Hybrid planning and distributed iterative repair for multi-robot missions with communication losses

Patrick Bechon, Charles Lesire, Magali Barbieri

Autonomous Robots : Special Issue on Multi-Robot and Multi-Agent Systems manuscript No.
(will be inserted by the editor)

Hybrid planning and distributed iterative repair for multi-robot missions with communication losses

Patrick Bechon · Charles Lesire · Magali Barbier

Received: date / Accepted: date

Abstract This paper presents a planning and execution architecture suited for the initial planning, the execution and the on-board repair of a plan for a multi-robot mission. The team as a whole must accomplish its mission while dealing with online events such as robots breaking down, new objectives for the team, late actions and intermittent communications. We have chosen a "plan then repair" approach where an initial plan is computed offline and updated online whenever disruptive events happen. We have defined an hybrid planner that mixes Partial Order Planning (POP) with a Hierarchical Task Network (HTN)-based modelling of actions. This planner, called HiPOP for Hierarchical Partial-Order Planner, computes plans with temporal flexibility (thus easing its execution) and abstract actions (thus easing the repair process). It uses a symbolic representation of the world and has been extended with geometrical reasoning to adapt to multi-robots missions. Plans are executed in a distributed way: each robot is responsible of executing its own actions, and to propagate delays in its local plan, taking benefit from the temporal flexibility of the plan. When an inconsistency or a failure arises, a distributed repair algorithm based on HiPOP is used to repair the plan, by iteratively removing actions in the plan in order to amend the global plan. This repair is done onboard one of the robot of the team, and takes care of partial communication. The whole architecture has been evaluated through several benchmarks, statistical simulations, and field experiments involving 8 robots.

DCSD - Université Fédérale de Toulouse Midi-Pyrénées - ONERA
2 avenue Edouard Belin
FR-31055 TOULOUSE CEDEX 4
E-mail: charles.lesire@onera.fr, magali.barbier@onera.fr

Keywords Multi-robot missions · Hybrid planning · Plan repair

1 Introduction

1.1 Context

Multi-robot missions are a way to widen the type of mission that a robotic system can achieve with respect to a single-robot mission. When the robots have different and complementary capabilities, the team can achieve together what individual robots cannot achieve on their own [46]. Ground robots for instance can see under the trees and carry heavy loads whereas aerial robots are faster and can survey larger areas. A team of both types of robots can thus survey large areas scattered with trees, or aerial robots can send ground robots to locations where they have detected an anomaly. In this work, we are interested in multi-robot missions that involve heterogenous robots, that are deployed in outdoor fields. Such missions have gained a high interest in the past years, through several projects targeting search-and-rescue missions after disasters [15] or in extreme situations [39], or information gathering after disasters (e.g., industrial disasters [52] or wreck leakage [13]).

To achieve this type of missions, the robots must cooperate and we argue that this cooperation should be autonomous. This autonomous cooperation is especially useful to face and tackle the constraints that arise from such missions:

- **uncertainties** related to the environment (presence of obstacles, location of injured persons, …) or to the realisation of the mission (time needed to perform a task, imperfect observations);

- **failures** that may reduce the robots' capabilities (some terrain may become impassable, some observation may become impossible), possibly leading to a robot out-of-service;
- **communication**-specific failures or uncertainties, that may make the robots unable to communicate with each-other or with the operation center;
- **operator interactions**, first by taking operator constraints into account before the mission to build a satisfactory plan; and second by providing supervision and interaction means to the operator during the mission execution (regular reports on robots' status, capability for the operator to modify the team objectives, …).

1.2 Case study

In this work, we consider a surveillance mission involving twelve (12) aerial and ground robots. This mission has been defined in the ACTION project[1]. Its objective is to deploy the team of robots to patrol a given zone and detect possible intruders. Upon detection, one or several robots may follow the detected target while the others carry on with the patrolling stage. This patrolling is defined by the objective of observing a set of points of interest, scattered in the zone. The objective is then to allocate observation tasks to the robots, according to their capability to observe a specific point of interest. To perform these observations, movement tasks have also to be computed and allocated. Finally, communication tasks will also be planned to ensure regular reports from robots to the operator. Communication availability is based on the type of robots and the distance between them.

In this mission, the following constraints hold:

- *uncertainty* about the presence and location of intruders; the robots have to adapt to possible detections;
- *uncertainty* about the duration of actions, especially for movements, or when waiting for another robot at a communication point;
- *failure* of a robot, making it unavailable for the rest of the mission;
- unavailability of *communications*, during meetings or when a plan repair is necessary;
- advised or forbidden zones on the area, depending on the type of robot, based on the *operator* prior knowledge and safety regulations;
- information from the *operator*, that can decide to get a robot out of the patrolling stage in order to follow a detected intruder, or to make a robot available again for the patrolling.

In our case study, the operator defines, before the beginning of the mission, the set of points of interest, the set of mandatory meeting between robots, the area where each robot (or robot type) is allowed to go and a set of patrol route for each robot. During the mission, the operator is presented with the current plan of each robot, updated in real time. He is also notified at each target detection. During the mission, if a target is detected for instance, the operator can decide to allocate zero, one or two robots to its tracking. He can also remove robots from the plan (in case of a robot failure for instance) or remove some goals (for instance remove some point of interest or some mandatory communication).

2 Related works

2.1 Architecture for distributed execution of a plan

Several architectures have been proposed to execute of a mission with multiple robots, with different methods to deal with the communication and the synchronization between robots. Amigoni et al [2] distinguish several types of connection requirements: *none* (robots are not required to communicate), *event-based* (robots must communicate on particular events), and *continuous* (robots must always form a connected graph). Some works have been done in case no connection is required, hence providing architecture unaffected by communication mistakes. For instance Khadka et al [35] have studied how the experience of some robots can influence and help other robots to perform tasks. It uses team-level memory to develop multi-robot coordination strategies off-line.

Most of the works are in the case of *continuous connectivity*. Some works assume this connectivity will be available by an external mean. This allows every robot to send back data or videos and a central planner to update the plan. Portugal and Rocha [51] have proposed a distributed architecture with full communication and use bayesian planning for more random patrols. Franchi et al [25] developed a method for decentralized control of a swarm of robot for the entrapment of a target while Dames [16] used a probabilistic filter to estimate the number of targets and the areas more likely to compute the best actions for each robot. Moarref and Kress-Gazit [41] proposed an automatic process to synthesize

[1] ACTION was a project of ONERA and LAAS-CNRS, funded by the French Procurement Agency. The work described here was used for the final demonstration of the project. For more information, see `http://action.onera.fr/`

decentralized controllers from the desired collective behavior. Communication between robots is used for the synchronization of those controllers. Garzón et al [26] have worked on multi-robot missions for signal searching while trying to minimize bandwidth. But during the real life experiment, the communication graph is complete. Other works use planning to ensure this network connectivity. For instance Pei and Mutka [47] have used some robots to explore while other are able to be a communication relay. Abichandani et al [1] provided a mathematical formulation to allow those global communication constraints to be integrated into a Mixed Integer Nonlinear Planning framework. Penumarthi et al [49] described a method to construct a communication map based during a first part of the mission with multiple robots, allowing communication aware planning afterward. In a real-life setting, using a routing algorithm can emulate this full communication graph if the communication graph is connected.

Works in the case of *event-based connectivity* mostly use distributed architectures, where each robot manages its own local plan and must synchronize with others. In practice, it usually requires the communication graph to be connected, as there is no model of when the events occur, and hence no robot or group of robot can be isolated from the rest of the team. Robots can use auction to distribute their actions at execution based on the current situation, even including task with deadlines [38]. Barbulescu et al [6] specifically dealt with adapting local plans to ensure future deadline constraints between robots. Arrichiello et al [3] have studied the fault detection inside a team of robots. They do not assume a complete communication graph but a static graph for its real-life experiments. Banfi et al [5] have used the concept of recurrent connectivity where the planing is centralized and the robots must connect back to the central planner only when a new information is available, without needing a full communication graph.

Finally, few works are built to be robust to more stringent communication constraints, and can then manage temporary abscence of communication. Lesire et al [37] describe a distributed execution algorithm for a hierarchical plan able to repair it in case of failure or unexpected events, even with intermittent communication. Otte et al [45] have studied an auction algorithm when communication are lossy and Ponda et al [50] have studied an auction algorithm with a dynamic topology of the communication graph. Brinon-Arranz et al [13] developed an architecture geared toward underwater robots when robots adapt their actions to their neighbors and are robust to communication delays.

In our case, missions require *event-based connectivity*, where communications are needed when disruptive events occur, in order to repair the plan. We are then interested in architectures that can deal with loss of communication but where the operator can have an understanding of the current situation and provide specific instruction when needed. To do that we decided to plan before the mission and update the plan when needed as in [37].

2.2 Planning

Automated Planning is a very active area of research, with diverse algorithms and methods [42], due to the complexity of most of the planning problems (usually NP-hard [21]). Domain-independent planners have tried to tackle this complexity issue by looking for smart heuristics, while some problems have been solved by integrating expert knowledge into domain-dependent approaches.

2.2.1 Domain-independent planning

First approaches in domain-independent planning have started with STRIPS planning [22] (for STanford Research Institute Problem Solver), and are known as *state-space search* approaches. The idea is to represent the world as a set of boolean literals and a plan as a succession of actions. For instance the presence of a robot at a given point or the visibility between two points can be represented as boolean literals. An action will modify a set of literals representing a state of the world to another set of literals. The search is then defined as an A^* algorithm [29] for which a heuristics is needed to estimate the remaining cost between the current plan and the goal. There is a lot of variety for those heuristics, so we cannot describe all of them in detail here but we will give some pointers to different families. In state-space planning, heuristics are usually based on the estimation of the cost to a given state, for instance using landmarks-based heuristics [33, 53].

Graph-based search is another planning paradigm, represented by the well-known GraphPlan [11] algorithm. GraphPlan is a planner where a relaxation of the planning problem is used as a heuristic for the general planning problem. This work has lead to the definition of the h^{add} heuristic that is used later in this paper (see 4.2.3 for the definition). This heuristic can be used in state-space algorithms like YAHSP [57]. More recently, other heuristics have been defined to improve the performances of domain-independent algorithms, like h^{ff} [32] and the h^m family [31].

The third paradigm of domain-independent planning is called *plan-space search*, or *Partial-Order Planning* (POP). Plans are represented as a set of actions with constraints between them. This allows to represent plans where all the actions are not sequentially ordered. For instance, in a plan with several agents, this allows each agent to have actions that are only loosely tighten to the other agents. In addition, this also allows to produce temporally flexible plans: the plan can be easily adapted to a longer or shorter action (or at least the consistency of this change can be checked). Several implementations of POP have been proposed such that UCPOP (Universal quantification and Conditional effects Partial Order Planner) [48] or VHPOP (Versatile Heuristic Partial Order Planner)[59]. Another benefit of plan-space search is that the plan contains elements of justification on why each action is part of the plan, which helps to explain it to a human [9] and to mix two plans computed independently [30]. A drawback of this method is that until the plan is complete, there is no full state available in the plan to compute a heuristic on, leading to computationally heavy planners. Some heuristics do not need a full state (such as h^{add}) and can still be used but this usually means that the planning time is higher than with state-space search.

2.2.2 Domain-dependent planning

Among the most successful domain-dependent planning are Hierarchical Task Networks (HTN) [43]. The idea is to use user-defined abstract actions. Each abstract action is associated with a set of methods, each method being a set of actions which can replace the abstract action in a plan. These actions can be sequential or concurrent. For instance a search action for an helicopter could be described as a sequence of observations that the helicopter should perform in a specific order according to an existing procedure. These abstract actions are described in the problem and the planner uses them to improve the search process and to enforce constraints on the solution (when the solution must be described in term of abstract actions). An abstract action can be composed of abstract actions and of elementary actions, these latter being the low-level actions that the planner sends to the robot for its moves, its perceptions or its communications. The first HTN planners were state-based and they usually did not deal with time constraints and concurrent actions. Some formalisms deal with these conditions, such as [14, 28, 44]. The effectiveness of these planners is highly dependent on the defined abstract actions but can be significantly better than non-hierarchical planners in certain conditions.

2.2.3 Hybrid planning

Approaches that mixes POP and HTN, in a attempt to have the benefits of both, have been studied under the term of *hybrid planning*, for example by including hierarchical planning in a more general framework in UCP [34] or in a constraint satisfaction programming (CSP) planner in CHIMP [56]. Using the PANDA system, Schattenberg [55] conducted an empirical study of several search strategies in hybrid planning. The FAPE system [19] integrates closely planning and acting and do temporal planning with resource constraints. This system is able to repair the plan of a robot when its action fails.

Planning family	Pro	Cons
State-space	- Good heuristics - Domain independent	- All actions are sequential
Graph-based	- Good heuristics - Domain independent	- All actions are sequential
POP	- Temporal and multi-agent reasoning - Domain independent	- Slower than other approaches
HTN	- Fast	- Domain dependant
Hybrid planning	- Temporal and multi-agent reasoning - Fast	- Domain dependant

Table 1 Summary of the strength and weaknesses of each planning family.

2.3 Repairing

We define the repair as solving a planning problem starting with an imperfect plan (i.e. a plan that is not executable or that do not achieve all the goals of the problem). Unlike other approaches [27], we assume that not only the initial conditions and the goals can change but also the actions available to the planner. We want the repairing function to find a solution plan that is "close" to the given initial plan. This property is called *plan stability* [24]. In this section, we specifically discuss repair algorithms for POP and HTN planners.

Local search algorithms can add or remove elements to the current plan. A heuristic is used to get closer to a solution plan. GPG [27] tries first to modify a small portion of the plan then increases the size of this portion if no solution is found. This strategy can also be

used to improve a given plan rather than just repairing a broken plan, as demonstrated in [4]. A drawback of local search is to prevent the algorithm of re-doing something that has already been done: the algorithm should not enter a loop where it adds an element it already removed. To avoid this issue, some techniques starts by removing some elements and after this step they can only add some elements. This is what we call the "unrefine-then-refine" approach. This second phase is similar to a lot of planning techniques and the repair consists then in figuring out what elements to remove. As examples, POPR (Partial Order Plan Repair) [36] is built on top of VHPOP whereas Replan [12] is built on top of an HTN planner.

Another approach is to specify rules to apply (similar to planning where HTN is a way to specify rules). O-Plan [18] for instance has a list of all the events that can happen and of what elements to add to the plan when they happen. But this approach can only deal with disruptive events that are expected and for which there exists methods to solve them.

Instead of using specific rules, the repair algorithm can also replay the reasoning that lead to the initial plan. The same choice is made if the decision is still valid, else the algorithm has to explore the different choices. As examples, the HTN-based RepairSHOP [58] and the PANDA hybrid planner [10] use this approach to repair.

3 Contribution

To allow the achievement of multi-robot missions with communication breakdowns and the occurrence of disturbances, we have then decided to compute an initial deterministic plan offline and to update it when needed during the execution. This update is done in a distributed way: each robot is responsible of executing its own actions, and to adapt the plan in case of delays. When a repair is needed (because of a robot failure, a robot leaving/entering the team, or an unmanageable delay), the repair is made by one of the robots, that can modify only the parts of the plan involving the robots with which it can communicate. This architecture allows to manage plan repairs in a distributed way even with partial communication links.

