Noname manuscript No.
(will be inserted by the editor)

Using Pre-Computed Knowledge for Goal Allocation in Multi-Agent Planning

Nerea Luis[1] · Tiago Pereira [2,3,4] · Susana Fernández[1] · António Moreira[2,4] · Daniel Borrajo[1] · Manuela Veloso[3]

Received: date / Accepted: date

Abstract Many real-world robotic scenarios require performing task planning to decide courses of actions to be executed by (possibly heterogeneous) robots. A classical centralized planning approach has to find a solution inside a search space that contains every possible combination of robots and goals. This leads to inefficient solutions that do not scale well. Multi-Agent Planning (MAP) provides a new way to solve this kind of tasks efficiently. Previous works on MAP have proposed to factorize the problem to decrease the planning effort i.e dividing the goals among the agents (robots). However, these techniques do not scale when the number of agents and goals grow. Also, in most real world scenarios with big maps, goals might not be reached by every robot so it has a computational cost associated. In this paper we propose a combination of robotics and planning techniques to alleviate and boost the computation of the goal assignment process. We use Actuation Maps (AMs). Given a map, AMs can determine the regions each agent can actuate on. Thus, specific information can be extracted to know which goals can be tackled by each agent, as well as cheaply estimating the cost of using each agent to achieve every goal. Experiments show that when information extracted from AMs is provided to a multi-agent planning algorithm, the goal assignment is significantly faster, speeding-up the planning process considerably. Experiments also show that this approach greatly outperforms classical centralized planning.

Keywords Multi Agent Planning · Actuation Maps · Goal Allocation · Robotics · Distributed Planning · Path Planning

[1]Universidad Carlos III de Madrid
Avda. Universidad 30, Madrid, Spain
Tel.: +34916248898
E-mail: nluis@inf.uc3m.es

[2]Faculty of Engineering, University of Porto, Portugal
[3]Carnegie Mellon University, Pittsburgh, USA
[4]INESC-TEC, Porto, Portugal

1 Introduction

Real-world robotic scenarios, in which a set of robots need to solve a certain amount of tasks, usually require the combination of path-planning and motion-planning techniques. An example of this type of scenarios is the coverage problem, which consists of distributing the space among the set of robots, so that each one explores a certain region of the environment. The coverage problem planning task is to find a route for each robot so that all the feasible space is covered by the robots' actuators, while minimizing the execution time. Vacuum cleaning robots can be potential candidates for this problem. We assume that we have a team of heterogeneous robots with different sizes. While the smallest robot can reach more areas, a bigger robot cleans a wider area while traveling a smaller distance. Nevertheless, other similar problems can also be solved with our contributed technique e.g. heterogeneous robots executing surveillance tasks, cooperative mapping of the environment or search and rescue tasks. As long as there exist (1) some navigation graph where we can extract information to help the planner; and (2) agents with similar or different capabilities, it will be a potential domain to solve with our approach. We have encoded our problem as a Multi-Agent Planning (MAP) task. Automated planning is the field of Artificial Intelligence which deals with the computation of plans. A plan is a sequence of actions that, if executed in order from the initial state, reaches another state where all the feasible goals are achieved.

The problem is modeled with the standard PDDL language [14]. For that purpose, we use a discrete representation of the map, i.e., a 2D grid of waypoints. Robots can move from one waypoint to another as long as they are grid neighbors and do not collide with obstacles. Moreover, robots can actuate other waypoints if their distance to the robot's current position is less than their actuation radius. Here we consider actuation as a robot performing an operation that results in some task being executed in the environment. Therefore, the actuation capabilities could be modeled not only as the operations where a robot changes its surrounding, but also as any perception-like operation e.g. the planning problem to solve would be mathematically equivalent if the overall goal was not to clean all reachable space, but instead to measure the temperature everywhere. Our framework allows to solve planning problems where sensing operations need to be executed at specific waypoint locations. e.g. mapping the Wi-Fi signal strength in buildings, taking measurements of temperature and humidity on a set of pre-defined locations such as a computer cluster and server sites, different kinds of inspection or even surveillance problems. For vacuum cleaning robots, the sensing is performed through the robot's actuator that cleans the floor. Cleaning a specific waypoint location can be seen as the robot accomplishing a specific cleaning task, and therefore the coverage problem would represent a robot moving through a map and executing multiple tasks, i.e., actuating on all available waypoints on the environment with the objective of cleaning all reachable regions of the environment. In the cleaning case, the actuation capabilities of each robot depends on the specific actuator each robot uses, e.g., its shape and size. In general, for each robot and domain a different actuation model can be considered e.g the actuation radius being smaller than the robot's footprint, the actuation range being exactly the same as the robot's footprint, or extending it further than the footprint. An example of the last case is a mobile manipulator, where an arm can be extended and actuate on regions beyond the robot's range in terms of its shape and footprint. In this paper, for

simplicity we only focus on two cases. First, circular robots where the actuation range is smaller or equal to the robot footprint's radius, and second, any-shape robots whose actuation range is the same as its footprint.

We expect the planner to output a plan that accomplishes all the feasible actuation tasks, by moving the robots to all the reachable locations from where they can actuate the goal waypoints. Given the robot heterogeneity, some tasks might only be feasible for a subset of the robots.

From a MAP point of view, the multi-robot problem we propose forces us to deal with two issues regarding the performance of the planning process: (1) the size of the search space grows with the number of waypoints and goals; and (2) some goals are not feasible for some robots. On one hand, real-world scenarios are big enough to make almost impossible for a planner to solve this problem in a reasonable amount of time by just assigning all goals to all agents (following a centralized planning approach). On the other hand, some Multi-Agent planners invoke a goal-allocation phase before starting to plan to decrease the effort of computing individual plans [4], [24]. During goal allocation, a relaxed plan is computed per goal and robot to either return an estimated cost or to identify unfeasibility. This process would be repeated multiple times, concretely $|Agents| \times |Goals|$ resulting in a huge loss of computation time, especially to identify unfeasible goals i.e exploring most of the search space.

Therefore, we contribute a methodology that uses Actuation Maps (AMs) to extract path-planning related information. That information is used to later boost the performance of a multi-agent planner. In concrete, we use these maps as a preprocessing step to speed-up the goal assignment phase. Actuation maps are built only once before the planning process, one per robot, at a very low cost in comparison to the impact on time savings observed later in goal assignment. The AMs, from the robot-dependent reachability maps [27], not only determine the feasibility of each pair robot-goal but also allow us to efficiently compute an estimated cost of achieving that goal. As a result, the planner receives the estimated cost information as input, and saves time by simplifying the goal allocation computation, directly assigning goals to robots.

In our previous work, we showed that combining a preprocessing operation with MAP could bring huge savings on planning time [29]. More specifically, we used AMs to estimate the cost of each robot when actuating each goal waypoint. Then, we introduced those estimations into the goal assignment phase to distribute the goals among the robots and plan individual paths that together solved the coverage problem.

In this paper, we extend that approach with various contributions:

- Introducing collision avoidance, with a replanning phase solving the conflicts between the individual paths of each robot.
- Generalization of robot shape models, from circular only to any-shape, updating both the Actuation Maps and the information extraction of generate the PDDL problem.
- Evaluation of four different configurations of our approach against two centralized planners and three multi-agent planners.
- Description of the architecture -now organized into four modules- and detailed explanation of the preprocessing step.

- Description of the general problems where our approach can have a huge impact plus the coverage problem.
- Description of the Goal Allocation process and the deletion of unfeasible goals.
- Description of the MAP algorithm - including a new version that detects and solves interactions among agents.

This paper is structured as follows: in Section 2 we describe the kind of problems for what our approach has been designed for and include some planning formalizations. Then, on Section 3 we describe the coverage problem we want to solve. On Section 4, our approach is presented. Section 5 explains in detail the preprocessing step, which is our main contribution. After that, Section 6 contains the description of the MAP algorithm. On Section 7 we briefly extend our approach to MAP problems that involve interactions. Then, on Section 8, we extend the formalization of circular robots to any-shape robots. On Section 9 we show illustrative experimental results of our algorithm on different scenarios. Finally, we discuss the related work and we present our conclusions and directions for future work.

2 General Problem Formulation

Our approach can be easily applied to any robotic problem that involves at least the following elements:

- a map of the environment.
- a set of potential tasks to be executed by an agent over the environment.
- a way to model that scenario into a PDDL domain and problem.

The map can be modeled in different ways (e.g. a navigation graph, a grid of waypoints, a building floor etc.). The model is useful for our approach as long as the features related to the information of the environment can be extracted. In this paper the map is generated as an Actuation Map.

The set of potential tasks can vary depending on the problem to solve. In this paper we are focusing on the coverage problem and as a result it is enough for the robots to move through the environment. Some other alternatives tasks would be looking for objects, opening doors or achieve some clients' orders through the environment.

The potential of our approach relies on the ability to extract information from the map related to the tasks. The aim is to transform that information into a set of estimation costs that can speed up the planning process i.e we have computed the cost as the distance to each of the waypoints on the coverage problem. Some other alternative tasks would be the distance to a required object, the dangerousness or reliability of a path, the features of a robot (velocity, arms) etc.

In order to transform this kind of problems into PDDL we have to model (1) a domain; (2) a problem; and compute (3) a set of estimated costs. The domain and problem are a lifted representation in predicate logic of the planning task. However, most of the planners always perform a grounding transformation from the domain and problem to generate the planning task.

Definition 1 Planning Task (Single Agent). A single-agent STRIPS planning task [12] is a tuple $\Pi = \langle F, A, I, G \rangle$, where F is a set of propositions, A is a set of instantiated actions, $I \subseteq F$ is an initial state, and $G \subseteq F$ is a set of goals.

Each action $a \in A$ is described by (1) a set of preconditions (pre(a)) that represent literals that must be true in a state to execute the action; (2) and a set of effects (eff(a)), which are literals that are expected to be added (add(a) effects) or removed (del(a) effects) from the state after the execution of the action. The definition of each action might also include a cost $c(a)$ (the default cost is one). As we are working with multiple agents, we consider a MAP formalization where a set of m agents, $\Phi = \{\phi_1, \ldots, \phi_m\}$, has to solve the given task.

Definition 2 Multi Agent Planning task. The MAP task is formed by a set of planning subtasks, one for each agent, $M = \{\Pi_1, \ldots, \Pi_m\}$ where M refers to the MAP task. Each planning subtask Π_i includes only the facts, actions, goals and initial state related to the agent ϕ_i.

Last, we define the estimated cost per agent and goal.

Definition 3 Estimated cost per agent and goal (EC). $EC = \{(g, \phi_i, c) \mid g \in G, \phi_i \in \Phi, c = C(g, \phi_i)\}$ such that c represents the cost to reach the goal position g from the agent's initial state. If a goal cannot be reached by an agent, c will be ∞.

Usually, these estimated costs are computed to divide the goals among the agents before the planning process starts. In planning, c is obtained with a heuristic function. However, in our approach, we compute c using a function based on the path planning and problem features. Even though MAP is domain-independent, our function to compute the estimated costs is domain-dependent and should be set up differently on each domain, though it would be very similar to this one in most related robotic domains.

3 The Coverage problem description

In this work we only consider heterogeneous teams of circular robots that actuate in a 2D environment, where the world is represented by a 2D image that can be down sampled to a 2D grid of waypoints. The AM gives information about the actuation capabilities of each robot, as a function of robot size and initial position [27]. In the example with vacuum cleaning robots, the AM represents the regions of the world each robot can clean.

At first, we assume that robots are circular and thus the only *Robot* feature is its size, with 2D grid positions being rotation-invariant. Other shapes can also be trivially considered in our approach by extending the PDDL domain file to take into consideration robot orientation as well.

As it was previously said, we modeled the domain and problem using PDDL. The domain has two types of objects: *robots*, which act as agents; and *waypoints*, which represent positions in the discretized world. We consider a coverage problem, where the goal is to have the robots actuating on waypoints. In this version of the coverage problem, robots actuate a waypoint if it is inside its actuation radius. Thus, they do not need to be exactly placed on the waypoint to actuate it.

Therefore, the set G is a list of waypoints to actuate on (positions that need to be covered). The PDDL domain we created has four predicates:

- `At (robot, waypoint)`: defines the robot position;

- `Connected (robot, waypoint, waypoint)`: establishes the connectivity between waypoints, specified for each robot, and given the robot heterogeneity, some connections might be traversable by some robots and not by others;
- `Actuated (waypoint)`: indicates which waypoints were already actuated; this predicate is used to specify goals;
- `Reachable (robot, waypoint, waypoint)`: shows which waypoints can be actuated by a robot when located on a different waypoint location.

Robots have to actuate every waypoint in G. The waypoints, when connected, generate a navigation graph for a certain robot. The two actions that are defined in the domain are called `navigate` (Listing 1) and `actuate` (Listing 2). The first one moves a robot from its current waypoint location to a neighbor waypoint as long as both are connected. The second action is used to mark a waypoint as actuated if it is identified as `reachable` from the robot's current waypoint location i.e the waypoint was located inside the robot's actuation radius on the real environment. `Navigate` and `actuate` are the two actions that can be executed by an agent when it is placed on a waypoint. Both `navigate` and `actuate` have as effect the predicate `actuated`.

Listing 1 Action Navigate in PDDL

```
(:action navigate
  :parameters (?r - robot ?y - waypoint ?z - waypoint)
  :precondition (and
                (connected ?r ?y ?z) (at ?r ?y)
   )
  :effect (and
          (not (at ?r ?y))(at ?r ?z) (actuated ?z)
   )
)
```

Listing 2 Action Actuate in PDDL

```
(:action actuate
   :parameters (?r - robot ?y - waypoint ?z - waypoint)
   :precondition (and
         (at ?r ?y)(reachable ?r ?y ?z)
   )
   :effect (and
         (actuated ?z)
   )
)
```

In order to generate a PDDL problem, the waypoints' grid resolution is defined in advance using a discretization step. After that, a navigation graph and a set of reachable waypoints are defined for each robot, taking into account their physical characteristics. All this information is generated on the preprocessing step, further explained in Section 5.

4 Architecture

As it was previously said, this work combines Actuation Maps (AM) with Multi-Agent Planning (MAP). The contributed architecture can be seen in Figure 1. It has been divided into four modules and receives as input the map of the environment, the general knowkedge related to the task to solve and the features of the set of robots. The aim of each module is described as follows:

1. Actuation Maps module: it is in charge of generating the AMs for each given robot. It also extracts the map features that can potentially alleviate the planning process e.g path-planning features, transforms them into a set of estimation costs and generates the planning problem in PDDL. It is explained on Section 5.1.
2. Multi-Agent Planning Task Generation module: once the outputs from the prior module and the domain are received as input, the goal assignment process is launched. This module is in charge of dividing the goals among the agents following some goal-strategy. Then, a set of domain and problem is generated for each agent, which is known as factorization. It is explained on Section 5.2.
3. Multi-Agent Planning Algorithm module: the individual planning process and the merging phase are run on this module. It is explained on Section 6.
4. Conflicts solver module: if any interactions need to be solved, this module employs a plan-reuse-planner to fix them. It is explained on Section 7.

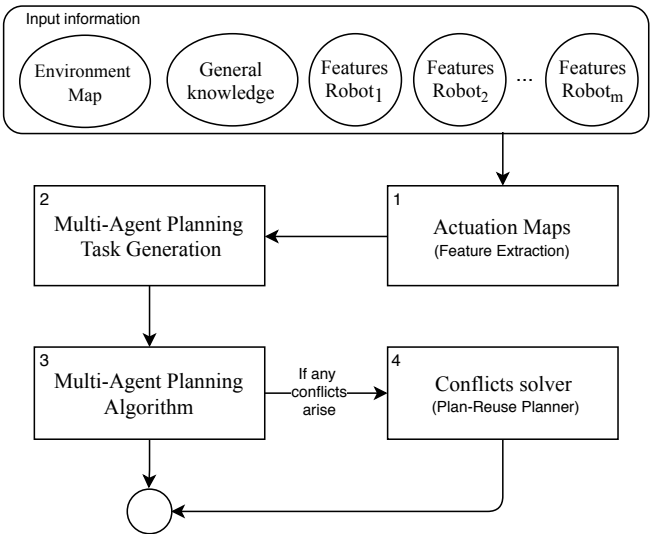


Fig. 1 Complete architecture that combines Actuation Maps and Multi-Agent Planning

The following subsections explain the essential information regarding both fields: Actuation Maps and Multi-Agent Planning. The aim is to fully understand later the preprocessing step and the Multi-Agent Planning algorithm.

4.1 Actuation Maps

Our system receives as input the *Environment map* which represents a 2D environment (e.g. building floor plan) and m *Robot* models with the agents' features. There is a third input provided by the user that refers to the *General knowledge* of the environment (i.e: tasks to solve). These three inputs represent the input information described in Figure 1.

We briefly summarize here the process of building the Actuation Space [27]. We assume there is an occupancy grid map, i.e., a gray-scale image representing the environment (Figure 7(a)). In this image each pixel has a value with the probability of the corresponding world position being occupied by an obstacle. This occupancy grid map is first transformed into a binary image of free and obstacle pixels, using a fixed threshold.

We define $\mathcal{G}$ as the set with all pixel positions from the input binary image. This input image (*Environment map* in Figure 5) is represented by $\mathcal{M}$, the set with the obstacle pixel positions. We define the structuring element as an image that represents the robot shape. Using the robot-model image, $\mathcal{R}_i$ is the set with pixel positions from a circle with radius equal to the robot size. The morphological operation dilation on the obstacle set $\mathcal{M}$ by $\mathcal{R}_i$ is:

$$\mathcal{M} \oplus \mathcal{R}_i = \bigcup_{\mathbf{r} \in \mathcal{R}_i} \mathcal{M}_{\mathbf{r}} \tag{1}$$

where $\mathcal{M}_{\mathbf{r}}$ is the translation of $\mathcal{M}$ by vector $\mathbf{r}$.

The visual output of applying this dilation operation to a map of obstacles is the inflation of obstacles by the robot size. The free configuration space, $\mathcal{C}^{free}$, is then defined as:

$$\mathcal{C}_i^{free} = \{\mathbf{p} \in \mathcal{G} \mid \mathbf{p} \notin \mathcal{M} \oplus \mathcal{R}_i\} \tag{2}$$

where $\mathcal{G}$ is the grid set with all the pixel positions. The free configuration space represents the feasible positions for the robot center, but does not give any information about the regions that can be actuated by the robot.

In order to determine the Actuation Space instead, the partial morphological closing operation is used. Morphological closing is a dilation operation followed by a morphological erosion. Because dilation and erosion are dual operations, the morphological closing of obstacles (erosion applied to the image with inflated obstacles) is equivalent to the dilation of free configuration space.

However, the morphological closing cannot be used to determine the Actuation Space, because it does not consider which points are reachable from the initial robot position, and different initial positions do change the overall actuation capabilities. The partial morphological closing was introduced in order to consider the initial robot position when determining the Actuation Space. In order to use the partial morphological closing, the algorithm needs to find the navigable regions first. The set of navigable regions from a starting robot point $\mathbf{r}_i^0$ is always a subset of $\mathcal{C}_i^{free}$.

$$\mathcal{L}_i(\mathbf{r}_i^0) = \{\mathbf{p} \in \mathcal{G} \mid \mathbf{p} \text{ connected to } \mathbf{r}_i^0 \wedge \mathbf{p} \in \mathcal{C}_i^{free}\} \tag{3}$$

The navigable set $\mathcal{L}_i(\mathbf{r}_i^0)$ is the set of points that are connected to the initial position $\mathbf{r}_i^0$ through a path of adjacent cells in the free configuration space.

In Figure 2 we show a simulated map with 2 robots with different sizes, and the respective navigable space.

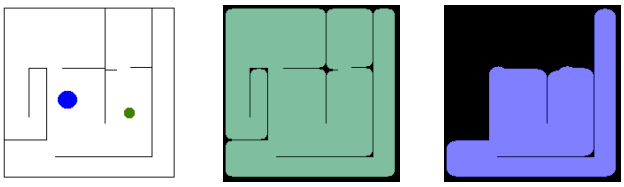
(a) Map (b) Robot1 (c) Robot2

Fig. 2 Simulated map and two heterogeneous robots with different sizes in (a); colored regions represent the navigable space, $\mathcal{L}_i(\mathbf{r}_i^0)$, for 2 robots with different sizes, depending on size and initial position of robots.

Finally, by applying the the second dilation operation of the morphological closing to the navigable set (subset of $\mathcal{C}^{free}$) instead of applying it to the free configuration space, we obtain the partial morphological closing operation. The Actuation Space is thus the dilation of the navigable space.

$$\mathcal{A}_i(\mathbf{r}_i^0) = \mathcal{L}_i(\mathbf{r}_i^0) \oplus \mathcal{R}_i \tag{4}$$

In Figure 3 we show the Actuation Spaces after applying the partial morphological closing operation to the original map. The Actuation Space represents what the robot can actuate from any point reachable from its initial position.



(a) Map with 2 Robots (b) Actuation Space 1 (c) Actuation Space 2

Fig. 3 Colored regions in Figure (b) and Figure (c) represent actuation spaces for respective robots, i.e. the points in the environment that each robot can actuate, depending on their size and initial position shown in Figure (a).

As an example, we can consider again the vacuum cleaning robot case. The configuration space represents the possible center positions for the robot; the actuation space $\mathcal{A}$ represents the regions the robot can clean; and unreachable regions like corners result from the circular shape of the robot.We briefly summarize here the process of building the Actuation Space [27]. We assume there is an occupancy grid map, i.e., a gray-scale image representing the environment. In this image each pixel has a value with the probability of the corresponding world position being occupied by an obstacle. This occupancy grid map is first transformed into a binary image of free and obstacle pixels, using a fixed threshold.

We define $\mathcal{G}$ as the set with all pixel positions from the input binary image. This input image (*Environment Map Image* in Figure 5) is represented by $\mathcal{M}$, the set with the obstacle pixel positions. We define the structuring element as an image

that represents the robot shape. Using the robot-model image, $\mathcal{R}_i$ is the set with pixel positions from a circle with radius equal to the robot size. The morphological operation dilation on the obstacle set $\mathcal{M}$ by $\mathcal{R}_i$ is:

$$\mathcal{M} \oplus \mathcal{R}_i = \bigcup_{\mathbf{r} \in \mathcal{R}_i} \mathcal{M}_{\mathbf{r}} \tag{5}$$

where $\mathcal{M}_{\mathbf{r}}$ is the translation of $\mathcal{M}$ by vector $\mathbf{r}$.

The visual output of applying this dilation operation to a map of obstacles is the inflation of obstacles by the robot size. The free configuration space, $\mathcal{C}^{free}$, is then defined as:

$$\mathcal{C}_i^{free} = \{\mathbf{p} \in \mathcal{G} \mid \mathbf{p} \notin \mathcal{M} \oplus \mathcal{R}_i\} \tag{6}$$

where $\mathcal{G}$ is the grid set with all the pixel positions. The free configuration space represents the feasible positions for the robot center, but does not give any information about the regions that can be actuated by the robot.

In order to determine the Actuation Space instead, the partial morphological closing operation is used. Morphological closing is a dilation operation followed by a morphological erosion. Because dilation and erosion are dual operations, the morphological closing of obstacles (erosion applied to the image with inflated obstacles) is equivalent to the dilation of free configuration space.

However, the morphological closing cannot be used to determine the Actuation Space, because it does not consider which points are reachable from the initial robot position, and different initial positions do change the overall actuation capabilities. The partial morphological closing was introduced in order to consider the initial robot position when determining the Actuation Space. In order to use the partial morphological closing, the algorithm needs to find the navigable regions first. The set of navigable regions from a starting robot point $\mathbf{r}_i^0$ is always a subset of $\mathcal{C}_i^{free}$.

$$\mathcal{L}_i(\mathbf{r}_i^0) = \{\mathbf{p} \in \mathcal{G} \mid \mathbf{p} \text{ connected to } \mathbf{r}_i^0 \wedge \mathbf{p} \in \mathcal{C}_i^{free}\} \tag{7}$$

The navigable set $\mathcal{L}_i(\mathbf{r}_i^0)$ is the set of points that are connected to the initial position $\mathbf{r}_i^0$ through a path of adjacent cells in the free configuration space.

Finally, by applying the the second dilation operation of the morphological closing to the navigable set (subset of $\mathcal{C}^{free}$) instead of applying it to the free configuration space, we obtain the partial morphological closing operation. The Actuation Space is thus the dilation of the navigable space.

$$\mathcal{A}_i(\mathbf{r}_i^0) = \mathcal{L}_i(\mathbf{r}_i^0) \oplus \mathcal{R}_i \tag{8}$$

4.2 Discretization

For the planning problem, it is possible to consider each individual pixel as a waypoint. However, that approach results in a high density of points that would make the planning problem excessively complex. Moreover, there is some redundancy in having points that are too close to each other, as their difference is not significant in terms of the environment size and localization accuracy.

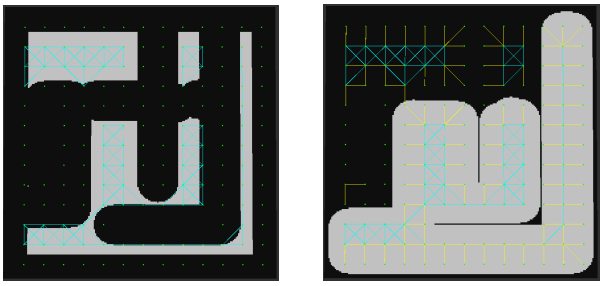
Therefore, we reduced the set of locations from all pixels to a smaller set of locations. We considered again waypoints distributed into a grid, but now the grid-size is greater than one pixel. Then, we can find the connectivity between

points to construct the navigation graph of each robot, shown in Figure 4(a). It is also possible to find which waypoints can be actuated from other waypoints using the distance between them, as shown in Figure 4(b), by considering the maximum actuation radius.

The problem of such discretization is the change in the actuation space topology. Adjusting the position of waypoints could allow a better representation of the topology of the environment, but the multi-robot nature of the problem compromises that solution. In order to deal with multiple robots with different reachable sets, for each agent, we independently adjust the waypoint position -temporarily- in a hidden manner invisible to the other agents. When discretizing each robot's configuration space, we might consider a waypoint as belonging to the free configuration space even if it is strictly outside it, as we assume an error margin to compensate for the discretization error. Nevertheless, we still maintaining the original waypoint position in further steps, such as determining actuation feasibility. When determining the navition graph of each robot, an unreachable waypoint position is moved to the closest point in the reachable space, as long as the adjustment is under a given margin δ. The adjustment is always temporary to the construction of the connectivity graph, i.e., after the connectivity is tested, the waypoint position resets to its default grid position for the next steps.

Moreover, when determining the connectivity of waypoints for the navigation graph, only the eight grid neighbors are considered. A* is then used to determine the real distance between waypoints (e.g., around obstacles), and connectivity is only considered if the real distance is at most a factor of $\alpha = 1.2$ the straight line distance between them.

The grid density is chosen manually in order to adjust the level of discretization. As for the α and δ parameters, they were tuned empirically such as the free space topology is still maintained even while using lower density discretization of the environment. By trial and error, we found empirically that $\alpha = 1.2$ works for all the tested scenarios. As for the δ parameter, we set it to always start with a value of 3, then build the discretized model and verify if it is valid, i.e., if all the



(a) Navigability (b) Actuation

Fig. 4 In Figure (a), the free configuration space for the bigger robot in Figure 2(a), with the discretization waypoints shown as green dots. The blue lines represent the connectivity between waypoints in the navigation graph of the robot. Using parameters δ and α it is possible to maintain the topology of the free configuration space by allowing points in the navigation graph that were originally unfeasible for the robot. In Figure (b), the actuation map of the same robot, and the respective actuation graph represented with yellow lines.

waypoints belonging to the actuation map become feasible for the respective robot in terms of the discretized representation. If not, we increment the parameter until a topologically consistent representation is found (number of feasible goals equals number of waypoints inside actuation map).

Finally, all waypoints that belong to the robot actuation map should be connected to some waypoint of its navigable graph. If that is not the case after the previous steps, we connect the isolated waypoints to the closest navigable vertex in line of sight, even if their distance is greater than the maximum actuation distance, again to compensate for the discretization error. Therefore, while the planner may return an actuate action to cover waypoint A from the navigable waypoint B in the discretized world, a real robot would have to move closer from the waypoint B to waypoint A in order to actuate the latter.

4.3 Multi-Agent Planning

When multiple agents are involved (e.g robots, workers, drivers) we talk about Multi-Agent Planning(MAP) [34]. MAP computes a plan for/by a set of agents that jointly solve a planning problem. Usually, MAP tasks have to deal with some coordination issues among agents or the sharing of resources. From the perspective of MAP, planning domains exhibit a coupling level that ranges from loosely-coupled to tightly-coupled, depending on the degree of interaction between agents plans [7]. Agents on loosely-coupled domains barely interact with each other. Tightly-coupled domains have a considerable level of interaction among agents. This implies solving interactions while planning (agents' communication) or afterwards (conflict solving). Our domain is loosely-coupled, as agents barely interact with each other. However, there might be collisions that are not detected by the planner. A different approach involving collision-detection is further explained on Section 7.

In Multi-Agent Planning two main approaches have been commonly used: centralized and distributed. The centralized approach involves a master agent which knows everything about the agents and the environment. The master agent sees the rest of agents as resources and is also responsible for coordinating and solving the interactions that might arise during the planning process. On the distributed approach each agent builds its own plan synchronously with the rest of the agents. Depending on the amount of communication allowed among the agents, they will need to share their information during the planning process or they have to later merge their plans and solve the conflicts that might have arisen. Our MAP algorithm follows the later, using plan merging [13, 25] to build the solution plan after the individual planning process. Thus, no cost of communication is involved and there is no implementation inside the algorithm regarding communication.

5 Preprocessing

The contributed preprocessing step is shown in Figure 5. This is the point where both techniques, AMs and MAP, are combined and complement each other. Section 5.1 describes the generation of goals, the detection of unfeasible regions and the

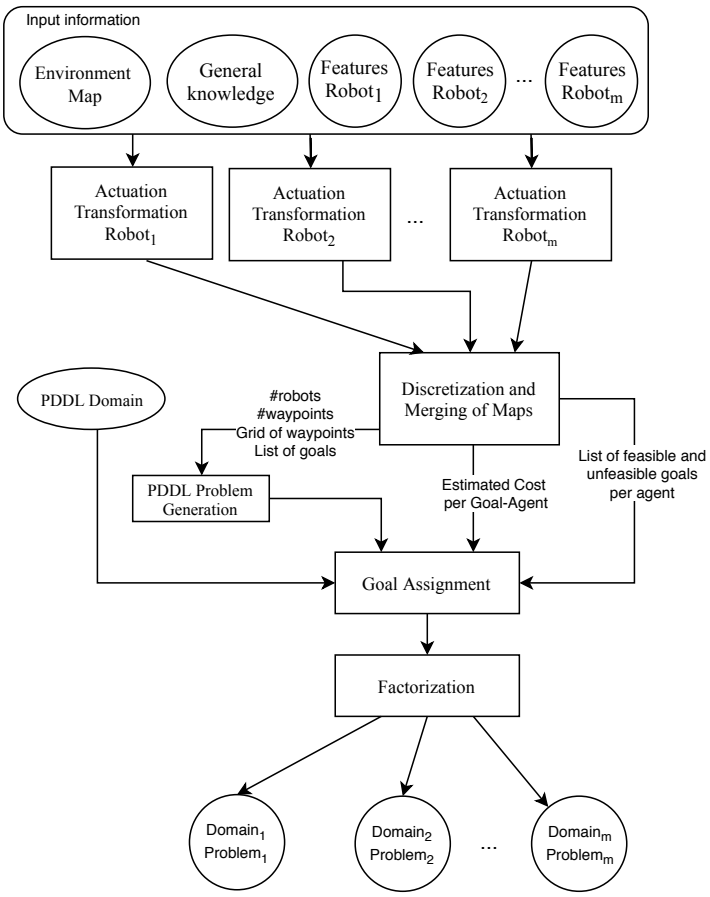


Fig. 5 Preprocessing stage before the planning process starts. First, inputs are processed in order to generate the AM for each agent. Then, a discretization is applied to generate all the required information for planning such as the navigation graph for the PDDL problem, the list of estimated costs etc. Once this information has been generated, GA starts and the MAP problem is divided into subproblems with specific goals assigned to each individual problem.

computation of estimated costs. Section 5.2 describes the process of generating the MAP task and how the task is factorized (divided) in subtasks.

5.1 Extracting information from Actuation Maps

When converting the original map and the Actuation Space to the PDDL description, it is possible to consider each individual pixel as a waypoint in a grid with the size of the whole image. However, that approach results in a high density of points that makes the planning problem excessively complex. There is also redundancy in having points that too close to each other, as their difference is not significant in terms of the environment size and localization accuracy.

Therefore, we reduce the set of possible locations by downsampling the grid of waypoints. The downsampling rate s_r is set manually. If the original pixel

resolution is used, the resulting grid of waypoints $\mathcal{G}'$ contains all pixels and is equivalent to $\mathcal{G}$. Otherwise the set $\mathcal{G}'$ represents the grid waypoint positions after downsampling.

Using the Actuation Space it is possible to very easily find UG, the list of *unfeasible goals per agent*:

$$UG = \{g \in \mathcal{G}', \phi_i \in \Phi \mid g \notin \mathcal{A}_i(\mathbf{r}_i^0)\} \tag{9}$$

The positions in the actuation space $\mathcal{A}_i(\mathbf{r}_i^0)$ are feasible goal positions for actuation tasks. Even though this was used to find the unfeasible list UG, the original Actuation Transformation does not provide any information about the cost for each robot to execute a feasible actuation task.

For that purpose, we contribute the following extension. We build the navigable space $\mathcal{L}_i$ in an iterative procedure, from the starting position $\mathbf{r}_i^0$. In the first iteration we have $\mathcal{L}_i^0(\mathbf{r}_i^0) \leftarrow \{\mathbf{r}_i^0\}$, and then the following rule applies:

$$\begin{aligned}\mathcal{L}_i^j(\mathbf{r}_i^0) = \{\mathbf{p} \in \mathcal{G} \mid \exists \mathbf{q} \in \mathcal{L}_i^{j-1}(\mathbf{r}_i^0) : \mathbf{p} \text{ neighbor of } \mathbf{q} \\ \wedge \mathbf{p} \in \mathcal{C}_i^{free} \wedge \mathbf{p} \notin \mathcal{L}_i^a(\mathbf{r}_i^0) \quad \forall a < j\}\end{aligned} \tag{10}$$

When using this recursive rule to build the navigable space, we guarantee that any point in the set $\mathcal{L}_i^j(\mathbf{r}_i^0)$ is exactly at distance j from the initial position $\mathbf{r}_i^0$.

Furthermore, if we build the actuation space sets with the intermediate navigable sets $\mathcal{L}_i^j(\mathbf{r}_i^0)$,

$$\mathcal{A}_i^j(\mathbf{r}_i^0) = \mathcal{L}_i^j(\mathbf{r}_i^0) \oplus \mathcal{R}_i \tag{11}$$

then the intermediate actuation set $\mathcal{A}_i^j(\mathbf{r}_i^0)$ represents the points that can be actuated by the robot from positions whose distance to $\mathbf{r}_i^0$ is j. The actuation space defined in the previous section can also be alternatively defined as

$$\mathcal{A}_i(\mathbf{r}_i^0) = \{\mathbf{p} \in \mathcal{G} \mid \exists a : \mathbf{p} \in \mathcal{A}_i^a(\mathbf{r}_i^0)\} \tag{12}$$

The Actuation Map is defined for $g \in \mathcal{A}_i(\mathbf{r}_i^0)$:

$$\mathcal{AM}_i(\mathbf{r}_i^0, g) = \min\{j \mid g \in \mathcal{A}_i^j(\mathbf{r}_i^0)\} + 1 \tag{13}$$

The Actuation Map $\mathcal{AM}_i(\mathbf{r}_i^0, g)$ represents, for each $g \in \mathcal{A}_i(\mathbf{r}_i^0)$, the minimum number of actions needed for the robot to actuate the grid waypoint g if starting from the initial position $\mathbf{r}_i^0$, measured in the pixel-based grid $\mathcal{G}$. In Equation 13, the minimum j^* represents the minimum distance (i.e., minimum number of *navigate* actions) needed to travel from $\mathbf{r}_i^0$ to some point from where g can be actuated. The added one in Equation 13 accounts for the one *actuate* action needed to actuate g, after the j^* *navigate* actions needed to reach a place from where the robot can actuate g.

Thus, the cost function c is defined in Equation 14, where s_r is the downsampling rate. The division by s_r transforms the estimated cost of actions measured in the pixel-based grid $\mathcal{G}$, $\mathcal{AM}_i(\mathbf{r}_i^0, g)$, to the respective cost value in the downsampled grid of waypoints $\mathcal{G}'$. The *ceil* function rounds up the result of the division to the smallest integral value that is not less than $\mathcal{AM}_i(\mathbf{r}_i^0, g)/s_r$. The cost function c is domain-dependent and works for the coverage problem. If a different problem is given as input, the cost function should be redefined.

$$c(\mathcal{AM}_i(\mathbf{r}_i^0, g)) = ceil\left(\mathcal{AM}_i(\mathbf{r}_i^0, g)/s_r\right) \tag{14}$$

Finally, the *Estimated Cost per Goal-Agent* list EC is defined in Equation 15.

$$EC = \{\langle g, \phi_i, c\rangle \mid g \in \mathcal{G}' \wedge \phi_i \in \Phi \wedge g \in \mathcal{A}_i(\mathbf{r}_i^0) \wedge c \in c(\mathcal{AM}_i(\mathbf{r}_i^0, g))\} \tag{15}$$

5.2 Multi agent planning task generation

One the discretization of maps has been performed, we have all the information needed to generate the MAP task M, which is formed by a domain (received as input) and a problem (generated through the discretization). The inputs to the Goal Assignment (GA) phase are (1) the PDDL domain; (2) the PDDL problem; (3) the list of estimated costs where c is computed as the number of steps for an agent to reach the goal position g from its initial position; and (4) the list of unfeasible goals $UG = \{g \in G, \phi_i \in \Phi \mid C(g, \phi_i) = \infty\}$. The cost of navigating between two neighbor grid waypoints is 1 unit. As long as EC is provided, UG is not used inside the MAP algorithm. The case when EC is not provided is later explained in this section.

In Multi-Agent systems, in order to perform task allocation [8, 18] some strategy has to be determined or implemented, as the aim it is to divide the MAP task in subtask to alleviate the planning process afterwards. In addition, a goal-assignment strategy (GAS) needs to be chosen to define the way goals are assigned to agents by the system.

In our approach we took the Load-Balance (LB) strategy previously defined in [2], that first calculates $k = \left\lceil \frac{|G|}{|\Phi|} \right\rceil$, which represents the average number of goals per agent. Then, it assigns each goal $g \in G$ to the agent $\phi_i \in \Phi$ that achieves g with the least cost. This strategy avoids, if possible, assigning more goals than k to each agent. The LB assignment strategy is used when minimizing the maximum number of actions per agent (makespan). As a second option, we also took the Best-Cost (BC) strategy also defined before in [2], which simply assigns each goal to the agent that can achieve it with the least cost. The BC strategy is used when minimizing the total number of actions over all robots (plan length).

As in [2], only when the information about estimated cost per pair robot-goal is not available for some reason, our MAP algorithm would perform GA computing a relaxed plan using the FF heuristic [19]. This is not a contribution of the paper itself, as it was already in [24]. However, in that work, when a goal was unfeasible for every agent, it was assigned to all of them. In our approach, when the goal has been identified as unfeasible by all agents, the relaxed plan is not computed for that pair robot-goal and the goal is not included into the M task. This behavior is not common in classical deterministic Automated Planning, as planners expect that the problem does not contain any unfeasible goal. As our approach separates goal allocation from planning, we can easily deal with unfeasible goals. This small contribution gives us more flexibility when working for real environments, as it is better to obtain a plan that solves 95% of the goals than just failing during planning. To plan using soft-goals [23] or working on oversubscription planning [31][17] would have been other ways to deal with unfeasibility, but they are out

of the scope of this work. In summary, there are two contributions to the GA process: (1) the detection and deletion of unfeasible goals is a contribution that helps not only on skipping the computation of those relaxed plans but also avoids the planning process to fail; and (2) to use information from AMs, as the algorithm receives and processes the estimated costs from the AMs to skip the computation of the relaxed plans.

The first step is to allocate the feasible goals to the agents. This step uses the information of estimated costs received from AMs. Goal assignment phase (GA) returns as output (1) a subset of Φ' agents, $\Phi' = \{\phi_1, \ldots, \phi_n\}$, that will be the only ones who will plan to solve the problem; and (2) a new MAP task $M' = \{\Pi_1, \ldots, \Pi_n\}$. As a result, a specific PDDL domain and problem will be generated for each ϕ'_i agent which only includes the goals each agent has to achieve. As it previously mentioned, if a goal is unfeasible for all the Φ' agents, the MAP algorithm will discard it from the new MAP task M', so that the process of planning will not fail. The ouput of this process is a pair (domain, problem) for each agent as in Figure 5 after the *Factorization* step and in Equation 16.

$$M' = \langle \Pi_1, ..., \Pi_n \rangle = \Big\{ \langle D_1, P_1 \rangle, \langle D_2, P_2 \rangle, ..., \langle D_n, P_n \rangle \Big\} \tag{16}$$

The Multi-Agent Planning Algorithm

Inputs: M', SAP
Output: π

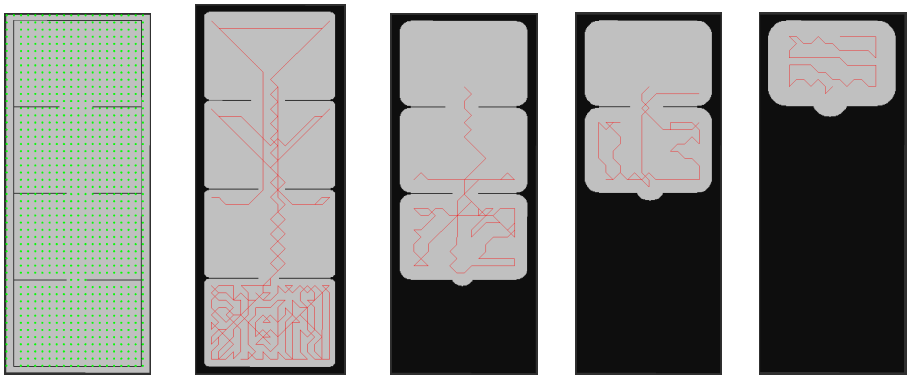
---

1 **Forall** $\phi'_i \in \Phi'$ do π_i =`plan`(Π_i, SAP)
2 π_{seq} = `concatenate`$(\pi_1, \ldots, \pi_n)$ /* where $n = |\Phi'|$ */
3 **If** `valid`(π_{seq})
4 **Then** return π =`parallelize`(π_{seq})
5 **Else** return *no-solution*

Fig. 6 High level description of the MAP algorithm. Inputs: MAP task (M'), Single-Agent Planner (SAP). Output: resulting plan (π) or no solution.

6 The Multi-agent Planning Algorithm

In order to solve the MAP task we use the distributed phase of a previous work of the authors [24]. Our algorithm receives as input a MAP task, which consists of a PDDL domain and problem files (Figure 6) for each agent in $\phi_i \in \Phi'$. On the first step, each agent builds its plan individually (line 1). Then the agents' plans are merged by a simple concatenation of plans in one resulting plan (line 2). If the merged plan is valid, we parallelize the plan (line 4) generated by the merging step. Parallelization is performed in two steps: converting the input total-order plan into a partial-order one by a similar algorithm to [35]; and parallelizing this partial-order plan by ordering actions in the first time step that satisfies all ordering



(a) Waypoints (b) Path 1 (c) Path 2 (d) Path 3 (e) Path 4

Fig. 7 These figures represent the Corridor scenario used in the experiments. The waypoint discretization is shown in Figure (a). The resulting path for each robot is shown in Figures (b) to (e), after solving the planning problem using load balance as goal-strategy. Path 1 belongs to the smallest robot. Path 4, to the biggest one.

constraints in the partial-order plan. The main advantage of parallelization is to be able to return a plan in which on each time step, more than one action can be executed at the same, taking advantage of having multiple robots in the environment as well. As the domain does not have any interactions, there is no need of fixing the plan regarding interactions.

To set up the algorithm, an off-the-shelf planner had to be chosen. The advantage of this MAP algorithm is that any state-of-the-art planner can be easily included on the distributed planning phase without further code modification. Our configuration uses LAMA-UNIT-COST as the planner SAP of the algorithm. LAMA-UNIT-COST corresponds to the first search that LAMA performs, using greedy-best-first with unit costs for actions [30]. The merged plan is validated using VAL [20], the validator from the International Planning Competition (IPC)[1]. On Figure 7 we show a solution example obtained from the MAP algorithm. It corresponds to the scenario called Corridor-High later on the experiments.

7 Dealing with interactions

Real-world robotics environments might imply to deal with potential interactions among robots (at the very least) e.g collisions, sharing a resource, cooperation, etc. On the previous description of the Coverage problem we did not explicitly considered any kind of interactions. Our idea was (1) to test first the scalability of the MAP algorithm; and (2) to generate, as fast as possible, a valid solution. Robots might occasionally collide at some specific step of the solution plan. However, that collision could be easy resolved during execution by forcing one of the robots to wait until the other robot has left the conflict zone. Then, the stopped robot will continue executing the rest of its plan. On the other hand, there is a subarea of Automated Planning called Planning by Reuse that has been widely

[1] http://icaps-conference.org/index.php/main/competitions

employed in areas such as Case-Based Planning [6], or replanning when plan execution fails [15]. Usually, planners that perform plan repair receive three inputs: a domain, a problem and a plan to be fixed. Examples of this kind of planners are LPG-ADAPT [15] or ERRT-PLAN [5]. Therefore, an improvement of our approach is to detect and fix potential collisions right after the individual planning process using an off-the-shelf plan reuse planner. This new feature makes our architecture more robust when executing the solution plan in a real environment. Thus, we slightly changed our PDDL domain to track the collisions by adding a new predicate called `occupied`.

– `Occupied (waypoint)`: indicates that there is a robot on that waypoint.

That predicate is set as a new `precondition` of the navigate action described in Listing 3. This allows the agent to only traverse a connection if the destination waypoint is not occupied by a robot.

Listing 3 Action Navigate that now checks occupied positions

```
(:action navigate
   :parameters (?r - robot ?y - waypoint ?z - waypoint)
   :precondition (and
         (connected ?r ?y ?z) (at ?r ?y)
         (occupied ?y) (not(occupied ?z))
    )
   :effect (and
        (not (at ?r ?y)) (not(occupied ?y))
        (at ?r ?z) (actuated ?z)(occupied ?z)
    )
)
```

The new version of our MAP algorithm is described in Algorithm 8. As the MAP algorithm starts with the individual planning phase, no collisions will be detected at that point (line 1). After concatenation, the solution plan is validated by VAL (line 3). The validator will detect if the `occupied` predicate is true twice on the

The Multi-Agent Planning and Reuse Algorithm

Inputs: M', SAP, R
Output: π

---

1 **Forall** $\phi'_i \in \Phi'$ do π_i =`plan`(Π_i, SAP)
2 π_{seq} = `concatenate`$(\pi_1, \ldots, \pi_n)$ /* where $n = |\Phi'|$ */
3 **If** `valid`(π_{seq})
4 **Then** π =`parallelize`(π_{seq})
5 **Else**
6 π_{reuse} = `plan-reuse`(M', π_{seq}, R)
7 π =`parallelize`(π_{reuse})
8 **If** `valid`(π) return π
9 **Else** return *no-solution*

Fig. 8 MAP algorithm that also includes the plan-reuse phase. Inputs: MAP task (M'), Single-Agent Planner (SAP). Plan-reuse planner (R). Output: resulting plan (π) or no solution.

same waypoint and if the predicate `at` from two robots refers to the same waypoint on the same planning step. If so, the plan will be invalid. As a result, the M' task and the invalid plan are sent to the plan reuse planner (line 6). When the plan is fixed, the parallelization step is applied (line 7). Finally, the MAP algorithm runs VAL again (line 8) and if the plan is valid it is returned as the solution. The configuration of our algorithm is the same as the previous version - the single-agent planner is LAMA-UNIT-COST. The plan reuse planner is LPG-ADAPT.

8 Extending Approach to Any-Shape Robots

For the case of non-circular robot footprints (Figure 9), given the robot model is not rotation invariant, we need to discretize orientation as well. We use a world representation that is composed of multiple layers, using the partial morphological closing operation to each layer, and as such determining individually for each orientation the corresponding actuation space [28].

First, the algorithm needs images to model both the robot and its actuation capabilities. Both are parametrized by images that can be rotated and scaled to represent any robot. As input, it is also necessary to give the center of the robot and actuation in terms of their model images, and their relative position.

Here we assume a quantization of the orientation given by n_θ layers. Using the input image for the robot model, we rotate it by $2j\pi/n_\theta$, where $0 \leq j < n_\theta$, in order to build a model of the robot for each possible orientation, as shown in Figure 10.

In terms of morphological operations, we consider two structuring elements, $\mathcal{R}$ and $\mathcal{T}$, to represent the robot and actuation models respectively. After rotating them, we get $\mathcal{R}(\theta_j)$ and $\mathcal{T}(\theta_j)$, with $0 \leq \theta_j < n_\theta$.

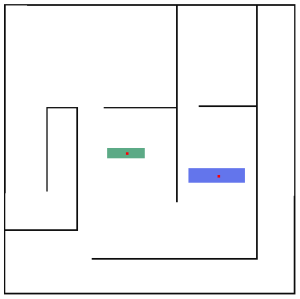


Fig. 9 Environment and robot models used to test the extended approach to any-shape robots .

Using the structuring element for the robot model, we again apply morphological operations to determine the free configuration space, now for each possible quantized orientation, dilating the map using a different robot shape model for each layer. We use a circular representation for the layered orientation, where the next layer after $\theta_j = n_\theta - 1$ is layer $\theta = 0$.

$$\mathcal{C}^{free}(\theta_j) = \{\mathbf{p} \in \mathcal{G} \mid \mathbf{p} \notin \mathcal{M} \oplus \mathcal{R}(\theta_j)\} \quad \forall 0 \leq \theta_j < n_\theta \tag{17}$$

(a) Robot Model $\mathcal{R}$ for $\theta = 0^o$

(b) Robot Model $\mathcal{R}$ for $\theta = 45^o$

(c) Robot Model $\mathcal{R}$ for $\theta = 90^o$

Fig. 10 Example of an image representing the robot footprint, rotated for three different angles, and used as structuring element in the morphological operations applied to the respective orientation layers; robot center shown in red.

In order to model a robot that navigates through waypoints, we need to establish the type of connectivity between points in different layers, such as it is equivalent to the type of motion the robot actually has. As an example, using the connectivity graph from Figure 11, where one point is connected to all its neighbors in the same layer, and the respective positions in adjacent layers, is equivalent to considering an omnidirectional model of navigation.

Given the connectivity mode, it is then possible to find all points in each layer of the configuration space that connect with the starting robot location $\mathbf{r}_i^0$, obtaining the navigable set $\mathcal{L}_i(\mathbf{r}_i^0, \theta_j)$. We can then use a second dilation operation to the navigable space in each layer to obtain the actuation space for each orientation. The structuring element for this second operation is the one that models the actuation capabilities, $\mathcal{T}$, which dilates the space according with the actuation model. If instead the structuring element $\mathcal{R}$ is used again, that would be equivalent to assuming an actuating ability completely coincident with the entire footprint.

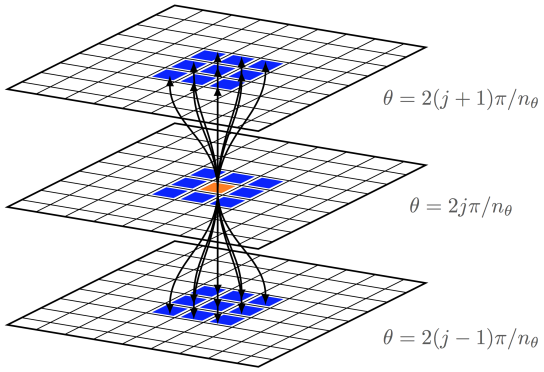
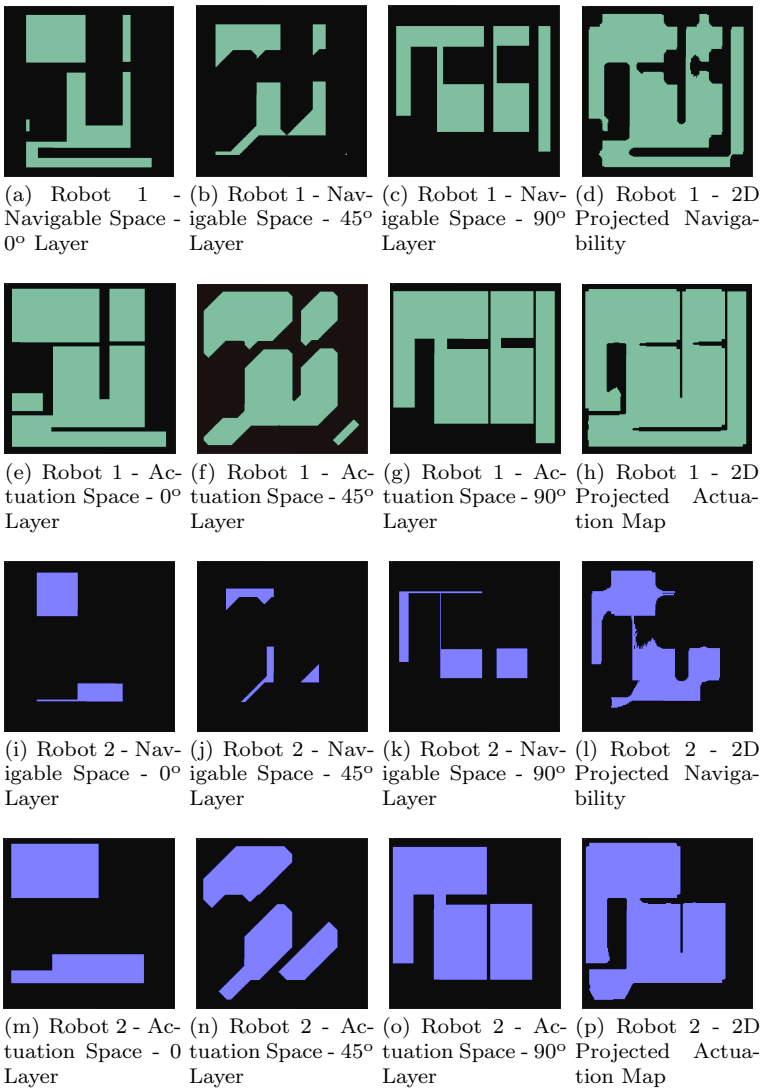


Fig. 11 Three adjancent layers of the discretized orientation, showing in blue the neighbor points of a central orange dot, representing the connectivity/motion model.

Then, the actuation space for each layer would be given by

$$\mathcal{A}(\mathbf{r}_i^0, \theta_j) = \mathcal{L}_i(\mathbf{r}_i^0, \theta_j) \oplus \mathcal{T}(\theta_j) \tag{18}$$

The actuation space gives the actuation capabilities for each orientation for a given robot shape and starting position. So, if a point belongs to $\mathcal{A}(\mathbf{r}_i^0, \theta_j)$, then it can be actuated by the robot. We show in Figure 12 the navigable and actuation spaces for different layers, given the robots and map shown in Figure 9.



(a) Robot 1 - Navigable Space - 0º Layer (b) Robot 1 - Navigable Space - 45º Layer (c) Robot 1 - Navigable Space - 90º Layer (d) Robot 1 - 2D Projected Navigability

(e) Robot 1 - Actuation Space - 0º Layer (f) Robot 1 - Actuation Space - 45º Layer (g) Robot 1 - Actuation Space - 90º Layer (h) Robot 1 - 2D Projected Actuation Map

(i) Robot 2 - Navigable Space - 0º Layer (j) Robot 2 - Navigable Space - 45º Layer (k) Robot 2 - Navigable Space - 90º Layer (l) Robot 2 - 2D Projected Navigability

(m) Robot 2 - Actuation Space - 0 Layer (n) Robot 2 - Actuation Space - 45º Layer (o) Robot 2 - Actuation Space - 90º Layer (p) Robot 2 - 2D Projected Actuation Map

Fig. 12 Navigable and Actuation Space for 2 non-circular robots with different sizes, for the scenario shown in Figure 9.

After determining the actuation space for each layer, we can obtain the overall actuation map in a rotation-invariant representation by projecting the multiple layers into one single 2D image.

$$\mathcal{P}(\mathbf{r}_i^0) = \bigcup_{\theta_j} \mathcal{A}(\mathbf{r}_i^0, \theta_j) \tag{19}$$

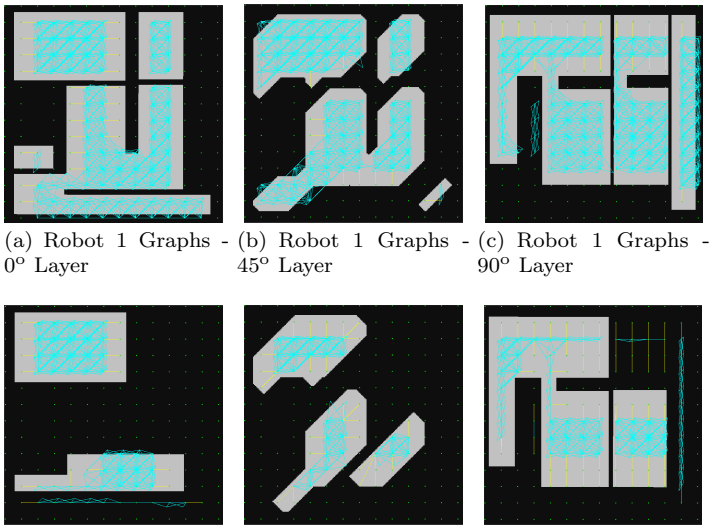
$\mathcal{P}(\mathbf{r}_i^0)$ has the same kind of representation we had with the circular robot, where the actuation space is a single 2D image not depending on the orientation.

For the any-shape robots, a multi-layer representation is used to determine the Actuation Map, representing different orientations. However, in terms of accomplishing goals, we assume it is irrelevant the orientation from which a robots actuates on a waypoint position.

Therefore, while on the rotation-invariant scenario the domain was discretized in a series of 2D waypoints, for the any-shape case there are two types of waypoints: the 3D waypoints representing (x, y, θ) position, and the 2D waypoints representing (x, y) positions invariant to orientation.

Therefore, the navigability graph now becomes a graph of 3D waypoints connected to each other, modeling the motion capabilities of robots in the world in terms of both rotation and translation, individually or combined, as exemplified for different orientation layers on Figure 13.

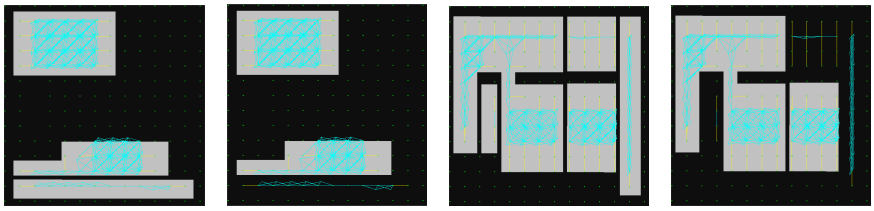
On the other hand, the actuation graph is now a graph of 3D waypoints connected to 2D waypoints, representing the actuation of a rotation-independent position in the projected 2D actuation map, from a 3D robot waypoint location,



(a) Robot 1 Graphs - 0º Layer (b) Robot 1 Graphs - 45º Layer (c) Robot 1 Graphs - 90º Layer

(d) Robot 2 Graphs - 0º Layer (e) Robot 2 Graphs - 45º Layer (f) Robot 2 Graphs - 90º Layer

Fig. 13 The `connected` and `reachable` graphs shown in blue and yellow, respectively; as shown for each layer, the yellow actuation graph connects 3D waypoints to the original 2D green waypoints, and the blue connectivity graph connects 3D waypoints not only to neighbors in the same layer, but also in adjacent layers.



(a) Robot 2 - Graphs on Free Configuration Space - 0º Layer (b) Robot 2 - Graphs on Navigable Space - 0º Layer (c) Robot 2 - Graphs on Free Configuration Space - 90º Layer (d) Robot 2 - Graphs on Navigable Space - 90º Layer

Fig. 14 The discretized graphs constructed are independent of the initial robot positions, allowing to run the problem from different initial positions; the white regions (navigable space, dependent on initial position) are covered by the graphs, but some black regions (if open in the configuration space, independent on initial position) are also covered by the constructed graphs.

also shown in Figure 13. The predicates on the PDDL problem are represented as follows:

- `Connected (robot, 3Dwaypoint, 3Dwaypoint)`
- `Reachable (robot, 3Dwaypoint, 2Dwaypoint)`

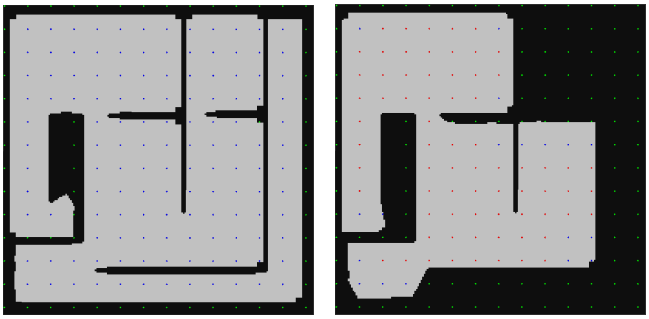
For each 2D waypoint in the circular robot scenario, there is now n_θ 3D waypoints in the same (x, y) position, representing the different orientations a robot can have on the same 2D waypoint. As we show in Figure 14, the two graphs are constructed independently of the initial position, allowing very easily to change the starting location of any robot and solve a different instance of the same problem. Thus, there were no modifications in the modeling of the PDDL problem. The 3D to 2D representation is transparent to the planning process.

The `navigate` action moves through 3D waypoints, and the `actuate` action makes 2D waypoints have the `actuated` predicate. The list of goals to solve the problem is still given by a list of 2D waypoints that cover all the space. Thus for the same map, the coverage problem is still the same in terms of goal waypoints and its modellization, but now we plan for robots to move through the environment and actuating goal positions from some planned orientation. The PDDL domain did not need any further modifications.

If we project the multiple layers of the graphs in a 2D image, we can analyze which waypoints are navigable in terms of the robot motion, and which ones are only feasible through an actuation action. As we show in Figure 15, some of the waypoints are not feasible by any of the robots, and all the feasible waypoints lie inside the Actuation Space (grey region of the images).

9 Experiments and results

In this section we show the results of the experiments that were designed test the impact of the preprocessing on two different versions of our MAP algorithm. First, on the following section we describe the five scenarios designed to run the experiments. Then, on Section 9.2 the experiments on the Coverage problem are



(a) Robot 1 (b) Robot 2

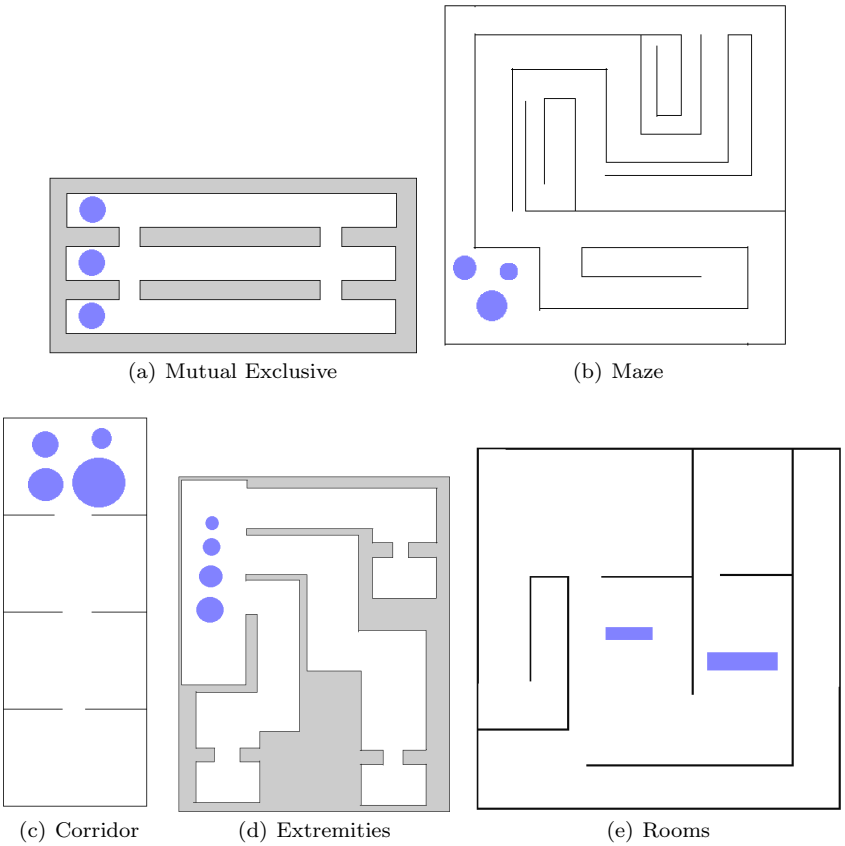
Fig. 15 All goal waypoints are shown as spheres on top of the Actuation Map: green represent unfeasible waypoints, in red the ones covered by the `connected` graph, and in blue the ones only covered by the `reachable` graph; for the smaller robot 1, the two graphs are the same.

analyzed. These results were partially included on the previous version of the paper [29]. Finally, on Section 9.3 we show the results on the Coverave problem including collision detection.

9.1 Simulation Description

Here we describe in detail the scenarios used for running the experiments. We designed three different scenarios, shown in Figure 16, each one with two levels of waypoint density (H, the higher, and L, the lower density) plus two more scenarios that only have one density level. The scenarios are designed for circular robots except for the last one (called *Rooms*) that is designed for any-shape robots. Furthermore, in Table 1 we present the size of each map image, and the number of feasible and unfeasible goals for each scenario.

- Mutual Exclusive: three wide parallel horizontal halls, connected between them by two narrow vertical halls; 3 robots move within the horizontal sections, one in each, and their actuation reachabilities are mutually exclusive.
- Corridor: four wide sections with openings of different sizes connecting them; the opening decreases from the top to the bottom, with all 4 robots being able to actuate in the top region, to only 1 being able to reach the bottom.
- Extremities: wide open section with three halls departing to different directions, where all 4 robots actuate; at the end of each hall there is a room that can be accessed through an opening, with only one robot reaching the extremity connected with the smallest opening, to three reaching the one connected with the biggest opening.
- Maze: maze-like scenario with narrow halls and passages with different sizes, resulting in bigger robots not reaching some parts of the maze, or needing to traverse bigger paths to arrive to the same locations as smaller robots.
- Rooms: simple floorplan environment with some room-like spaces connected through passages of different sizes as well, used to test the non-circular robot case where they can traverse the passages using only certain orientations.



(a) Mutual Exclusive (b) Maze

(c) Corridor (d) Extremities (e) Rooms

Fig. 16 Maps of the five scenarios used in the experiments. Grey regions represent out-of-reach regions which cannot contain goal waypoints. They are unfeasible for all the robots. Robots are represented with blue circles positioned in the region of their starting position.

Table 1 Number of feasible and unfeasible goals for all robots in each problem, and respective grid size.

	Feasible	Unfeasible	Grid Size
CorridorH	819	118	49x19
CorridorL	384	92	33x13
ExtremeH	1993	1325	51x63
ExtremeL	896	589	34x42
MutExH	499	513	45x21
MutExL	223	242	30x14
Maze	572	100	25x25
Rooms	131	61	13x13

9.2 Experiments on the Coverage problem

In this section we show some experiments that test the impact of the preprocessing on the MAP algorithm (MAP in advance). As it was previously said, we have modeled five different scenarios that include up to four agents with different sizes, and thus different actuation capabilities. Planning results are shown using as metrics the time in seconds, the length of the resulting plan and the makespan. In non-temporal domains, we refer as makespan the length of the parallel plan (number of execution steps, where several actions can be executed at the same execution step). Given that we are dealing with MAP tasks that have no interactions, it is expected that agents can execute their actions in parallel whenever possible.

Four different configurations of our MAP algorithm have been set up:

- MAP-LB-EC with estimated-cost information (EC). EC refers to the configuration that combines Actuation Maps and MAP.
- MAP-BC-EC with estimated-cost information (EC), also combining Actuation Maps and MAP.
- MAP-LB, same as before but without EC information.
- MAP-BC same as before but without EC information.

As it was mentioned in Section 5.2, the LB strategy helps to minimize the *makespan* metric. The BC strategy focuses on minimizing the *plan length* metric. We also run the problems without the preprocessing stage in order to evaluate our impact in terms of computation time and plan quality.

Furthermore, the following state-of-the-art planners have been chosen as a comparison baseline:

- LAMA [30], centralized planer and winner of IPC 2011.
- YAHSP [36], a greedy centralized planner.
- ADP [9], a multi-agent planner that automatically detects agents.
- SIW [26], a multi-agent planner that factorizes the problem into subproblems solving one atomic goal at a time until all atomic goals are achieved jointly.
- CMAP [3], a multi-agent planner that employs a centralized approach to solve the problem.

The three multi-agent planners that have been chosen participated on the 1st Competition of Distributed and Multi-agent Planners (CoDMAP[2]) and obtained good results on the final classification.

Neither of these five planner perform a goal allocation phase separated from the planning process. Thus, we had to test them using the equivalent PDDL problems that do not contain unfeasible goals. Also, in order to fairly compare the results of the makespan metric, we had to apply our parallelization algorithm to the resulting plans of ADP and SIW, as they only return the sequential plan.

We have generated two problems per scenario, one of them with less number of waypoints (which we identify as L in tables) and the other one with a high density of waypoints (H), except for the last two scenarios that only have one density level (*Maze* and *Rooms*), making it a total of eight problems. The *Rooms* scenario works for any-shape robots while the rest work for circular robots. Before discussing the results on the tables we need to clarify that a maximum of two hours was given

[2] http://agents.fel.cvut.cz/codmap/

to each planner to solve each scenario. YAHSP results do not appear in the tables because it could not solve any of the scenarios.

The maximum time spent on the preprocessing for any scenario was 170 milliseconds, for the Extremities problem with 4 robots. We included the preprocessing times (to generate the AMs) in the GA column of Table 2, and in the total time in Tables 3 and 6. Hardware used for running the planner was IntelXeon 3,4GHz QuadCore 32GB RAM. AMs were computed using a 2.5GHz DualCore 6GB RAM. Table 2 is shown to prove the remarkable impact that information from Actuation Maps (AMs) has in combination with the MAP algorithm. Goal assignment (GA) times in Table 2 are minimal (MAP-LB-EC) in comparison with the ones when MAP-LB needs to compute the relaxed plans for every goal-agent pair. Even though the individual planning time and parallelization time for MAP-LB-EC is slightly higher than MAP-LB, the time gains in GA completely dominate the overall planning time.

Table 2 Detailed time results in seconds for the MAP algorithm using the Load Balance strategy with and without estimated cost information. From left to right total time, goal assignment time, individual planning time and parallelization time.

	MAP-LB-EC				MAP-LB			
Name	TOTAL(s)	GA	Planning	Parallel	TOTAL(s)	GA	Planning	Parallel
CorridorH	33.58	0.64	24.37	8.57	1232.97	1204.20	20.88	7.89
CorridorL	6.18	0.26	4.62	1.30	128.78	123.59	4.10	1.09
ExtremH	602.68	3.06	428.28	171.34	*timeout*			
ExtremL	58.32	0.92	40.93	16.47	3870.00	3823.75	32.89	13.36
MutExH	7.39	0.34	5.03	2.02	903.65	896.82	4.81	2.02
MutExL	1.39	0.12	1.04	0.23	69.41	68.19	0.98	0.24
Maze	254.40	0.32	210.32	43.76	*timeout*			
Rooms	146.60	0.15	94.30	52.15	1620.02	1554.36	95.39	50.45

Table 3 Total time results in seconds. From left to right MAP with estimated-cost information in Load-balance (LB-EC); MAP without estimated cost information in LB; MAP with estimated cost information in Best-cost (BC-EC); MAP without estimated cost information in BC; ADP, SIW and CMAP are other multi-agent planners and LAMA is a centralized planner.

	Total Time (s)							
	MAP-LB-EC	MAP-LB	MAP-BC-EC	MAP-BC	ADP	LAMA	SIW	CMAP
CorridorH	**33.58**	1232.97	41.53	2839.79	*mem.limit*	*timeout*	96.02	*timeout*
CorridorL	6.18	128.78	9.07	135.58	104.47	**5.75**	9.42	180
ExtremH	**602.68**	*timeout*	1788.69	*timeout*	*mem.limit*	*timeout*	*timeout*	*timeout*
ExtremL	**58.32**	3870.00	112.03	3929	*mem.limit*	*timeout*	196.91	*timeout*
MutExH	7.39	903.65	7.27	910.18	5.54	6.29	**4.89**	1274.14
MutExL	1.39	69.41	1.38	72.24	**0.84**	1.12	1	95.77
Lab	**254.40**	*timeout*	308.32	*timeout*	427.08	*timeout*	320.15	*timeout*
Rooms	**146.60**	1620.02	268.44	1628.93	*mem.limit*	387.12	288.65	1328.65

The easiest scenario to be solved using planning is the Mutual Exclusive (MutExH, MutExL) because it is designed for each robot to traverse a mutual exclusive subset of waypoints. This is the reason why time results are very similar among all planners except for MAP-LB and MAP-BC where the planner needs to compute

the relaxed plans for each pair robot-goal. Regarding time results, the fastest configuration is MAP-LB-EC if all total times are summed up. Also, the impact of combining information from actuation maps with MAP can be easily appreciated if columns from MAP-LB-EC and MAP-LB are compared in Table 3. The same happens with BC configurations. ADP and LAMA were only capable of solving four problems. ADP reached the memory limit when planning the solutions before the two hours limit. Even though ADP is a multi-agent planner, the effort of computing plans when all goals are assigned to all agents is very big. LAMA reached the two hours limit (timeout) without returning a solution on the other four problems.

Table 4 shows the results regarding the plans' length and Table 5 the results regarding makespan. The best configuration overall regarding plan length is MAP-BC-EC. Regarding makespan, MAP-LB-EC is better. Moreover, MAP-LB-EC configuration is the best one for problems with higher density of waypoints, while MAP-LB proves to be better for reducing makespan in low density problems. This can be explained by the discretization errors from Equation 13, which are greater when the down sampling rate is bigger. When allocating goals, the estimation costs are the only guide for the MAP algorithm. The consequence of having slightly inaccurate cost estimates results in the allocation of some goals to different agents than the ones that the estimated costs from the relaxation of plans would suggest. However, this issue does not have a big impact on makespan and plan length results.

From the set of planners chosen to compare our approach, SIW obtains the best performance on time, plan length and makespan. SIW is able to solve most of the scenarios due to its factorization process. The importance of factorizing a MAP problem is a conclusion that can be extracted after observing Tables 4 and 5, as the planners that do not perform factorization (LAMA, ADP, CMAP, YAHSP) have to solve bigger and more complex tasks.

Regarding our configuration, MAP-BC-EC's goal-allocation works better than the one performed by MAP-BC. On the other hand, the lower the number of agents used to plan, the harder the planning task needed to solve for the ones being used. Thus, plan length is better with BC, but on the contrary total time is usually worse than LB configurations.

Table 4 Plan length: from left to right MAP with estimated-cost information in Load-balance (MAP-LB-EC); MAP without estimated cost information in LB; MAP with estimated-cost in Best-Cost (MAP-BC-EC); MAP without estimated cost information in BC; ADP, LAMA, SIW and CMAP.

	Plan Length							
	MAP-LB-EC	MAP-LB	MAP-BC-EC	MAP-BC	ADP	LAMA	SIW	CMAP
CorridorH	1289	1268	1226	**1136**			1154	
CorridorL	605	598	588	475	1403	**470**	492	**470**
ExtremH	3428		**3116**					
ExtremL	1490	1587	1365	**1233**			1398	
MutExH	**642**	**642**	**642**	**642**	748	**642**	723	**642**
MutExL	**277**	**277**	**277**	**277**	278	**277**	339	**277**
Maze	1463	1437	**1355**	1346	1553		1376	
Rooms	473	**469**	475	475	481	478	476	478

Table 5 Makespan: from left to right MAP with estimated-cost information in Load-balance (MAP-LB-EC); MAP without estimated cost information in LB; MAP with estimated-cost in Best-Cost (MAP-BC-EC); MAP without estimated cost information in BC; ADP, LAMA, SIW and CMAP.

	Makespan							
	MAP-LB-EC	MAP-LB	MAP-BC-EC	MAP-BC	ADP	LAMA	SIW	CMAP
CorridorH	403	408	461	734			**397**	
CorridorL	219	**189**	303	458	1313	286	263	286
ExtremH	**1453**		1929					
ExtremL	556	511	928	1140			**463**	
MutExH	**116**	**116**	**116**	**116**	162	**116**	187	**116**
MutExL	**59**	**59**	**59**	**59**	60	**59**	101	**59**
Maze	466	479	597	793	603		**461**	
Rooms	**240**	**240**	242	242	257	253	245	253

9.3 Experiments detecting potential collisions

In this section we show the results obtained on the same scenarios as in the previous section but using instead the PDDL domain that detects collisions described in Section 7. In Tables 6 and 7, we refer to MAP&R-LB-EC as running the Algorithm 8 using the LB strategy. MAP&R-BC-EC runs the BC strategy instead. We have also compared our approach against the same set of planners as in Section 9.2. The maximum time for each planner to solve each scenario is two hours. YAHSP results are not shown in the tables as it was not able to solve any problem. The aim of this experiment is to analyze the impact of detecting and fixing collisions on the makespan and time metrics. Plan length is not relevant, as the difficulty relies on the planner's ability to manage several agents and collision avoidance at the same time. That is a feature that directly affects the makespan result.

Regarding time results in Table 6, it can be seen that the number of problems solved decreases considerably. Also, time results have increased in all planners. This is due to the collision avoidance effect. On one hand, centralized approaches can deal with it more easily, as the master agent has the whole control of the agents. However, it is still facing the same issue as in the previous experiments: the tasks are harder to solve and now the search space is bigger. On the other hand, distributed approaches as SIW solve more scenarios and obtain better results. Again, the factorization of the planning task is relevant for the planners' performance. SIW's time results increased in comparison with those on Table 3.

Table 6 Time in seconds from left to right MAP with estimated-cost information in Load-balance (MAP&R-LB-EC); MAP with estimated-cost in Best-Cost (MAP&R-BC-EC); ADP, LAMA, SIW and CMAP.

	Total Time (s)					
	MAP&R-LB-EC	MAP&R-BC-EC	ADP	LAMA	SIW	CMAP
CorridorH	*timeout*	*timeout*	*mem.limit*	*timeout*	**298.33**	*timeout*
CorridorL	175.44	190.27	285.4	220.53	**26.77**	383.75
ExtremH	*timeout*	*timeout*	*mem.limit*	*timeout*	*timeout*	*timeout*
ExtremL	*timeout*	*timeout*	*mem.limit*	*timeout*	**557.95**	*timeout*
MutExH	**12.51**	13.05	*mem.limit*	14.06	13.49	1425.34
MutExL	**4.00**	4.27	4.10	4.3	5.16	124.30
Maze	472.89	*timeout*	*mem.limit*	*timeout*	**405.66**	*timeout*
Rooms	296.55	357.83	*mem.limit*	*timeout*	**284.66**	*timeout*

Table 7 Makespan: from left to right MAP with estimated-cost information in Load-balance (MAP&R-LB-EC); MAP with estimated-cost in Best-Cost (MAP&R-BC-EC); ADP, SIW, CMAP and LAMA.

	Makespan					
	MAP&R-LB-EC	MAP&R-BC-EC	ADP	LAMA	SIW	CMAP
CorridorH					**577**	
CorridorL	387	553	545	358	**345**	358
ExtremH						
ExtremL					**505**	
MutExH	198	198		204	**158**	204
MutExL	88	88	**60**	90	79	90
Maze	**503**				512	
Rooms	**250**	**250**			288	

This is directly related to the collision avoidance. As SIW solves one new goal on each iterated factorization, a bigger amount of robots' actions are changed due to collisions; and that requires a bigger amount of time to be fixed.

Our approaches are halfway between the centralized and the distributed approach. The first part of our algorithm is distributed while the plan-reuse phase is centralized. Thus, the success of our algorithm depends on the number of collisions and the difficulty of solving them. LPG-ADAPT focus first on reutilizing the greater possible number of the actions from the invalid plan. When a collision is detected, LPG-ADAPT will search for a valid action on the part of the search space that is closer to the invalid action and its current planning state. This causes LPG-ADAPT to iteratively explore the search space starting from a very concrete section. The exploration distance will be increased as long as the valid action is still not found. This approach works well if the collision requires a small change to be fixed i.e. it only affects to a couple of actions - the solution can be found near the search space of the action and current state. But if the way to avoid the collision affects to a bigger part of the plan i.e. robots have to move back several waypoints and change path directions; LPG-ADAPT might get lost on the search space, as it will try to search first on the space closer to the invalid action and the solution might be far away from there. