computing infinite plans for LTL goals using a classical planner. In *IJCAI*. 2003–2008.
[48] Amir Pnueli. 1977. The temporal logic of programs. In *Foundations of Computer Science*. IEEE, 46–57.
[49] Michael Melholt Quottrup, Thomas Bak, and RI Zamanabadi. 2004. Multi-robot planning: A timed automata approach. In *International Conference on Robotics and Automation*, Vol. 5. IEEE, 4417–4422.
[50] Nicholas Roy, Geoffrey Gordon, and Sebastian Thrun. 2006. Planning under uncertainty for reliable health care robotics. In *Field and Service Robotics*. Springer, 417–426.
[51] Nicholas Roy, Geoffrey Gordon, and Sebastian Thrun. 2006. *Planning under Uncertainty for Reliable Health Care Robotics*. Springer, 417–426.
[52] Philipp Schillinger, Mathias Bürger, and Dimos Dimarogonas. 2016. Decomposition of Finite LTL Specifications for Efficient Multi-Agent Planning. In *International Symposium on Distributed Autonomous Robotic Systems*.
[53] The Angen research and innovation apartment: official website 2014. (2014). http://angeninnovation.se
[54] Wolfgang Thomas et al. 2002. *Automata, logics, and infinite games: a guide to current research*. Vol. 2500. Springer Science & Business Media.
[55] Jana Tumova and Dimos V Dimarogonas. 2016. Multi-agent planning under local LTL specifications and event-based synchronization. *Automatica* 70 (2016), 239–248.
[56] Sebastian Uchitel, Dalal Alrajeh, Shoham Ben-David, Victor Braberman, Marsha Chechik, Guido De Caso, Nicolas D'Ippolito, Dario Fischbein, Diego Garbervetsky, Jeff Kramer, et al. 2013. Supporting incremental behaviour model elaboration. *Computer Science-Research and Development* 28, 4 (2013), 279–293.
[57] Sebastian Uchitel, Greg Brunet, and Marsha Chechik. 2009. Synthesis of partial behavior models from properties and scenarios. *IEEE Transactions on Software Engineering* 35, 3 (2009), 384–406.
[58] Eric M Wolff, Ufuk Topcu, and Richard M Murray. 2012. Robust control of uncertain Markov decision processes with temporal logic specifications. In *Annual Conference on Decision and Control (CDC)*. IEEE, 3372–3379.
[59] Tichakorn Wongpiromsarn, Ufuk Topcu, and Richard M Murray. 2009. Receding horizon temporal logic planning for dynamical systems. In *Conference on Decision and Control*. IEEE, 5997–6004.
[60] Tichakorn Wongpiromsarn, Ufuk Topcu, and Richard M Murray. 2010. Receding horizon control for temporal logic specifications. In *International conference on Hybrid systems: computation and control*. ACM, 101–110.
[61] Chanyeol Yoo, Robert Fitch, and Salah Sukkarieh. 2016. Online task planning and control for fuel-constrained aerial robots in wind fields. *The International Journal of Robotics Research* 35, 5 (2016), 438–453.