For the planning approach, we wanted the POP benefits (parallel task for the actions of different robots and easy merging of plans) but we also need to take into account operator interactions (for instance by taking into account their constraints on the plan), explain the plan to the operator and react to failures by repairing the plan. All those constraints advocate for the use of HTN. This is why we have chosen an approach that mixes POP and HTN approaches: hybrid planning.

For the repairing approach, the use of pre-defined rules seems to limit the types of events that can be dealt with. Since we need to be able to react to quite big changes (such as any robot having a complete failure at any given time or a delay of any action that would make the complete plan unfeasible), we also think the reasoning replay approach to be inappropriate because this type of events can invalidate a lot of different choices or choices that were made early in the reasoning process. Our objective being to use as much as possible the same algorithm for planning and repairing, we therefore choose the "unrefine-then-refine" approach over the local search.

Those choices were made to tackle the constraints highlighted in the introduction. The **uncertainties** related to the environment and the **failures** are taken into account by being able to repair the plan at any time if needed. The **uncertainties** related to the time needed to perform a task are mostly taken into account by the temporal flexibility of the plan. If this is not enough and an inconsistency is detected, a repair is triggered. **Intermittent communication** are taken into account by relying only on very few mandatory communication. During the execution no communication is mandatory except at some defined meeting points. During a repair, even if some robots are not reachable, the plan can be repaired (the worst case scenario is that some goals will be temporarily dropped until the communication can be re-established). Finally, the **operator interactions** are taken into account with the definition of abstract action during planning, by being able to display hierarchical plan during the execution, and by taking operator notification into accounts at repair time.

To our knowledge, no existing algorithm satisfies our requirements: repair processes such as POPR [36] or Replan [12] are respectively used in POP and HTN frameworks, but no repair process has been studied for hybrid planners (excepted in PANDA [10], but the approach seems inappropriate to provide stable repaired plans). Our main contribution is then an "unrefine-then-refine" repair algorithm for hybrid planners. The specificity of the repair algorithm is that it is suited to situations where communications are intermittent, and is then able to perform partial repairs. The contribution is emphasized by a sound evaluation on academic benchmarks, and a demonstration on the field involving ground and aerial robots.

This repair algorithm settles on the hybrid planner HiPOP (see section 4). HiPOP main algorithm is a classical hybrid planner algorithm. We however made

some enhancements first in the management of abstract flaws, by introducing concepts such as *conflicts*; and second on the adaptation and improvement of some standard POP heuristic for hybrid planners (see 4.2.3). We moreover proposed a specific management of geometrical reasoning (see 4.3) when such a reasoning can be infered from the problem to solve.

In order to execute a plan computed by HiPOP in a distributed multi-robot system, we have defined a distributed architecture that manages the adaptation of every robot' plan to time delays, possibly taking into accoung other robots' information when available (see section 4.6). Moreover, during execution, several events can happen: actions failures, discovery of new goals, unmet deadlines. In addition, we also assume that the communications can fail at any time. The repair algorithm (see section 5) manages such failures and is suited for partial communication among the team of robots. Finally we have evaluated the whole architecture (execution and repair) in simulation and on a field experiment. We conclude with a discussion of the result and of the future work.

Parts of this work have already been published: the planning algorithm HiPOP [7] and an extension to repair plans [8]. This paper describes in addition the use of geometrical reasoning, how to deal with intermittent communications and evaluations of the whole architecture, both in simulation and in field experiments.

4 Hybrid planning

The first part of this section presents some classical definitions related to the concept of hybrid planning and the generic algorithm of a hybrid planner. The second part focuses on improvements made in HiPOP, regarding first the management of abstract flaws, and the enhancement of existing heuristics. Then the modifications made to integrate geometrical reasoning into this type of symbolic planner are described in a third part. Finally, we present the evaluation of HiPOP on several benchmarks, as well as its application to the surveillance mission.

4.1 Background on hybrid planning

We selected a literal-based description of the world where a state is represented as a set of positive literals. The negation of literal l is noted $\neg l$.

Definition 1 (problem) A problem P is a tuple $(L, A, Init, Goal)$ where L is a set of literals, A is a set of actions (each action being either elementary or abstract), $Init$ is the initial state and $Goal$ is the goal state.

The resulting state of the execution of an elementary action is obtained from a starting state by removing the set of its deletion effects and adding the additional ones.

Definition 2 (elementary action) An elementary action is a tuple $(name, Pre, Add, Del, dur)$ where $name$ is the unique name identifying the action, Pre is a set of literals representing the preconditions, Add is a set of literals representing the addition effects, Del is a set of literals representing the deletion effects and dur is the duration of the action (or a fixed estimation if the duration is not known at the planning time).

For instance the action *move* r_1 a b moves robot r_1 from location a to b. It has for precondition $\{$ *at* r_1 $a\}$, for addition effect $\{$ *at* r_1 $b\}$ and for deletion effect $\{$ *at* r_1 $a\}$. Its duration depends on the distance between a and b and on the speed of the robot. A graphical representation of this action is found in Figure 1.

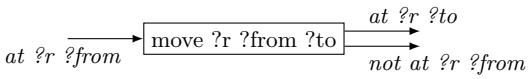


Fig. 1 Graphical representation of a move action. To instanciate this action, each literal that starts with a question mark should be replaced with a literal. This action could be instancited as *move r1 a b* or *move r1 a c*

An abstract action cannot be executed directly and has to be replaced by one of its methods, but can be manipulated during the search as an elementary action.

Definition 3 (abstract action) An abstract action, also called high-level or hierarchical action, is a tuple $(name, Pre, Add, Del, dur, \mathcal{M}, \mathcal{C})$:
- the first five elements $(name, Pre, Add, Del, dur)$ are the same as in an elementary action (Definition 2),
- $\mathcal{M}$ is a set of partial plans (called methods, see Definition 6 below) used to instantiate the action,
- $\mathcal{C}$ is a set of conflicts (see Definition 11 below).

For instance an action to move a parcel from one location to the other has the same description than the action *move* (Figure 1). The methods of this action describe how to actually perform it, for instance by having a robot pick it up, moving the robot and then dropping the parcel.

In a plan, the same action can be performed several times at different dates. A step represents the instantiation of an action in a plan. Several steps in the same plan can instantiate the same action.

Definition 4 (step) A step τ is a tuple $\tau = (a, t_s, t_e)$ where a is an elementary action ($name$, Pre, Add, Del, dur) or an abstract action ($name, Pre, Add, Del, dur, \mathcal{M}, \mathcal{C}$) and t_s and t_e are the indexes of timepoints in a STN (see next paragraph). These timepoints represent the start and end times of τ. We denote $act(\tau) = a$, $t_{start}(\tau) = t_s$, $t_{end}(\tau) = t_e$, $Pre(\tau) = Pre$, $Add(\tau) = Add$, $Del(\tau) = Del$ and $dur(\tau) = dur$. For the elementary actions, $t_e = t_s + dur$.

A Simple Temporal Network (STN) [17] is used to check schedulability over the timepoints. If the set of constraints allows at least one solution, the STN (or equivalently the set of constraints) is said to be *consistent*. Let τ_i, τ_j be steps. $\tau_i \prec \tau_j$ is the shorthand way of $t_{end}(\tau_i) \leq t_{start}(\tau_j)$ meaning that τ_i is scheduled before τ_j.

For a plan to be valid, the preconditions of an action must hold when its execution begins. To enforce this constraint, we introduce *causal links*.

Definition 5 (causal link) A causal link $(\tau_i \xrightarrow{l} \tau_j)$ is a tuple (τ_i, τ_j, l) where τ_i and τ_j are steps, l is a literal such that $l \in Pre(\tau_j) \cap Add(\tau_i)$ and τ_i should be applied before τ_j, i.e. $\tau_i \prec \tau_j$.

If a causal link $(\tau_i \xrightarrow{l} \tau_j)$ is present in the solution, the algorithm will ensure that no step will delete l between τ_i and τ_j and that $\tau_i \prec \tau_j$.

We can then define a plan as a set of steps and constraints between these steps.

Definition 6 (partial plan) A partial plan Π, also called method, is a tuple $(\mathcal{T}, CL, TL, H, Init, Goal)$ where:

- $\mathcal{T}$ is a set of steps (see Definition 4),
- CL is a set of causal links (see Definition 5),
- TL is a set of temporal links, each link (t_s, t_e) meaning $t_s \prec t_e$,
- $H = \{(\tau_i, m_i, \tau_i^0, ..., \tau_i^n)\}$ is a set of hierarchical relations where τ_i is a step associated to an abstract action a_i, m_i is a method of a_i and the τ_i^j are steps of the plan. Each relation means that the steps $\tau_i^0 ... \tau_i^n$ have been introduced when instantiating τ_i with m_i,
- $Init$ is the set of literals true in the initial state,
- $Goal$ is the set of literals to achieve.

Each plan has two special steps: the initial step adds effects corresponding to the initial state and the final step has preconditions being the goal to achieve. When this last step can be executed, the plan in question is a solution to the problem: the goal is achieved at the end of the plan. For the solution plan, $Init$ and $Goal$ are the same as in the problem definition. For the methods of abstract actions, $Init$ is the precondition of the action and $Goal$ is the union of Add and of the negations of literals of Del. Instantiating an abstract action means selecting a method (*i.e.* a partial plan) and adding all its elements (steps, causal links and temporal links) in the plan.

An example of a partial plan is shown in Figure 2. This plan contains three elementary actions, one instantiated abstract action and one uninstantiated abstract action. The abstract actions represent the fact of doing a round-trip to explore a location : at the end of the action the robot is in the same location than at the beginning.

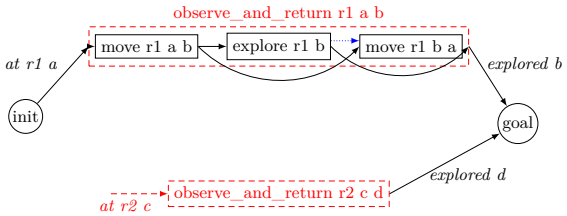


Fig. 2 Example of an hybrid plan with flaws. Black rectangles represent elementary actions. Red dashed rectangles represent abstract actions. Black arrows represent causal links. Blue dotted arrows represent temporal precedence links. Red dashed arrows represent open links.

Given a plan, one can define its flaws. Each flaw is an issue preventing the plan to be executed because a precondition is not guaranteed or because an abstract action is not instantiated.

Definition 7 (open link) An open link $(\xrightarrow{l} \tau_j)$ is a tuple (τ_j, l) where τ_j is a step and l a precondition that is not supported by any causal link i.e. $l \in Pre(\tau_j)$ and $\nexists \tau_i : \tau_i \xrightarrow{l} \tau_j$

In Figure 2, the uninstantiated abstract action has a precondition (`at r2 c`) that is not supported by a causal link. This open link is shown in dashed red.

Definition 8 (threat) A threat is a tuple $(\tau_k, \tau_i \xrightarrow{l} \tau_j)$ where:

- τ_k is a (threatening) step,
- $\tau_i \xrightarrow{l} \tau_j$ is a (threatened) causal link,
- $\tau_i \prec \tau_k \prec \tau_j$ is consistent,
- $l \in Del(\tau_k)$.

A *threat* represents an issue that can limit the schedulability of a set of steps by deleting a precondition of a step before its execution. In Figure 2, without the blue temporal link between `explore r1 b` and `move r1 b a`, the action `move r1 b a` would be allowed to be executed in parallel with `explore r1 b`. In this case the

action `move r1 b a` would threaten the causal link on the robot position between `move r1 a b` and `explore r1 b`. Basically this means that the robot must be in `b` to explore and cannot leave this position until the `explore` action is done.

Definition 9 (abstract flaw) An abstract flaw is a step τ such that no hierarchical relation of τ is present in the plan.

Abstract flaws represent the fact that a non-instantiated action is in the plan. In Figure 2, this action is drawn in red.

Definition 10 (flaw) A flaw is an open link, a threat or an abstract flaw.

Once the flaws are defined, the hybrid planning algorithm is relatively straightforward (see Algorithm 1). To solve problem P (see Definition 1), the algorithm inputs are the actions A of the problem, and an initial partial plan $\mathcal{I}$ containing the initial and goal states. The algorithm keeps a set $\mathcal{P}$ of potential solution plans. At each iteration of the algorithm, a first heuristic chooses which plan Π to expand (function *PopBestPlan* in line 3). A second heuristic chooses which flaw f to solve in this plan (function *PopBestFlaw* in line 7) from the set of flaws of a plan $\mathcal{F}(\Pi)$. For each way to solve this flaw, a new plan is created (function *Resolvers*) and added to the set of plans to explore (line 8). The algorithm stops when a solution plan is found (i.e. no flaw in the plan, line 5) or when no solution is found (i.e. no other plan can be expanded, line 10).

Algorithm 1: Hybrid planner

```
   Input : A (available actions), I (initial plan)
 1 P = {I} ;
 2 while P ≠ ∅ do
 3 |   Π = PopBestPlan(P) ;
 4 |   if F(Π) = ∅ then
 5 |   |   return Π ;
 6 |   end
 7 |   f = PopBestFlaw(F(Π)) ;
 8 |   P = P ∪ Resolvers(A, Π, f) ;
 9 end
10 return ∅
```

Each type of flaw is solved differently:

- Threat: two resolvers are available, one that schedules the threatening step before the causal link and one that schedules it after.
- Open link: a causal link must be added. The new causal link can be from an already existing step or a new step that is introduced in the plan.
- Abstract flaw: each method of the abstract action can be used to instantiate this action. All the elements of the method are introduced into the plan as children of the abstract step and with temporal constraints to schedule them during the abstract step.

4.2 HiPOP

For the requirements of our study, we created a new hybrid planner called HiPOP. HiPOP relies on the description made above of hybrid planning (especially Algorithm 1) with the following additions.

4.2.1 Conflicts in abstract actions

The first improvement is to modify the way the threats of abstract actions are detected. The threats of elementary actions are detected when the action deletes the literal of the causal link. With abstract actions, it is not enough to check only the literals deleted by the action. When the abstract action will be instantiated, several elementary actions could be added in the plan. Those actions may threaten causal links already in the plan, thus making the plan invalid even if the plan with the abstract action seemed valid.

To prevent this, we added in the description of the abstract actions a set of *conflicts*. The conflicts of an action are a set of literals given in the definition of the action such that this action cannot be concurrent with any causal link of one of its conflicts. We note this set of conflicts $\mathcal{C}$ (as in Definition 3). We can then update the definition 8 of a threat by adding a new condition for the last item.

Definition 11 (threat with conflict) A threat is a tuple $(\tau_k, \tau_i \xrightarrow{l} \tau_j)$:

- τ_k is a (threatening) step,
- $\tau_i \xrightarrow{l} \tau_j$ is a (threatened) causal link,
- $\tau_i \prec \tau_k \prec \tau_j$ is consistent,
- $l \in Del(\tau_k)$ or $l \in \mathcal{C}(\tau_k)$.

One way to define conflicts is to add all literals that are deleted in all the methods of a given action. This improvement allows the planner to compute plans with abstract actions that will be valid once its abstract actions will be instantiated without having to solve new threats at each instantiation.

For instance an abstract action can define a patrol for a robot that start and end at the same point. On a high level view, the position of the robot does not change so the action can be scheduled concurrently with a move action for this robot. But as soon as the abstract action is instantiated, the plan will become invalid. By defining the position of the robot as a conflict of the

patrol, we inform the planner that even with the high level view, the plan is invalid.

4.2.2 List of allowed actions

An issue that occurred in the earlier versions of our hybrid planner is the fact that the planner can both add the abstract action then instantiate it and add all the elementary steps themselves. The planner can thus develop several branches that will all lead to the same plan (with or without abstract actions). Moreover, when adding directly the elementary steps, the planner develops plans that do not conform to the operators intent. Indeed, if the operators have described an abstract action to achieve a goal, the priority is to have a plan that includes this action.

To solve this issue, we decided to prevent the planner to add some actions to solve an open link: these actions can only be inserted in the plan as part of the instantiation of an abstract action. For instance if an abstract action has a method containing the matching *load* and *unload* actions, we would prevent the planner to add a *load* or a *unload* action. The planner could only add this abstract action and instantiate it.

We then define a set $\mathcal{A} \subseteq A$ of allowed actions. On Algorithm 1, when the flaw f (line 7) is an open link, the *Resolvers* method can either add a causal link to an existing action or to a new inserted action that can only be part of $\mathcal{A}$. This set of allowed action is defined with the initial problem by the operator.

This could prevent the planner to find some solutions (for instance those that do not respect the operators intent behind the description of the abstract actions) but this is also critical to limit the number of plans searched.

In our case study, we created patrols as abstract actions (a set of observe and move actions) and we forbade the planner to add an observation action alone. But we allow the planner to add move actions for each robot to be able to move from the ending point of a patrol to the starting point of the next one.

4.2.3 HiPOP heuristics

Two heuristic functions are used in the HiPOP algorithm: *PopBestPlan* to choose which plan to expand next and *PopBestFlaw* to choose which flaw to solve next. The performance of the algorithm will be highly dependent on these two heuristics. The heuristics we use in HiPOP are POP heuristics [59] adapted and improved in the case of hybrid planning.

Plan heuristic HiPOP uses the A^* algorithm to sort the set $\mathcal{P}$ of all plans generated but not yet explored. They are stored in increasing order of $f(\Pi) = g(\Pi) + h(\Pi)$ where $g(\Pi)$ is the cost of the current plan and $h(\Pi)$ is a heuristic estimation of the cost to reach a solution plan from Π. In HiPOP $g(\Pi)$ is the minimum temporal makespan of the plan, computed from the STN. The computation of $h(\Pi)$ uses the h^{add} heuristic as described by VHPOP [59]. It assumes a sub-goal independence (which is false, meaning that h^{add} is not admissible) and computes the cost of a plan as the sum of the costs of achieving each open link, where the cost of each literal can be computed offline. To break ties between plans, we use an estimation of the remaining effort (as used in VHPOP). The definition of this effort is roughly the approximate number of iterations the planner has to make to reach the solution. This tie-breaker has been shown to improve performance by exploring plans closer to the solution when the cost is similar [59]. If both the remaining cost and the remaining effort are equal, then the plans can be selected randomly or using a LIFO (Last In First Out, to stay focused on the current goal).

The h^{add} heuristic does not take into account the action reuse, and VHPOP proposed a modification of h^{add} to partially take care of it, called *reuse*. It assumes a cost of 0 for an open link if a step producing the required literal is present before the open link in the plan.

We propose another version of this heuristic called *advance reuse* where the cost of a literal is 0 only if a step produces this literal before in the plan and if no other step removes this literal after its creation. If Π is a plan, $OL(\Pi)$ the set of open link, then we have:

$$h^{add}_{areuse}(\Pi) =$$

$$\sum_{(t,f)\in OL(\Pi)} \begin{cases} 0 & \text{if } \exists \tau \in \mathcal{T}(\Pi) \mid f \in Add(\tau) \wedge \\ & \qquad (\tau \prec t \text{ is possible}) \wedge \\ & \forall \tau_1 \in \mathcal{T}(\Pi) \mid f \in Del(\tau_1) \\ & \qquad \implies (\tau_1 \prec \tau \vee \tau \prec \tau_1) \\ h^{add}(l) & \text{else} \end{cases} \quad (1)$$

Flaw heuristic Previous work on VHPOP showed that solving threats before open links usually led to better results. This could be because focusing on threats allow to detect unfeasible plan early, whereas solving open links is usually always possible in our problems. We tried several suggestions to sort the open links: using their cost (computed with h^{add}) or solving the more recently introduced open links.

VHPOP does not deal with abstract flaws (as they do not exist in POP planning). The idea of hybrid planning is to first compute a valid plan with abstract ac-

tions in it and when such a plan is found to instantiate its abstract actions. So we solve the abstract flaws at the end.

Some abstract actions can "hide" the fact that their methods create low-level literals. For instance an abstract action can have preconditions and effects about the position of a team of robots and have methods with all the move actions for each robot of the team. This means that the action itself will not have preconditions and effects related to the individual position of the robots but its methods will. When several abstract actions are in the plan, this means that we need the instantiation of the previous actions to be able to insert causal links for instantiating an action. This is why we are sorting abstract flaws in chronological order.

In summary, the threats are solved first, then the open links and finally the abstract links. In the rest of this paper, we will identify a flaw heuristic by the method it uses to sort open links. The open links can be solved in decreasing order of h^{add} (called *sorted*) or by decreasing order of h^{add} among the open links of the last added step that still has open links (called *local*). This last heuristics allow the planner to focus on solving one high level goal before considering the others.

4.2.4 Completeness

The completeness of HiPOP highly depends on the available description and hypothesis. For instance literals can be masked by the hierarchical description if we assume that the elementary actions are forbidden. The proof is the same as the completeness of POP planning if we allow the algorithm to plan with abstract and elementary actions without restriction (i.e. if the set of allowed action $\mathcal{A} = A$), but this does not represent the actual use of the algorithm. The search is complete among the space of all plans that can be represented using only allowed actions at the higher level and their instantiation, i.e. among the plans respecting the user intend. In other words, all plans that respect the user intent are searched but not all plan that can be constructed from the elementary actions.

As stated in 2.2, the problem is NP-hard so the complexity of the planner is NP-hard.

4.3 Geometrical reasoning in hybrid planning

The hybrid planner described up to here can solve different problems but is unable to solve a multi-robot exploration problem in a reasonable time. The issue is not the completeness but the efficiency: by using symbolic reasoning HiPOP can have to explore a lot of different invalid plans before realizing that there are invalid.

To solve more efficiently the problems involving geometrical reasoning, we modified HiPOP in two ways. The first one is to detect and use the fact that the position of a given robot is unique. The second one is to detect and use *move actions*: actions whose only effect is to move a robot without any other precondition.

4.3.1 Unicity of the position

The first characteristic of the robots position that is not taken into account in our literal-based description is that a position is unique: one and only one must exist at any time. In some situations, the planner can have open links for the robot to be in different places at the same time and still cannot figure out that the plan is invalid until later, when it will try to solve these open links.

The first step to deal with this issue is to detect the literals that represent a position. More formally we need to detect sets of literals such that one and only one is true at any given time (except during the execution of an action where the position can be undefined, but at the end of the action it has to be defined). From the description of the domain using PDDL (Planning Domain Definition Language) [40], a list of candidate families is thus computed from the predicates. And for each family, it is labelled as a position family if there is one and only one literal of this family in the initial state.

Definition 12 (position literals) Let $P = (L, A, Init, Goal)$ be a problem.

The set of literals $\mathcal{G} \subset L$ is a position family if and only if:

1. $Init \cap \mathcal{G}$ has a single element.
2. $\forall a \in A, \forall f \in \mathcal{G},\ f \in Del(a) \implies f \in Pre(a) \wedge \exists g \in \mathcal{G} \mid g \neq f \wedge g \in Add(a)$.
3. $\forall a \in A, \forall f \in \mathcal{G},\ f \in Add(a) \implies \exists g \in \mathcal{G} \mid g \neq f \wedge g \in Pre(a) \wedge g \in Del(a)$.

This definition ensures that there is a single position at the beginning (Property 1) and that every action will keep one and only one position after its execution. Each action that deletes a position must ensure that this is the current position and that it will set another position (Property 2). Each action that creates a position must delete the previous position (Property 3). The family representing the position of robot `r1` is defined for instance as `at r1 *`, meaning all literals starting with `at r1` and another literal following. For instance, the literal `at r1 pos1` is a position literal in the family `at r1 *`.

Once the algorithm has detected families, HiPOP can use them as a mutex. If an action produces a position literal, the algorithm knows that it cannot be

concurrent with another position. So threats can be detected earlier: if an action produces a position literal then it will threaten any causal link on a different literal from the same family. We can then supplement the definition 11 of a threat by adding a new condition for the last item.

Definition 13 (threat with conflict and position) A threat is a tuple $(\tau_k, \tau_i \xrightarrow{l} \tau_j)$:
- τ_k is a (threatening) step,
- $\tau_i \xrightarrow{l} \tau_j$ is a (threatened) causal link,
- $\tau_i \prec \tau_k \prec \tau_j$ is consistent,
- $l \in Del(\tau_k)$ or $l \in \mathcal{C}(\tau_k)$ or $(\exists \mathcal{G} | l \in \mathcal{G}$ and $\exists p \in Add(\tau_k) | p \in \mathcal{G}$ and $p \neq l)$.

As an example (see Figure 3), this modification allows the planner to detect that the action `move r1 d e` threatens the causal link `move r1 a b` $\xrightarrow{at\ r1\ b}$ `move r1 b c`. This threat would not have been detected without this modification.

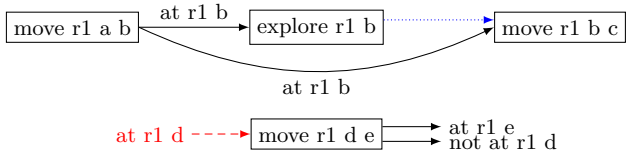


Fig. 3 Example of a flaw that is not detected without geometrical reasoning. The action `move r1 d e` has no common precondition or effects with the other actions but the unicity of the robot position make it clear that this action could not happen in parallel with the rest of the plan.

The position family can also be used when solving open links. If the open link is towards a position, the algorithm can only add a causal link towards the last known position of the robot, and not a causal link with a position that has already been changed. This modification prevents the planner to add a causal link to the plan that will automatically be threatened with no potential resolvers.

4.3.2 Move actions

Another feature of the majority of problems involving position is the availability of some kind of "move" actions: moving a robot is always an option and does not depend on anything else. In such problems, the planner knows for sure that it does not have to guarantee as soon as possible the precondition of position: it could always add a move action later. This allows it to focus on other flaws first and to add the moving actions between already scheduled actions.

Formally, we want for each position family to have access to a set of *move actions*. A *move action* must be available to go from any position to any other position and this action must only depend on the initial position and will modify this position. This action must also be the shortest available action to go from one point to the other. We can add these requirements on move actions to the definition 12 of position literals so that a position family is associated with a family of move action.

Definition 14 (position literals with move actions) Let $P = (L, A, Init, Goal)$ be a problem.

The set of literals $\mathcal{G} \subset L$ is a position family iif.:
1. $Init \cap \mathcal{G}$ has a single element.
2. $\forall a \in A, \forall f \in \mathcal{G},\ f \in Del(a) \implies f \in Pre(a) \wedge \exists g \in \mathcal{G} \mid g \neq f \wedge g \in Add(a)$.
3. $\forall a \in A,\ \forall f \in \mathcal{G},\ f \in Add(a) \implies \exists g \in \mathcal{G} \mid g \neq f \wedge g \in Pre(a) \wedge g \in Del(a)$.
4. $\forall from, to \in \mathcal{G}^2 \mid from \neq to, \exists a \in A \mid Pre(a) = \{from\} \wedge Add(a) = \{to\} \wedge Del(a) = \{from\}$. Let $a^{from \to to}$ be this action.
5. $\forall a \in A, \exists from, to \in \mathcal{G}^2 \mid from \in Pre(a) \wedge to \in Add(a) \implies dur(a) \geq dur(a^{from \to to})$.
6. The duration of the actions $a^{from \to to}$ must respect the triangle inequality: $\forall f, g, h \in \mathcal{G}^3,\ dur(a^{f \to g}) + dur(a^{g \to h}) \leq dur(a^{f \to h})$.

Property 4 states that a move action must exist between each couple of positions: an action that only changes the position without needing anything else. Property 5 states that the move action is the shortest action that moves from a position to another. Property 6 states that the move actions must respect the triangle inequality to ensure that a succession of actions cannot be faster than a single move action.

This definition could be quite restrictive since it assumes that an action exists to go from any point to any other point. In our application, this property holds. It could also be implemented by computing all the missing actions and adding them as abstract actions. The methods of those new actions can be computed as the shortest path using the available elementary actions.

Those *move actions* can be used during the search in different ways, described hereafter.

Taking the plan into account in h^{add} The first use of the move actions is to improve the h^{add} heuristic. In this heuristic, the cost of a given literal is computed in the initial state. So the estimated cost of returning a robot to its starting location will always be 0, even if the robot has moved since.

Using the knowledge of move actions, the estimation of the cost of reaching a position is the cost of the move action that goes from the last known position of the robot to the wanted position. By modifying

the way h^{add} handles position literals, the heuristic becomes then more precise. We call this heuristic h^{add}_{motion}. Let $OL(\Pi)$ be the set of open links of a given plan and $\Omega(\Pi, t, \mathcal{G})$ be the last know literal of $\mathcal{G}$ in Π at time t. This is the last known position in the plan at time t. We can define h^{add}_{motion} as:

$$h^{add}_{motion}(\Pi) =$$

$$\sum_{(t,f)\in OL(\Pi)} \begin{cases} \min\limits_{g\in\Omega(\Pi,t,\mathcal{G})} dur(a^{g\to f}) & \text{if } \exists \mathcal{G} \mid f \in \mathcal{G} \\ h^{add}(f) & \text{else} \end{cases} \quad (2)$$

Instead of using h^{add} for non-position literals in this h^{add}_{motion} heuristic, we can also use h^{add}_{reuse} or h^{add}_{areuse} defined in section 4.2.3.

Solving open links in a new order The availability of *move actions* means that it is always possible to go from one location to another of the same family. Then it is not necessary to add as soon as possible a causal link when an open link towards a position appears. Instead, we decided to solve open links towards position literals after all other open links of the plan. The main advantage of this modification is that it avoids premature commitment: while there is no causal link between two *move actions*, it is still possible to add actions between them. The planner finds better plans than previously because it can insert steps between two steps requiring a given position. If there were a causal link, no other move could be inserted during this causal link since it would threaten it.

Adding temporal constraints Another feature of the *move actions* is the fact that they represent the shortest path to go from a position to another. So before even adding these actions to the plan, it is known that there must exist a minimum duration between different positions.

We propose to add a new temporal constraint in the plan each time an open link to a position literal is created, adding a minimum duration between the creation and the open link in the STN. When the open link will be solved, this constraint will become useless and will not over-constrain the plan since it uses the minimum duration of any chain of actions to go from one point to the other. But until then, the duration of the plan is more precise, and this improves the efficiency of the search.

Using the new temporal constraints in the plan heuristic The first modification (*Taking the plan into account in* h^{add}) uses the current plan to compute the last known position and estimate its h^{add} cost. The last modification (*Adding temporal constraints*) uses this last known position to add a new temporal constraint in the plan. With this temporal constraint, the h^{add} estimation of the cost of reaching a position seems redundant.

We then proposed another modification of h^{add} called *no cost motion* (and noted h^{add}_{ncm}) which sets the h^{add} cost of open links of position literal to 0:

$$h^{add}_{ncm}(\Pi) =$$

$$\sum_{(t,f)\in OL(\Pi)} \begin{cases} 0 & \text{if } \exists \mathcal{G} \mid f \in \mathcal{G} \\ h^{add}(f) & \text{else} \end{cases} \quad (3)$$

4.4 Evaluation

We want to evaluate our planning algorithm and the proposed improvements regarding the gains of using abstract and geometrical reasoning, and compare to other planners.

4.4.1 Problems

To evaluate our planner, we show here some evaluation on problems taken from 2 different domains. One is from the International Planning Competitions (IPC) and one was created to represent multi-robot missions as considered in the introduction. In all the problems, the language description used is PDDL2.1 [23] to represent temporal problems. In addition, since abstract actions cannot be modelled in PDDL2.1, we extended this language to represent abstract actions.

Satellite The *satellite* domain simulates the planning of activities for a satellite. The satellite has a set of instruments and a set of observations to perform. To perform an observation, the instrument must first be calibrated and the satellite must point in the correct direction. Abstract actions are used to calibrate or to perform an observation. Those 20 problems were directly taken from the IPC.

Survivors The *survivors* domain involves a team of robots on a crash site. Each location must be visited by a robot and some survivors are scattered and must be brought to an hospital. Moving one survivor requires at least 2 robots. The locations to explore are grouped into zones and the robots are split between teams. The abstract actions represent a patrol for a given team to explore all the locations of a given zone. This choice limits the size of the problem since there are less zones than locations and less teams than robots.

The robots are organized in team of 2 robots and the problem contains either 2 or 3 teams. The locations are organized in grid of zones. There is either 2, 3 or

4 rows of 2 zones each. Each zone is a square grid of locations of either 4, 9 or 16 locations. So in the largest problem, there is $4*2*16 = 128$ locations There is either 2, 3, 4 or 5 survivors placed randomly and 2 different hospitals. So in total, there is $2*3*3*4 = 72$ random problems generated. The solution found can contains up to 320 elementary actions in the largest problem.

4.4.2 Results

For each problem in each domain, we ran HiPOP in many different configurations. Each configuration was given 10 minutes and 4GB of RAM for each problem on an Intel X5670 processor running at 2.93Ghz.

What are the gains of using abstract actions? To evaluate the impact of abstract actions (thus comparing our hybrid planning with POP planning), we performed experiments in the same conditions without providing abstract actions to HiPOP. The results of the 3 best performing configurations are shown with the 2 best performing POP configurations (labelled `Bare`) in Figure 4.

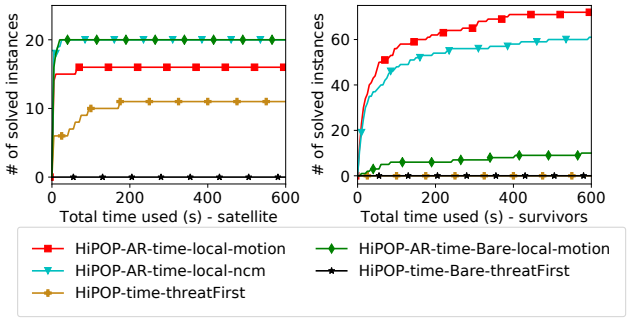


Fig. 4 Results of several configurations of HiPOP on 2 different domains. Each graph shows the number of solved problems (in the y-axis) in less than a given time (in the x-axis). The `AR` stands for *advance reuse* (see 4.2.3), the `R` stands for *reuse*. `local` states that the algorithm selects first the open link for the action most recently introduced in the plan (taken from VHPOP) whereas `threatFirst` does not sort them: they are taken in LIFO order. `motion` states that geometrical reasoning is used (introduced in 4.3) and `ncm` stands for *no cost motion*. `Bare` means that HiPOP is not taking the abstract actions in consideration when planning: it is equivalent to a POP algorithm.

In *survivors*[2], the number of solved problems is significantly higher with abstract actions. The abstract actions allow the planner to reduce the number of plans explored and to reduce the search space, which leads to faster planning.

In *satellite*, the abstract actions are not really interesting since they group together actions that the planner has no difficulty to associate. When the planner adds an abstract action and instantiates it, it only replaces the fact of adding two elementary actions that would have been the only way of achieving the goal anyway. So the abstract actions do not limit the search space and do not really reduce the number of stages needed to find a solution. This explains why `HiPOP-AR-time-local-motion` has the same result, on satellite, with and without abstract actions. It changes the order of resolution of flaws with the `threatFirst` heuristics but has no significant effect with the `local` heuristic.

To conclude, as in HTN, hierarchical planning requires additional work to describe the abstract actions but improves the performance of the hybrid planner in various problems.

What are the gains of using geometrical reasoning? For the brievety of this paper, we do not show the improvement of every modification introduced with geometrical reasoning but instead the improvement of all of them together. Only the "no cost motion" modification gave mixed results, so we present it separately from the others.

Figure 5 shows the performance of HiPOP on 2 domains. Five configurations are shown: the 3 best performing configurations without geometrical reasoning (namely HiPOP-R-time-local, HiPOP-time-threatFirst and HiPOP-AR-time-local) and the 2 best configurationd with it (one with the *no cost motion* heuristic: HiPOP-AR-time-local-ncm and the other without: HiPOP-AR-time-local-motion).

The most notable result can be seen on the *survivors* domain. Without the modification for geometrical reasoning, no problem was solved by HiPOP as the problem relies heavily on geometrical reasoning. But with the geometrical reasoning, all the problems could be solved by at least one configuration. In this domain the *no-cost-motion* heuristic seems to deteriorate the performance. In *satellite*, some problems could only be solved with the geometrical reasoning and the *no cost motion* heuristic.

Our interpretation for the *no cost motion* heuristic is that without this modification, the heuristic overestimates the cost of the plan. When an open link on a position is solved, the heuristic diminishes more than the cost increases: the search will be focused on this part of the search space. So if the first resolution leads to a good solution, this modification helps by focusing the search on a good branch of the search tree. But if the

[2] Similar results were obtained on other domains of the International Planning Competition (IPC) not shown here because of lack of space.

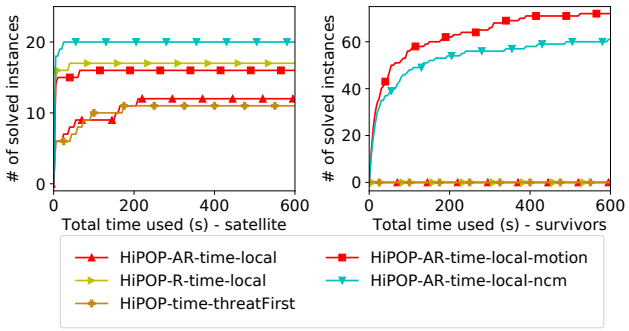


Fig. 5 Results of several configurations of HiPOP on 2 different domains. Each graph shows the number of solved problems (in the y-axis) in less than a given time (in the x-axis). The `AR` stands for *advance reuse* (see 4.2.3), the `R` stands for *reuse*. `local` states that the algorithm selects first the open link for the action most recently introduced in the plan (taken from VHPOP) whereas `threatFirst` does not sort them: they are taken in LIFO order. `motion` states that geometrical reasoning is used (introduced in 4.3) and `ncm` stands for *no cost motion*. `time` states that the objective of the planner is to reduce the time length of the plan.

first resolution does not lead to a good solution, a big space must be searched before considering another resolution for this open link. Improving this modification by finding the right balance between make span/distance of the plan and cost for each situation is an open problem, out of the scope of the article.

Overall, geometrical reasoning is an improvement as it allows to solve previously unsolvable problems.

How does HiPOP behave in relation to other planners? In addition to HiPOP configurations, we also ran several other temporal planners. Since the description of abstract actions is specific to our planner, we could not compare the hierarchical part with another planner. So we used classical temporal planners: Temporal Fast Downward [54], YAHSP [57] and VHPOP [59]. Figure 6 shows the number of problems solved by each planner, with only one configuration of HiPOP given.

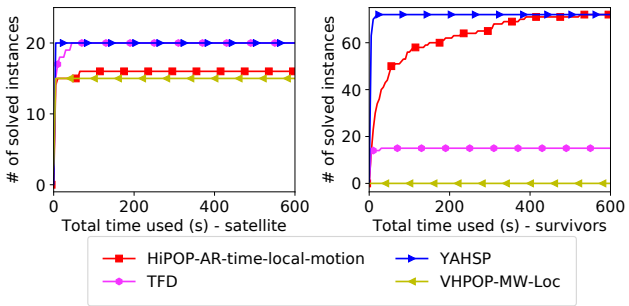


Fig. 6 Results of several planners on 2 different domains. Each graph shows the number of solved problems (in the y-axis) in less than a given time (in the x-axis).

First, one can see that in all the domains YAHSP is orders of magnitude faster than other planners. YAHSP was the winning planner in the agile track in the International Planning Competition. In this track the goal is to find a solution quickly without focusing on the quality of the solution. So it is not surprising to find YASHP to be faster than all other planners. The two other planners TFD and VHPOP are almost always slower than HiPOP. It can be seen that HiPOP can solve the same number of problems in *satellite* (with a configuration shown on previous figures) and more in *survivors* and other domains. The addition of abstract actions allowed HiPOP to be faster than those two planners.

Figure 7 shows the quality of the found solutions. Two configurations for YAHSP are shown: the nominal one and the *anytime*. In *anytime* mode, YAHSP uses all its allocated time to find the best plan possible. We can see here that even in *anytime* mode, the quality of the plans produced by YAHSP is at roughly an order of magnitude worst than with other planners, including HiPOP. This is explained by the fact that YAHSP searches for a plan as quickly as possible without trying to find a good plan. The other planners usually find better plans than HiPOP. This is because by mandating the use of abstract actions, we prevent the planner to search for all the possibilities, even if those possibilities would have been better. The abstract actions can accelerate the search but they worsen the plan quality: there is a trade-off.

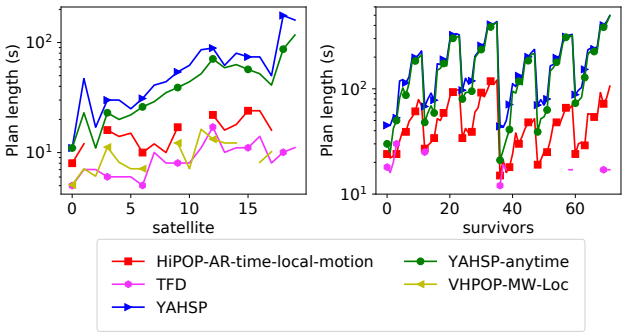


Fig. 7 Results of several planners on 2 different domains. Each graph shows the length of the solution plan (in the y-axis, log scale) for each problem in the domain.

4.5 Application to our case study

The planning algorithm described in this section was used offline to compute an initial plan for the surveillance mission (described in section 1.2). The elementary actions of the robots are: the move action, the

observe action and the communicate action (involving two robots).

The space has been discretized into several sets of waypoints by the operator. A set of reachable waypoints is given for each type of robot, matching its capabilities. The mission is defined by a set of points of interest that must be observed at least once during the mission: this is the goal of the team. An imperfect 3D-model of the terrain is known before the mission. This model has been used to compute an estimated duration for all the move actions and to compute for each point of interest which robot could observe it from which waypoint. The operator also defines a list of patrols as abstract actions: they correspond to allowed successions of move actions, with the corresponding observe actions possible along the resulting path.

The mission also contains mandated meetings between robots. This means that at a given time, a communication action between the two robots in question must be present in the plan. Listing 1 shows the PDDL description of an elementary move action. Listing 2 shows some of the problem data generated from the previous inputs and the 3D model of the environment.

```
(:durative-action move
  :parameters (?r - robot ?from ?to - loc-wp)
  :agent (?r)
  :duration (= ?duration (distance ?from ?to))
  :condition(and(over all (robot-allowed ?r ?from))
                (over all (robot-allowed ?r ?to))
                (over all (adjacent ?from ?to))
                (over all (not (= ?from ?to)))
                (at start (at-r ?r ?from)))
  :effect (and (at end (at-r ?r ?to))
               (at start (not (at-r ?r ?from))))
)
```

Listing 1 Example of a PDDL move action

```
(:init
  //list of allowed waypoints per robot
  (robot-allowed effibot3 effipt_11949_-4580
  //initial position of the robots
  (at-r effibot5 effipt_17600_6350)
  //visibility link between robot position and
  //observation points. Used for observed actions.
  (visible effibot1 effipt_22989_265
                    ptobs_23044_275)
  //visibility link for 2 robots to plan meetings
  (visible-com ressac1 mona r1pt_15139_-3419
                            lpt_17289_-1755)
  //distance between two waypoints.
  //used to compute the duration of a move action.
  (= (distance r1pt_18235_-2588
               r1pt_20232_-2588) 30.98)
  ...
)
```

Listing 2 Example of some generated PDDL statements for the problem

The initial constraints created by the operator are shown on Figure 8. The mission involves 8 robots of 3 different types and about 80 locations. The two "ressac" robots are AAV, the 3 "effibot" are small UGV that can only move on roads and the 3 others are big UGV that can move on grass. The temporal representation of the initial plan computed for the mission is shown in Figure 9. The initial plan has 65 actions and took about 10 seconds for the planner to compute (depending on the computer used).

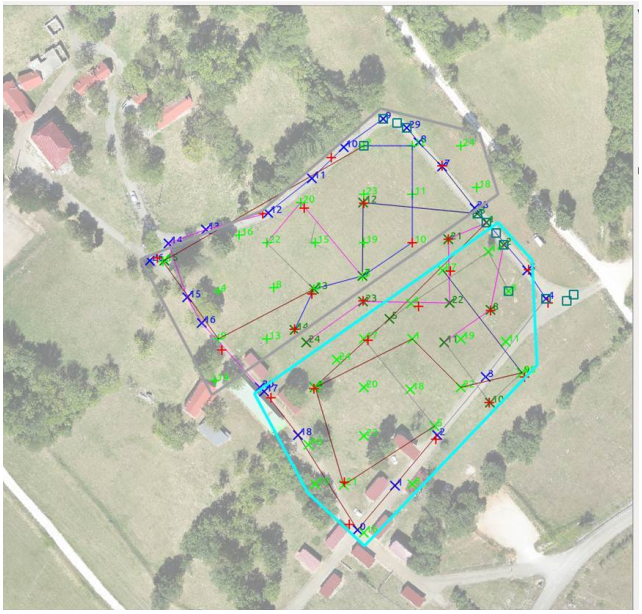


Fig. 8 Screenshot of the mission preparation software. Each red cross is a point to observe. The allowed positions for the AAV are in light green (encompassed in two safety zones in cyan and gray). For the two types of AGV, there are in blue and dark green. In addition, the patrols are drawn between allowed positions. The green squares represent the starting points of the robots.

4.6 Execution

In order to execute the plan onboard the robots during the mission, we have developed a distributed execution mechanism. As we must be robust to intermittent communication, we have adapted the classical STN execution process to a distributed architecture.

The idea is that each robot is executing its own part of the plan and broadcasts to the others when an action is executed. If no words are heard from other robots, they are expected to follow the original plan. Else, if a message is received, the plan is updated in consequence. If an inconsistency occurs, a repair is triggered. This strategy allow robots to limit the number of repairs to only the cases needing one, preventing a repair for every delay during the execution.

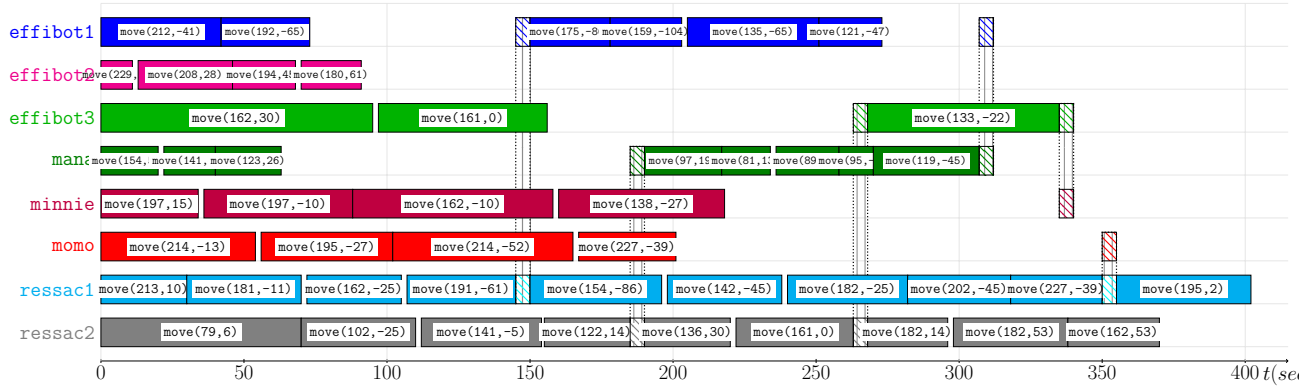


Fig. 9 Temporal representation of the initial plan (x-axis in seconds). Each horizontal line represents the plan of a given robot with each block representing a move action. Each vertical bold line represents a communication action between two robots (then 6 meetings).

5 Repair

During the execution of the mission, the team must be able to react to disruptive events. As stated in the section 2.3, we have chosen a "unrefine-then-refine" approach. This allows us to re-use our planning algorithm as a repair algorithm. This section first presents improvements made to HiPOP to repair a plan and its evaluation on an appropriate benchmark. We then present the improvements made to repair a plan during its execution and the plan execution algorithm developed to supplement the planning algorithm in the embedded architecture.

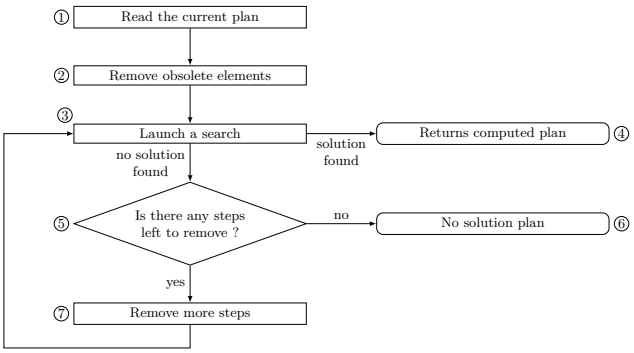


Fig. 10 Repair algorithm: the idea is to iteratively try a search starting from a partial plan computed from the initial plan until a solution is found or until the replanning fails.

5.1 Repairing a plan

In this work we assume that repairing is similar to a planning problem except that the repairing algorithm uses a non-empty initial plan. This plan might not be valid anymore but we expect (as a result of how the plan was built) that a solution plan close enough to the initial plan exists.

The main idea of the repair process is to successively call the HiPOP algorithm with a specific initial plan $\mathcal{I}$. When performing the initial planning, this starting plan is empty (see Algorithm 1): it only contains the *Init* and *Goal* states. In the repair setting, this plan is computed from the current plan by removing some elements (actions, causal links and temporal links). Given this initial plan, HiPOP will try to find a solution only by adding elements to it. If none is found, more elements are removed from the initial plan and a new search is launched using the new starting plan. The whole process is described in Figure 10.

The repair algorithm must be able to deal with several issues: obsolete actions (actions that are in the initial plan but not allowed anymore), a change in the initial conditions and a change in the goal.

The first stage is to remove from the plan all obsolete elements (stage 2 in Figure 10) and actions that are not in the problem description anymore. If actions are allowed to change their preconditions or effects, some causal links could also become invalid and are removed. This is the case for instance if the goal has changed: the last action of the plan has changed its precondition which could remove some causal links. Once those causal links are removed, the algorithm also removes all the steps that do not support causally another action as they do not accomplish something that helps accomplishing the goal (i.e. we remove all steps $\tau \in \mathcal{T}$ such that $\nexists \tau_i \; in \mathcal{T}, (\tau \longrightarrow \tau_i) \in CL$.)

HiPOP can then be run with the newly computed starting plan (stage 3 in Figure 10). If the search starting with this plan finds a solution, then the repair has succeeded (stage 4 in Figure 10). Else, another starting plan must be created by removing more elements.

We decided to use the causal links in the plan to determine what actions to remove next (stage 7 in Figure 10). The idea is that if a problem occurred in the plan, the first steps to remove are the one directly impacted by this event. The next one should be the steps "close" in the plan: steps that were added to support the removed steps or steps that were supported by the removed steps. So in this stage we remove all the steps that were causally linked to a step removed at the previous iteration: all steps $\tau \in \mathcal{T}$ such that $\exists \tau_i$ *in* $\mathcal{T}$, $\exists(\tau_i \longrightarrow \tau) \in CL$ and τ_i has been removed in a previous stage. For instance if a communication is removed from the plan, the actions that moves to and from this position are the next candidate to be removed from the plan as we want the modification of the plan to be as local as possible.

To ensure the completeness of the algorithm, if there is no more step to remove by following causal links, the last stage is to remove all steps and try one last search. In this case this is a "replanning" and no longer a "repairing": it is a planning problem that do not take into account the initial plan. If this last replanning fails, then there is no solution to the problem (stage 6 in Figure 10).

5.1.1 Using the hierarchy in the repair process

The previous modification allows a POP planner to repair a plan. In our hybrid planner, there are another information in the plan: the abstract actions. We believe that this information could be used to improve the repair process. Semantically, two steps that are parts of the same abstract action were added to achieve the same high-level goal. If this goal is no longer needed or if one of the actions is not available anymore, it could be better to remove the whole abstract step. This is the same idea than following the causal links: we want to find the steps that were enabled or required by the steps removed from the plan.

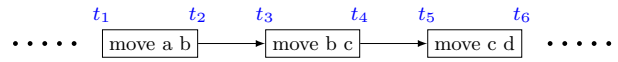
Whenever a step that is part of an abstract decomposition is removed, we choose to un-instantiate the abstract step. This means that the algorithm will remove all the elements (steps, causal links and temporal links) that were added by the method chosen to instantiate this abstract action. When an abstract step is removed, the algorithm will un-instantiate it before.

5.1.2 Phantom elements

By removing causal links and/or temporal links, the ordering of the steps can be changed. This change can create a lot of threats (thus increasing the amount of work needed to find a solution) and decreases the plan stability (because the produced plan can be very different from the initial plan).

To prevent these drawbacks, we decided to keep the ordering of the steps implied by all the elements of the initial plan, even if some of the elements are removed. This preservation is done by introducing *phantom links*: a removed step, causal link or temporal link, is replaced by a phantom link in the generated plan. This phantom link imposes a temporal precedence between the two timepoints. When removing a step, the algorithm will keep the timepoints of the start and of the end of the step in the STN as *phantom timepoints* with all their temporal constraints. In the initial plan the minimum separation between those points was the duration of the action. In the generated plan the constraint is only that the start is before the end.

Once all the constraints between the timepoints are computed, the phantom links and the phantom timepoints can be removed from the plan. This process is illustrated in Figure 11. The initial plan is shown in Figure 11(a). When the second step is removed from the plan, all causal links that are related to the removed step are also removed from the plan. All the removed timepoints and links are kept in the plan as phantom links (Figure 11(b)). Before using the plan as a starting point for the search, the phantom elements are removed from the plan and temporal constraints are added to keep the order implied by the phantom links. In this example, a temporal constraint is still present in the plan to force the first step before the third step (in blue in Figure 11(c)).



(a) Initial plan with 3 steps and 2 causal links (in black).

(b) The second step is removed from the plan. Phantoms links are in dotted green.

(c) New temporal constraints are in blue.

Fig. 11 Use of phantom elements on an example.

5.2 Evaluation

We created a benchmark to evaluate the ability of HiPOP to repair. A repair problem is defined by the initial

problem, the initial plan (a plan solution of the initial problem) and the problem to solve (with the assumption that this problem is close enough to the initial problem). We modified the generator of *survivors* problems to first generate an initial problem, secondly solve it with HiPOP and thirdly modify it to create a repair problem (*survivors-repair*). We introduced 3 types of disruptive events: a change in the initial position of the robots (no modification of the PDDL domain), the detection of a new survivor (a new goal to achieve) and the modification of the position of a survivor (modification of the goals and of the domain by removing abstract actions related to the old position and adding abstract actions related to the new position). These 3 events are respectively called *startingPos*, *newSurvivor* and *survivorPos*.

We built a benchmark in the same condition than in 4.4.2 by generating a set of repair problems and running many configurations of HiPOP. The configuration labelled *repair* have been given the initial plan while the other are replanning: they are only given the new problem to solve. To analyse more closely the result, the data are split according to the introduced disruptive event.

What are the gains of repairing instead of replanning? The results of the two best repair configurations and of the two best replanning configurations are shown in Figure 12.

One can see on the first row that for all the events, repairing is faster than replanning. This is expected when the initial plan is close enough to a solution plan.

The third row shows that, as expected, the plan produced when repairing is close to the initial plan: no matter the size of the initial plan is, the modification only deals with a fixed number of actions. This result means that the plan is only locally modified. On the other hand, when replanning, the number of changed actions is proportional to the number of actions in the initial plan.

If the plans produced when repairing are more stable, one could expect their length to be higher. This loss of quality would be due to the fact that the exploration is not performed on the whole space. The second row shows that this effect is not significant. Since HiPOP is not an optimal planner, there are cases where repairing produces better plans and cases where repairing is worse. But the difference is not significant.

What are the gains of repairing with abstract actions? We also wanted to highlight the role of abstract actions in the process. As seen previously, HiPOP is not able to solve this type of problems without abstract actions. So we created a new configuration that could use abstract actions but that discards every information about hierarchy in the initial plan. This configuration is called *repairFlat* as it tries to repair a flatten version of the initial plan.

The results of the best *repair*, *repairFlat* and replanning configurations are shown in Figure 13. One can see that the hierarchy has different effects according to the type of disruptive event.

When the starting position of robots is changed (event *startingPos*), *repairFlat* is similar to *repair*. That's because the first action in the plan is usually a move action to reach the initial position of an abstract action. As changing the initial position only impacts the move action, the two configurations are similar.

When a new goal is added (event *newSurvivor*), *repairFlat* is similar in speed to replanning but keeps the plan more stable. Since abstract actions are removed, some causal links are removed which creates more work for the planner. In some cases, the planner times out before re-establishing all those causal links and another iteration is needed (which explains the spike in the last row: a lot of actions are removed during the second iteration).

The hierarchy is best used in the last setting, when a goal is changed (event *survivorPos*). In this case repairing without abstract actions is worse than replanning. This is due to the fact that several actions must be removed. As they are all part of the same abstract action, the *repair* configuration removes them all at the same time. The *repairFlat* does not have this piece of information. So it must remove iteratively several actions following causal links, which leads to removing too many actions: causal links also linked to actions that were in the following or the previous abstract action. The plan instability is then higher (since more actions have been removed) and the search slower.

In conclusion, repairing instead of replanning shows an improvement of the performance of HiPOP. Repairing also allows the process to be more predictable for the supervisor: a large part of the initial plan is kept when a repair modifies the plan. In addition, we have also shown that the presence of abstract actions in the initial plan improves the repair process: it is faster and the obtained plan is more stable.

5.3 Repairing a partially-executed plan

This part describes the modifications made in HiPOP to allow it to repair a partially-executed plan. Once its execution has started, new constraints arise: the duration of actions can change and the past actions can

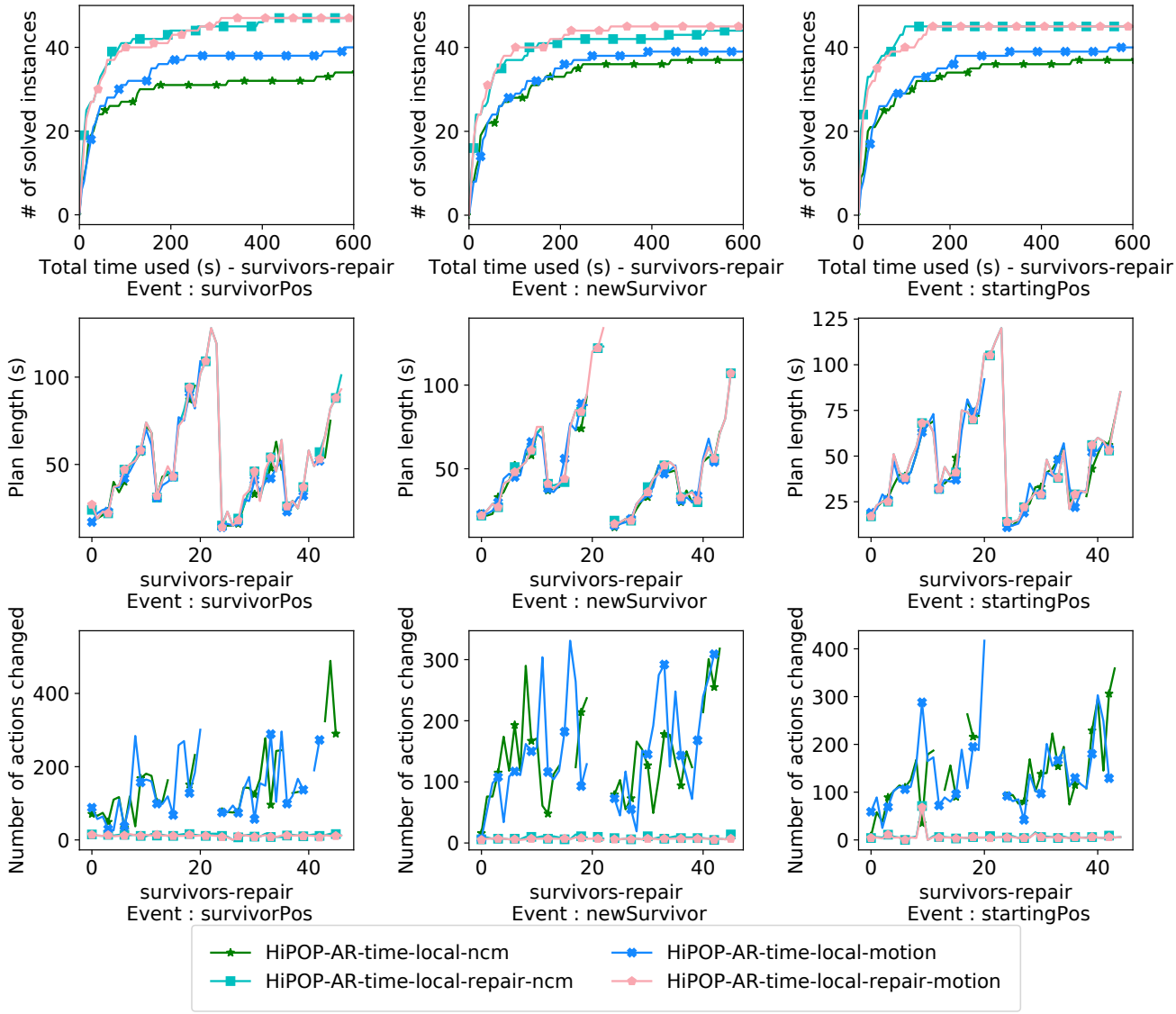


Fig. 12 Results of several configurations of HiPOP on a repair benchmark. Each column represents a disruptive event to repair (i.e. a change in the problem to solve). The first row gives the number of problems solved in a given time (in the x-axis). The second row gives the length of the found plan for each problem. The third row gives the plan *instability*, i.e. the number of actions that have changed between the initial plan and the solution plan for each problem. Since the problems are generated by changing some factors (the number of zones, number of robots, etc.) the problem are not sorted by order of difficulty. This explains the 'shape' of the plot in the last 2 rows, and some discontinuities appears if a problem is not solved but the next one is.

not be removed from the plan. So HiPOP has to be adapted to these constraints if we want to use it online. The starting plans for the search will then be different from the ones used in the repair algorithm presented in Figure 10.

The first constraint is the fact that executed actions are in the on-going plan. The planner must also be able to deal with actions that had a different duration than their nominal definition. This could impact the feasibility of meeting a deadline. To repair a plan with an unreachable deadline, we decided to remove all steps between the current time and the deadline. This choice guarantees that nothing unnecessary is imposed to let the planner meet this deadline. If there is enough time, the repair process will add back some of those steps before the deadline, or they will be scheduled later. In any case, it is the planner job to choose how to achieve the deadline, the only goal of this repair process is to avoid over-constraining the planner. A summary of all the stages of the repair algorithm is shown in Figure 14. The consideration of deadlines (stages 3 and 4) is the added value of this repair based on a partially-executed plan.

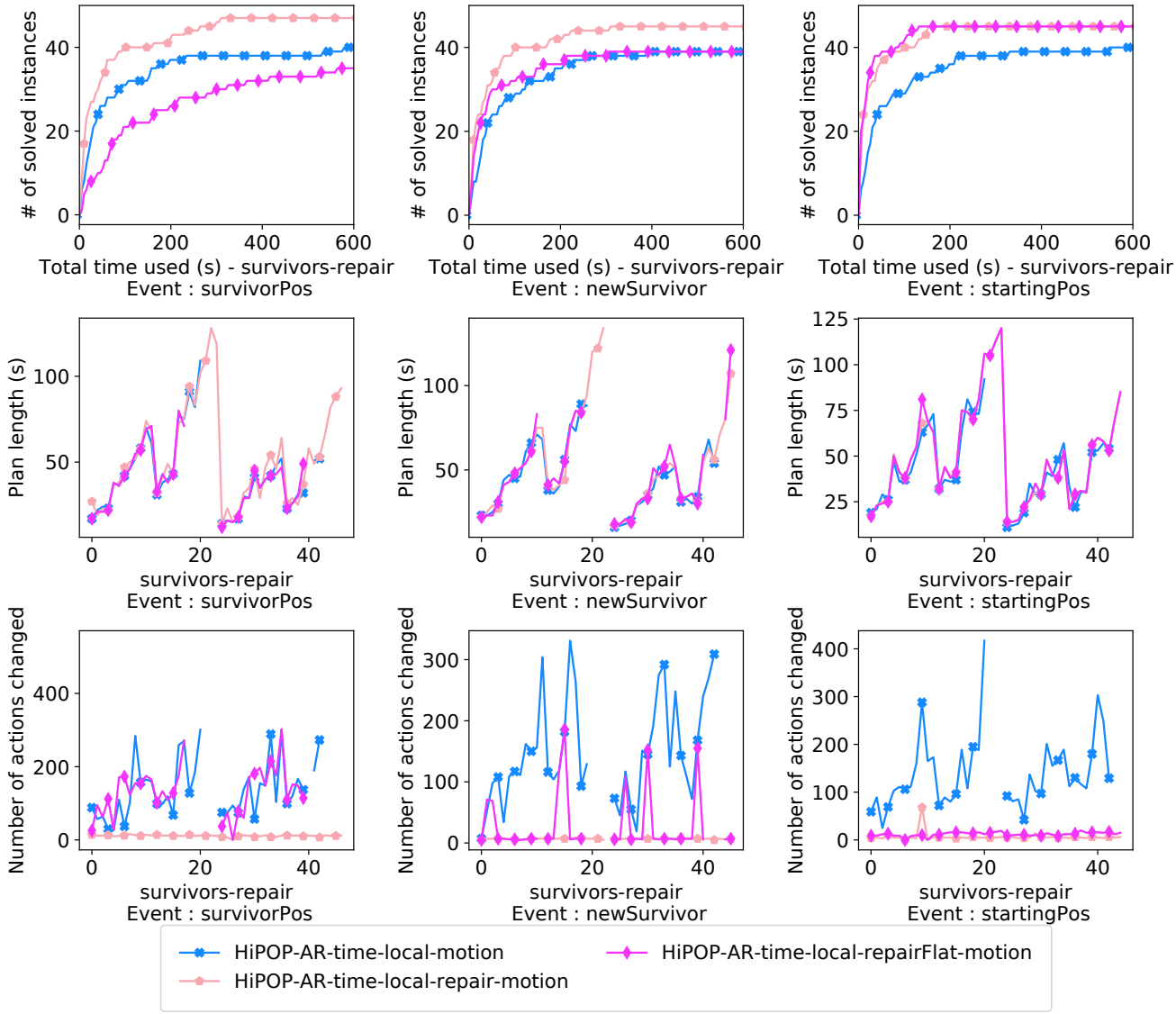


Fig. 13 Results of several configurations of HiPOP on a repair benchmark. Each column represents a disruptive event to repair (i.e. a change in the problem to solve). The first row gives the number of problems solved in a given time (in the x-axis). The second row gives the length of the plan that are found for each problem. The third row gives the plan *instability*, i.e. the number of actions that have changed between the initial plan and the solution plan for each problem. Since the problems are generated by changing some factors (the number of zones, number of robots, etc.) the problem are not sorted by order of difficulty. This explains the 'shape' of the plot in the last 2 rows, and some discontinuities appears if a problem is not solved but the next one is.

Another difficulty comes from the fact that the actions already executed or in execution cannot be modified. In the process described earlier, only non-executed steps can be removed. A step tagged to be removed by the stage 2 but already executed has then to stay in the starting plan.

This fact has some consequences when abstract actions are in the plan. We decided to un-instantiate all steps associated to an abstract action or one of its children when we want to remove an abstract action. But when one of its children has already been executed, there can be in the plan an half-executed abstract action, i.e. an abstract action with only a part of one method. This contradicts the idea of methods and of allowed actions (described in section 4.2): we wanted to limit the set of actions that could be used directly to solve an open link because we assumed that the methods would always be used in full. In the cases where an half-executed abstract action must be un-instantiated, the planner removes then all non-executed steps and must allow all the actions to be added in the plan to be able to repair this method, not just the allowed actions (i.e. $\mathcal{A} = A$).

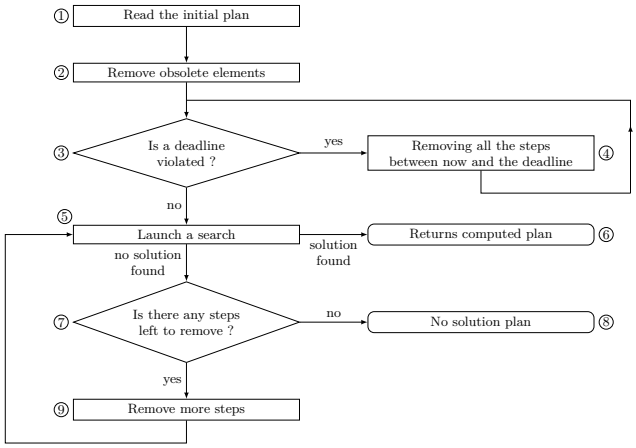


Fig. 14 Repair algorithm: extension of Figure 10 taking into account the deadlines in the plan. Step 3 and 4 can be repeated if multiple deadlines are violated.

Another problem that can arise is the fact that we have no guarantee on the time taken to repair the plan. This means that when a repair was done, the plan was usually still temporally valid. In the rare case when the plan was invalid, it was because a deadline was nearly infeasible during the first repair and in this case the second repair was successful. In more time critical mission, the operator might be able to intervene to remove some goal or the deadline to ease the search. But this is out of scope for this paper.

5.4 Repairing with partial communication

Since plans can be executed in a distributed fashion, we need to adapt HiPOP to enable it to repair a distributed plan. The repair process is shown in Figure 15 for an example with 3 robots, one being out of reach when the repair is triggered, the communication coming back after a moment. When a repair begins during the execution phase, since the planning algorithm only deals with global plans, all the robots must send their local plan to the robot tasked with the repair (merger phase). Since the plans use the POP formalism and come from the same global plan, they can be directly merged. So the tasked robot have all the information to repair the plan for the whole team (repair phase) and will send the repaired plan to everyone else (dispatch phase). This process allows one robot to repair for everyone else without relying on a single "master" or "server". Any robot, at any time, can trigger a repair when it detects a problem. If some problem arise preventing this robot from completing the repair process, any other robot can detect it (by realising that the new plan is not sent) and start the repair process itself.

Some adaptations must be done to HiPOP to deal with imperfect communications. We assume that either a robot can communicate with the repairing robot during the whole repair process or it cannot. With this assumption, when a repair is triggered, the global plan is computed using the local plans of reachable robots and the last known local plans of unreachable robots. During the merger phase of Figure 15, Robot 1 keeps the last known local plan of Robot 3 (the last line of its global plan, in red) since it cannot obtain from Robot 3 its updated local plan. The repairing algorithm must not modify the local plan of unreachable robots: as with executed steps, the planning algorithm is thus not given the possibility to modify steps of unreachable robots. This constraint guarantees that after the repair process, the robots will have compatible local plans. During the repair phase of Figure 15, the last line of the global plan cannot be modified, only the other lines (represented with a cross pattern) can be modified.

Even with local plans generated this way, each robot must have the same global plan when exchanging information about their plan. To achieve this, a global identifier (ID) is assigned to each global plan and sent with it. This ID allows each robot to detect whenever the global plan of a reachable robot has changed. So the robots executing the latest plan can detect it and do nothing while the robots executing the oldest plan can take the new plan as the global plan (synchronization phase). Since this new plan has been computed without modifying their local plan, the new global plan is also valid for them. In the case where two repairs have been carried out without any communication between them, a new repair is carried out to make sure that the two plans together are still valid and that all the problems have been dealt with by all the robots. This assumes that at the end of the mission, the robots must be in communication range with the others (or with the operator) to notify them that they have finished their part of the mission. For instance, this can be introduced in the plan by having a given position to reach at the end for every robot that is in communication range with the others.

We also ensured that two robots in communication range will not try to repair the plan at the same time. When a robot repairs it starts by contacting all the other robots to compute a global plan. So a robot can detect if another robot wants to repair at the same time and only one robot is allowed to repair. We choose to enforce it by having a priority among robots based on their name: only the first one in alphabetical order continues with the planning. One could use any other criteria for this, such as available computing power or bandwidth for instance. In addition, if a repair is currently

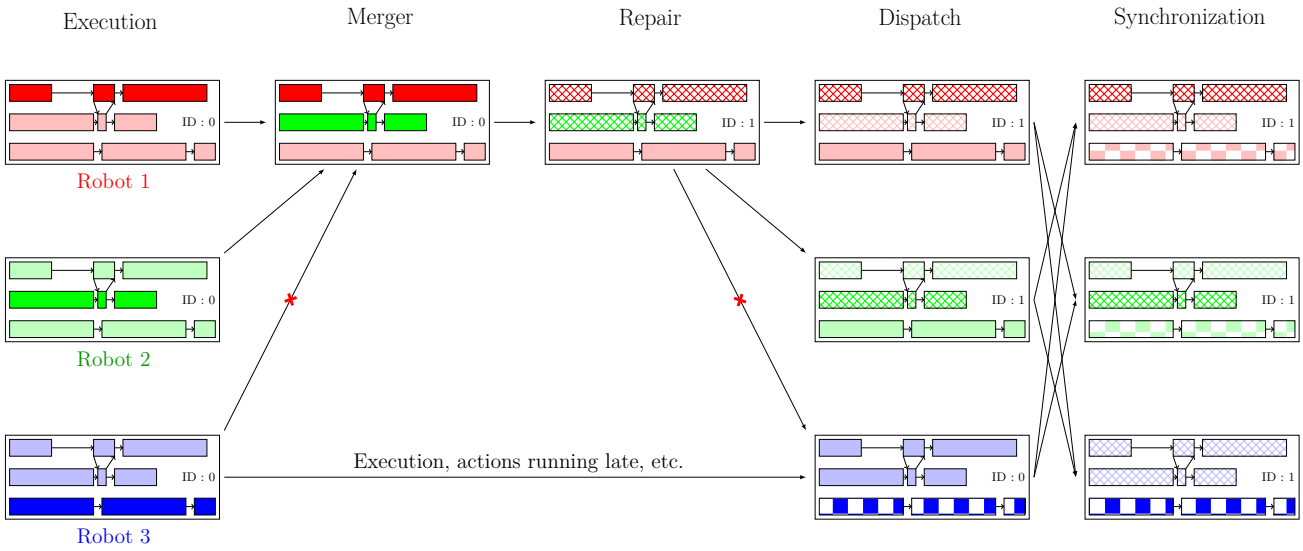


Fig. 15 Repair process for a team of robots where one robot is isolated for a moment. The global plan for each robot is represented in a box and each line in this box shows the local plan of each robot. During the execution, Robot 1 encountered an event that triggered a repair with only Robot 2, Robot 3 being out of reach at this time. The communication is re-established after, triggering the synchronization phase.

going on (i.e. if a robot has been contacted by another robot and has not yet received a repaired plan), this robot will not start a repair until the new plan is broadcasted. When the new plan arrives, the plan might not be valid and need an additional repair.

5.5 Repairing applied to the surveillance mission

In the surveillance mission we consider (see section 1.2), the repair can be triggered by:

- an inconsistency in a robot STN, meaning that a future deadline will be violated; such deadline may be linked to the current communication task, meaning that the other robot is too late at the meeting point;
- a robot being out of service, which is indicated by the operator;
- a robot being allocated to the tracking stage, then being unavailable for the surveillance.

Moreover, if the repair algorithm fails to find a solution, we try to remove some goals and re-try the repair in order to not abort the mission. We can then drop meetings. If after removing all the meeting no solution is found, then the operator is notified that the mission cannot be carried out, and can decide to withdraw some observation points, or to allocate robots currently in the tracking stage to the surveillance stage in order to find a new feasible plan. If a meeting is dropped by a robot, it will try to notify the other robot. If the other robot is out of reach, it may never learn about it and wait for the first robot at the meeting point. But in any case, it will trigger a deadline and the other robot will also drop the meeting to carry on with the mission. So having one robot drop a meeting without being able to notify the other robot will not prevent the mission from being carried out.

6 Evaluation of the whole architecture

The whole architecture, including the HiPOP algorithm, the execution and repair processes, has been evaluated, first in simulation in order to assess the performance and the robustness of our contribution. We have then perfomed experiments on the field with real robots.

6.1 Statistical evaluation

The statistical evaluation has been performed in an ad-hoc simulation. Our objective is to evaluate the performances of the architecture composed of the distributed execution and the distributed repair. We have executed the plans by assuming that each action takes its expected duration to run and we added disruptive event to this lightweight simulation. This has allowed us to run multiple simulation in a short timeframe for the statistical analysis.

We have created a set of patterns of the disruptive events that could happen during the execution: a broken robot, a robot out of communication, a target found, a delayed action and many combinations of these events. The chosen value for the delay of an action is of

45 seconds. This value is enough to trigger the deadline associated with a communication (when a robot waits for another robot for a scheduled communication, it will not wait longer than 30 seconds to prevent a deadlock when the other robot is stuck out of communication range). This pattern could have no impact (if the plan is flexible enough to deal with it) or it could increase the duration of at most 45 seconds. When a robot is out of communication (labeled "out of reach"), it cannot communicate with any other robot for one minute. Since the execution does not need the communication, we guarantee that this happens while another disruptive event is taking place in order to evaluate the robustness of our architecture to communication failures.

For each pattern, 30 scenarios were randomly generated and ran. The following results show the mean of several metrics for each pattern. The reference is the nominal achievement of the mission (no disruptive event), shown with a vertical bar in the following figures. The first significant result is that for all the executed missions, HiPOP never failed a repair and the missions were thus carried out to the end. For each mission where a target is detected, a robot was given the order to follow it, and the repair without this robot has been successful.

Figure 16 shows the duration of the mission. The longest missions are in the last pattern when 5 random events happen. When 2 targets are found, two robots have to be replaced, which also increases significantly the mission duration. But this increase is always relatively limited (less than 20% on average).

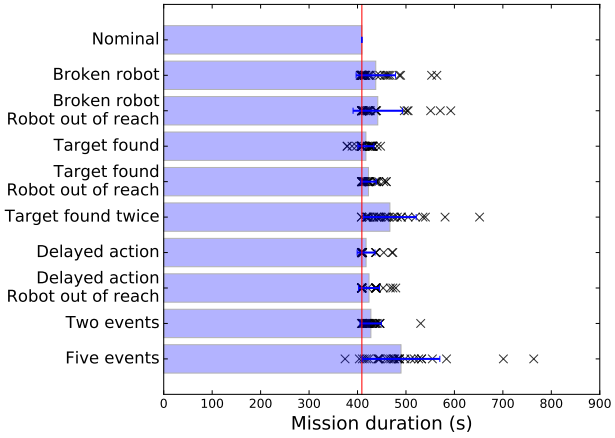


Fig. 16 Duration of the mission when disruptive events happen during the mission. The red line shows the mean for the nominal mission, each cross represents a mission and blue bars represent the mean for all the missions of the same pattern.

Figure 17 shows the number of points of interest that are successfully observed during the mission. When a robot breaks down or is assigned to a target, its actions must be reallocated to other robots. In our data, there exist 3 points of interest that can only be observed from the waypoints of only one robot. So if this robot fails, 1, 2 or 3 points could no longer be observed. This explains why the minimum number of observed points is 22 out of 25. Even if all points of interest have not been observed, these missions are considered successful for the architecture point of view: this is the best possible result regarding the failures that happened.

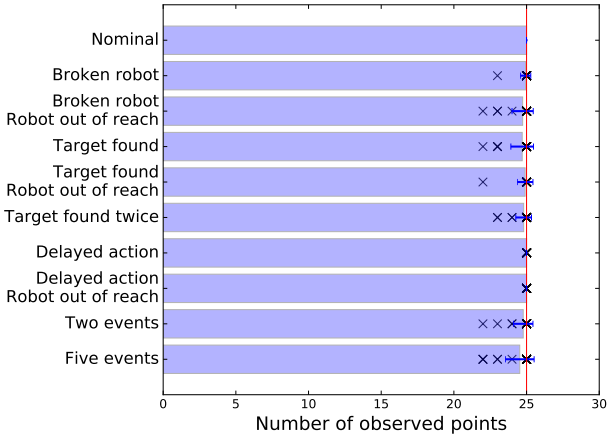


Fig. 17 Number of observed points when disruptive events happen during the mission. The red line shows the mean for the nominal mission, each cross represents a mission and blue bars represent the mean for all the missions of the same pattern.

Figure 18 shows the number of successful meetings between the robots. 6 meetings were planned in the initial mission. When a robot breaks down, follows a target or is too late, a meeting can be canceled to carry on with the plan. But we did not implement the possibility to schedule new meetings, this is why the number of meetings quickly drops with the number of disruptive events.

Figure 19 shows the number of messages transmitted between the robots. When no disruptive events happen, only synchronization messages are sent. When disruptive events happen, a repair can be triggered. The repair process needs some messages and can increase the number of actions, thus increasing further the number of messages. The number of messages is correlated with the mission duration and is limited. So the repair process does not increase significantly the number of messages needed to be exchanged.

Further to this statistical evaluation by simulation, our architecture proved its robustness with respect to the occurrence of several types of disruptive events.

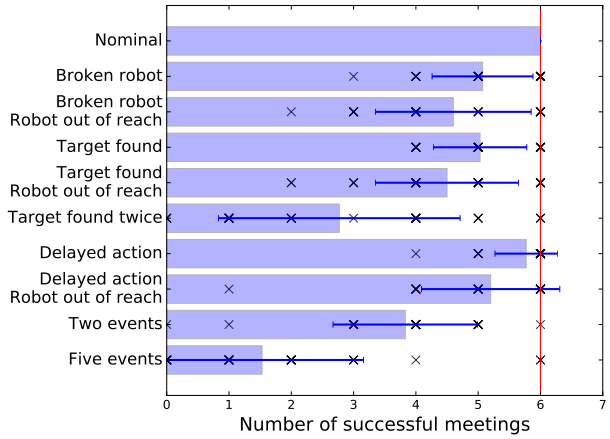


Fig. 18 Number of successful meetings when disruptive events happen during the mission. The red line show the mean for the nominal mission, each cross represents a mission and blue bars represent the mean for all the missions of the same pattern.

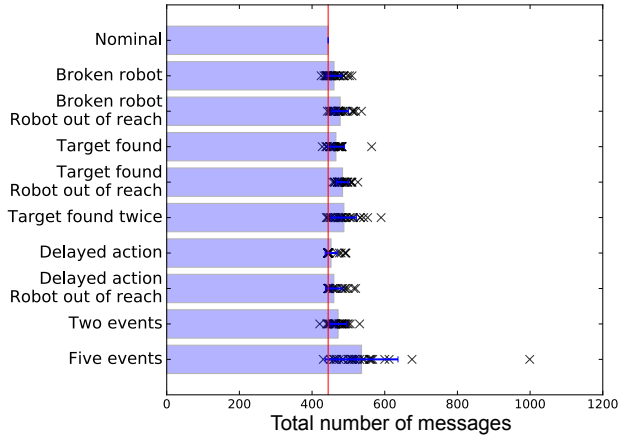


Fig. 19 Number of messages transmitted when disruptive events happen during the mission. The red line show the mean for the nominal mission, each cross represents a mission and blue bars represent the mean for all the missions of the same pattern.

6.2 Experimental demonstrations

The mission described in section 1.2 has led to the initial plan presented in section 4.5. We have then performed the mission with 2 aerial robots from Onera (ressac1 and ressac2), 3 ground robots from LAAS-CNRS (Mana, Minnie and Momo), and 3 ground robots from DGA (Effibots). We have moreover considered 4 simulated robots in the team (ressac3, Mona, effibot4, effibot5). The simulated robots had initially an empty plan (meaning no action scheduled) and could be activated if needed during a repair. The simulation environment used is the MORSE simulator [20]. Each robot was running HiPOP and our execution framework. Communication between robots was done using ROS. Each elementary action for HiPOP was given as a task for the robot middleware to execute. Each robot was running its own middleware, and we then implemented wrappers in our execution framework to manage action execution on the several middlewares (ROS, Orocos, socket).

8 missions have been played on an experimenting field located in the south of France. The robots deployed in the mission are heavy (several tens of kilograms), and require some logistics (in term of infrastructure and people involved). This prevented to perform a lot of experiments: simulations were used to statistically evaluate our architecture (see previous section), whereas experiments were used to assess the operation feasability of the mission.

One of the 8 missions ended without the need to repair a plan. Some actions were late but the temporal flexibility in the plan was enough to deal with them without repairing. The trajectory of the robots for this nominal mission is shown in Figure 20 and the temporal representation of the executed plan is shown in Figure 21. One can see the trajectory of the two aerial robots (ressac1 and ressac2) being distorted mainly because of the wind, which lead to all the move actions being late (but without the need to repair as mentioned previously). The initial plan has a duration of 410 seconds. This execution lasted about 450 seconds.

Plan repair was needed for the 7 other missions. In every case, HiPOP was able to repair and the mission to finish. One such mission is represented in Figure 22. Two simulated robots are used to take the actions of robots unable to finish their plans. The two broken robots had a problem after about 2 min but the operator declared them unavailable only after 500 seconds. This explains why the spare robots did not start earlier and why the mission lasted over 700 seconds. Out of the 6 meeting scheduled, 3 were dropped because one of the robot was unavailable, 2 were dropped because robots were too late and only 1 took place.

In the 6 other missions, two of them have no robot failure. In one of them a target appeared, meaning that a robot still has to discard its plan to follow the target. In the other one, a robot was too late at a meeting point meaning that a communication was dropped. The other 4 had at least one robot failure (i.e. a robot stuck somewhere, a robot for which the safety operator had to intervene, etc.). Overall, 13 robot failures happened during those 5 missions (including the one presented earlier). Each failure triggered a repair and the mission length was increased by 1 to 5 minutes depending on the mission.

These demonstrations were a success. They validated experimentally on realistic patrolling missions the HiPOP planning algorithm and its integration in

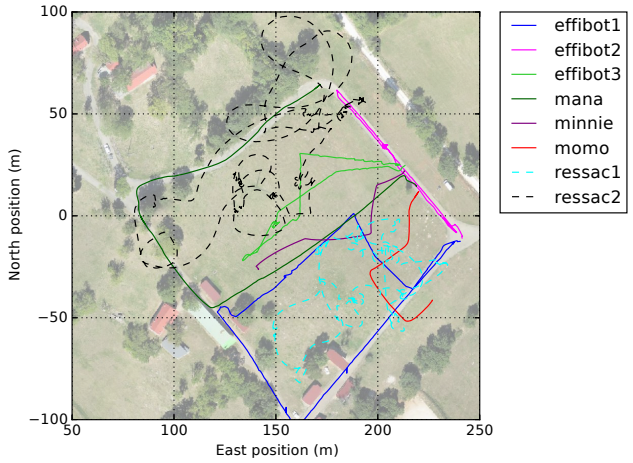


Fig. 20 Field experiment: trajectory of the robots during the nominal mission (no online repair needed).

an embedded architecture with the described execution algorithm. Robots cooperated efficiently in the team to achieve the missions in an autonomous way when disruptive events broke the current plan, and especially in case of communication losses. The human operator was at a minimum level in the loop: he only had to take high level decisions. The repair process only modified the plans of few robots and not of the whole team.

7 Conclusion and future work

The use of a lot of robots having different and complementary capabilities widens the type of missions that can be achieved by a robotic system. The autonomy of the team is essential when the communications are not guaranteed and it also reduces the workload of human operators who can focus on high level exchanges. This paper addresses more specifically the problem of using heterogeneous robots in complex missions operated in outdoor environments. The computation of a flexible initial plan for the team of robots is a central issue but being able to adapt this plan online when a disruptive event occurs is the added value of this work. This paper has described the design of an hybrid planner, HiPOP, that mixes HTN and POP planning. This planner is able to use hierarchical actions defined by the human operators to improve its search of a plan for the team of robots. It outputs a plan with temporal flexibility, which eases its execution, and with a hierarchy among actions, which eases the online repair process. The architecture has been evaluated for a surveillance mission both in simulation and during experimental demonstrations. Both evaluations were successful and validated our initial choice of using hybrid planning. The abstraction of the capabilities of the robots and the abstract actions allowed us to easily add a new type of robot in the team.

Future work could deal with the limited temporal scope of this approach: missions were planned for 5 to 10 minutes. The goal was to observe once every point of interest. For longer-term autonomy, the goal of the mission must then be updated. For instance instead of using a plan that visits each interest point once, the planning algorithm could try to find a plan that minimizes the time between two observations of the same point. Meetings could be automatically added to the plan in the repair process to guarantee synchronizations between all the robots of the team. Instead of just repairing the plan when a disruptive event occurs, the plan could be extended regularly. Another approach could be to compute a valid plan for the next few minutes and an abstract plan (i.e. a plan where abstract actions are not instantiated) for the next few hours. In this way there will be a visibility on the high level goals to achieve in the long term while still having a valid plan to execute. And periodically, the rolling plan would be instantiated for the next few minutes when needed.

Acknowledgements The authors would like to thank the French Procurement Agency (DGA) for its funding.

References

1. Abichandani P, Benson HY, Kam M (2013) Robust communication connectivity for multi-robot path coordination using Mixed Integer Nonlinear Programming: Formulation and feasibility analysis. In: International Conference on Robotics and Automation (ICRA), Karlsruhe, Germany
2. Amigoni F, Banfi J, Basilico N (2017) Multi-robot Exploration of Communication-Restricted Environments: A Survey. IEEE Intelligent Systems 32(6):48–57
3. Arrichiello F, Marino A, Pierri F (2015) Observer-based decentralized fault detection and isolation strategy for networked multirobot systems. IEEE Transactions on Control Systems Technology 23(4):1465–1476, DOI 10.1109/TCST.2014.2377175, URL
4. Bajada J, Fox M, Long D (2014) Temporal Plan Quality Improvement and Repair using Local Search. In: Starting AI Researcher Symposium (STAIRS), Prague, Czech Republic
5. Banfi J, Li AQ, Basilico N, Rekleitis I, Amigoni F (2016) Asynchronous multirobot exploration under recurrent connectivity constraints. International Conference on Robotics and Automation

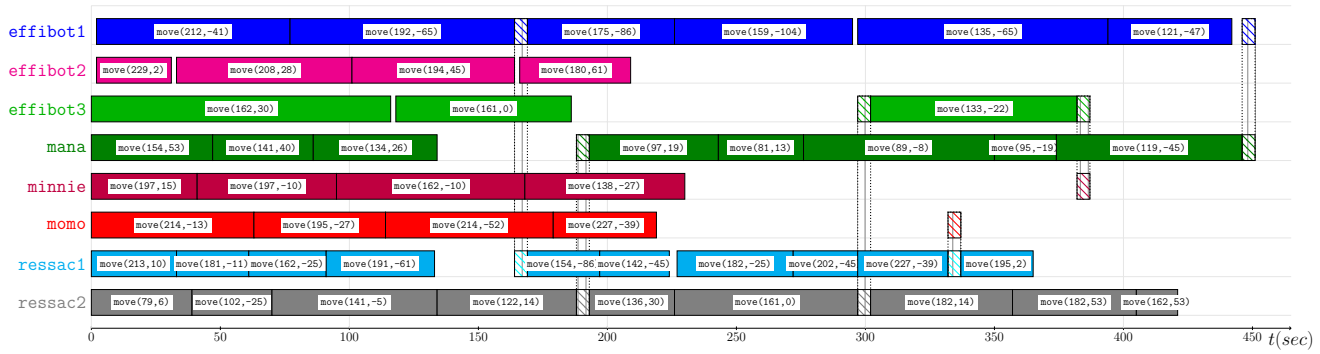


Fig. 21 Field experiment: temporal representation of the plan executed by the robots for the nominal mission. The vertical bars represent communication actions between two robots: the 6 scheduled meetings have been done.

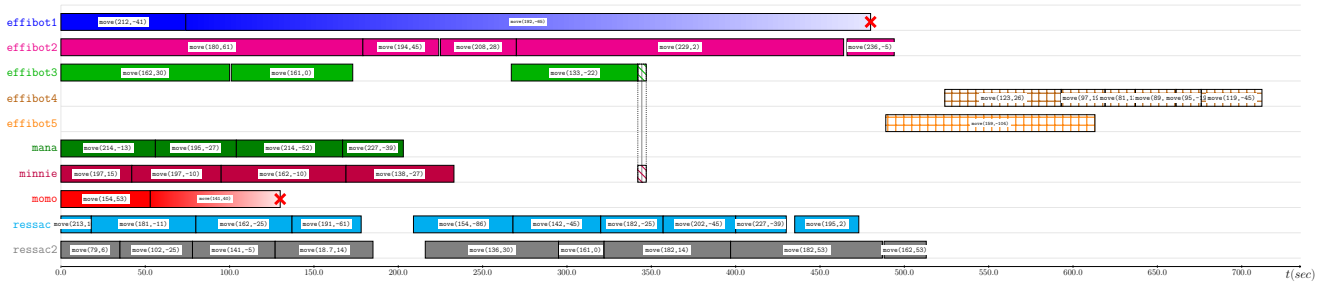


Fig. 22 Field experiment: temporal representation of the plan executed by the robots on a mission with 2 robots being broken. One can see that at t=500s approximately, the team is notified of the fact that two robots are broken (DEAD status). The plan is repaired and the robot *Effibot4* and *Effibot5* are given the actions to observe the relevant points. In this test, only one meeting was performed.

(ICRA) 2016-June:5491–5498, DOI 10.1109/ICRA.2016.7487763, URL

6. Barbulescu L, Rubinstein ZB, Smith SF (2010) Distributed coordination of mobile agent teams: the advantage of planning ahead. In: International Conference on Autonomous Agents and MultiAgent Systems (AAMAS), Toronto, Canada
7. Bechon P, Barbier M, Infantes G, Lesire C, Vidal V (2014) HiPOP : Hierarchical Partial-Order Planning. In: Starting AI Researcher Symposium (STAIRS), Prague, Czech Republic
8. Bechon P, Barbier M, Infantes G, Lesire C, Vidal V (2015) Using hybrid planning for plan reparation. In: European Conference on Mobile Robots (ECMR), Lincoln, UK
9. Bercher P, Biundo S, Geier T, Hoernle T, Nothdurft F, Richter F, Schattenberg B (2014) Plan, Repair, Execute, Explain - How Planning Helps to Assemble your Home Theater. In: International Conference on Automated Planning & Scheduling (ICAPS), Portsmouth, NH, USA
10. Bidot J, Schattenberg B, Biundo S (2008) Plan Repair in Hybrid Planning. In: German Conferance on Artificial Intelligence (KI), Kaiserslautern, Germany
11. Blum AL, Furst ML (1997) Fast planning through planning graph analysis. Artificial intelligence 90(1–2)
12. Boella G, Damiano R (2002) A replanning algorithm for a reactive agent architecture. In: International Conference on Artificial Intelligence: Methodology, Systems, and Applications (AIMSA), Varna, Bulgaria
13. Brinon-Arranz L, Schenato L, Seuret A (2015) Distributed Source Seeking via a Circular Formation of Agents under Communication Constraints. IEEE Transactions on Control of Network Systems 3(2)
14. Castillo L, Fdez-Olivares J, García-Pérez Ó, Palao F (2006) Efficiently handling temporal knowledge in an HTN planner. In: International Conference on Automated Planning & Scheduling (ICAPS), Cumbria, UK
15. Cubber GD, Doroftei D, Baudoin Y, Serrano D, Chintamani K, Sabino R, Ourevitch S (2012) ICARUS: An EU-FP7 project Providing Unmanned Search and Rescue Tools. In: IROS 2012 Workshop ROSIN, Vilamoura-Algarve, Portugal
16. Dames P (2017) Distributed multi-target search and tracking using the PHD filter. International Symposium on Multi-Robot and Multi-Agent Systems (MRS) pp 1–8, DOI 10.1109/MRS.2017.

8250924, URL
17. Dechter R, Meiri I, Pearl J (1991) Temporal constraint networks. Artificial intelligence 49(1)
18. Drabble B, Dalton J, Tate A (1997) Repairing plans on-the-fly. In: International Workshop on Planning and Scheduling for Space (IWPSS), Oxnard, CA, USA
19. Dvorak F, Bartak R, Bit-Monnot A, Ingrand F, Ghallab M (2014) Planning and Acting with Temporal and Hierarchical Decomposition Models. In: International Conference on Tools with Artificial Intelligence (ICTAI), Limassol, Cyprus
20. Echeverria G, Lemaignan S, Degroote A, Lacroix S, Karg M, Koch P, Lesire C, Stinckwich S (2012) Simulating complex robotic scenarios with MORSE. In: International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR), Tsukuba, Japan
21. Erol K, Nau DS, Subrahmanian V (1995) Complexity, decidability and undecidability results for domain-independent planning. Artificial Intelligence 76(1-2)
22. Fikes RE, Nilsson NJ (1972) STRIPS: A new approach to the application of theorem proving to problem solving. Artificial intelligence 2
23. Fox M, Long D (2003) PDDL2.1: An extension to PDDL for expressing temporal planning domains. Journal of Artificial Intelligence Research (JAIR) 20,
24. Fox M, Gerevini A, Long D, Serina I (2006) Plan stability: replanning versus plan repair. In: International Conference on Automated Planning & Scheduling (ICAPS), Cumbria, UK
25. Franchi A, Stegagno P, Oriolo G (2016) Decentralized multi-robot encirclement of a 3D target with guaranteed collision avoidance. Autonomous Robots 40(2):245–265, DOI 10.1007/s10514-015-9450-3, URL ,
26. Garzón M, Valente J, Roldán JJ, Cancar L, Barrientos A, Del Cerro J (2016) A Multirobot System for Distributed Area Coverage and Signal Searching in Large Outdoor Scenarios. Journal of Field Robotics 33(8):1087–1106, DOI 10.1002/rob.21636, URL
27. Gerevini A, Serina I (2000) Fast Plan Adaptation through Planning Graphs: Local and Systematic Search Techniques. In: International Conference on Artificial Intelligence Planning Systems (AIPS), Toronto, Canada
28. Goldman R (2006) Durative Planning in HTNs. In: International Conference on Automated Planning & Scheduling (ICAPS), Cumbria, UK
29. Hart PE, Nilsson NJ, Raphael B (1968) A Formal Basis for the Heuristic Determination of Minimum Cost Paths. IEEE Transactions on Systems Science and Cybernetics 4(2):100–107, DOI 10.1109/TSSC.1968.300136
30. Hashmi MA, Seghrouchni AEF (2010) Merging of Temporal Plans Supported by Plan Repairing. In: International Conference on Tools with Artificial Intelligence (ICTAI), Arras, France
31. Haslum P, Geffner H (2000) Admissible Heuristics for Optimal Planning. In: International Conference on Artificial Intelligence Planning Systems (AIPS), Toronto, Canada
32. Hoffmann J, Nebel B (2011) The FF planning system: Fast plan generation through heuristic search. Journal of Artificial Intelligence Research (JAIR) 14(1)
33. Hoffmann J, Porteous J, Sebastia L (2004) Ordered landmarks in planning. Journal of Artificial Intelligence Research (JAIR) 22(1)
34. Kambhampati S, Mali A, Srivastava B (1998) Hybrid planning for partially hierarchical domains. In: National Conference on Artificial Intelligence (AAAI), Madison, WI, USA
35. Khadka S, Chung JJ, Tumer K (2017) Memory-augmented multi-robot teams that learn to adapt. International Planning Competition (IPC) URL
36. Krogt RVD, Weerdt MD (2005) Plan Repair as an Extension of Planning. In: International Conference on Automated Planning & Scheduling (ICAPS), Monterey, CA, USA
37. Lesire C, Infantes G, Gateau T, Barbier M (2016) A distributed architecture for supervision of autonomous multi-robot missions - application to air-sea scenarios. Auton Robots 40(7)
38. Luo L, Chakraborty N, Sycara K (2013) Distributed algorithm design for multi-robot task assignment with deadlines for tasks. In: International Conference on Robotics and Automation (ICRA), Karlsruhe, Germany
39. Marconi L, Melchiorri C, Beetz M, Pangercic D, Siegwart R, Leutenegger S, Carloni R, Stramigioli S, Bruyninckx H, Doherty P, Kleiner A, Lippiello V, Finzi A, Siciliano B, Sala A, Tomatis N (2012) The SHERPA project: smart collaboration between humans and ground-aerial robots for improving rescuing activities in alpine environments. In: International Symposium on Robotic Reseach (ISRR), College Station, TX, USA
40. McDermott D, Ghallab M, Howe A, Knoblock C, Ram A, Veloso M, Weld DS, Wilkins DE (1998) PDDL-the planning domain definition language. Tech. rep., Yale Center for Computational Vision

and Control
41. Moarref S, Kress-Gazit H (2017) Decentralized control of robotic swarms from high-level temporal logic specifications. International Symposium on Multi-Robot and Multi-Agent Systems (MRS) pp 17–23, URL
42. Nau D, Ghallab M, Traverso P (2004) Automated Planning: Theory & Practice. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA
43. Nau DS, Cao Y, Lotem A, Muñoz-Avila H (1999) SHOP: Simple hierarchical ordered planner. In: International Joint Conference on Artificial Intelligence (IJCAI), Stockholm, Sweden
44. Nau DS, Au TC, Ilghami O, Kuter U, Murdock JW, Wu D, Yaman F (2003) SHOP2: An HTN planning system. Journal of Artificial Intelligence Research (JAIR) 20(1),
45. Otte M, Kuhlman M, Sofge D (2017) Multi-robot task allocation with auctions in harsh communication environments. International Symposium on Multi-Robot and Multi-Agent Systems (MRS) pp 32–39, URL
46. Parker L (2008) Distributed intelligence: overview of the field and its application in multi-robot systems. Journal of Physical Agents 2(2)
47. Pei Y, Mutka MW (2012) Steiner traveler: Relay deployment for remote sensing in heterogeneous multi-robot exploration. In: International Conference on Robotics and Automation (ICRA), St. Paul, MN, USA
48. Penberthy JS, Weld DS (1992) UCPOP: A sound, complete, partial order planner for ADL. In: International Conference on Principles of Knowledge Representation and Reasoning (KR), Cambridge, MA, USA
49. Penumarthi PK, Li AQ, Banfi J, Basilico N, Amigoni F, O'Kane J, Rekleitis I, Nelakuditi S (2017) Multirobot Exploration for Building Communication Maps with Prior from Communication Models. International Symposium on Multi-Robot and Multi-Agent Systems (MRS) URL
50. Ponda S, Redding J, Choi HL, How JP, Vavrina M, Vian J (2010) Decentralized planning for complex missions with dynamic communication constraints. In: American Control Conference (ACC), Baltimore, MD, USA
51. Portugal D, Rocha RP (2016) Cooperative multi-robot patrol with Bayesian learning. Autonomous Robots 40(5):929–953, DOI 10.1007/s10514-015-9503-7, URL
52. Pralet C, Lesire C (2014) Deployment of mobile wireless sensor networks for crisis management: a constraint-based local search approach. In: International Conference on Principles and Practice of Constraint Programming (CP), Lyon, France
53. Richter S, Helmert M, Westphal M (2008) Landmarks Revisited. In: International Conference on Artificial Intelligence (AAAI), Chicago, IL, USA
54. Röger G, Eyerich P, Mattmüller R (2008) TFD: A numeric temporal extension to Fast Downward. In: International Planning Competition (IPC), Syndey, Australia
55. Schattenberg B (2009) Hybrid Planning And Scheduling. PhD thesis, Ulm University, Institute of Artificial Intelligence
56. Stock S, Mansouri M, Pecora F, Hertzberg J (2015) Online task merging with a hierarchical hybrid task planner for mobile service robots. In: International Conference on Intelligent Robots and Systems (IROS), Hamburg, Germany
57. Vidal V (2011) YAHSP2: Keep it simple, stupid. In: International Planning Competition (IPC), Freiburg, Germany
58. Warfield I, Hogg C, Lee-Urban S, Muñoz-Avila H (2007) Adaptation of Hierarchical Task Network Plans. In: FLAIRS Conference, Key West, FL, USA
59. Younes H, Simmons R (2003) VHPOP: Versatile heuristic partial order planner. Journal of Artificial Intelligence Research (JAIR) 20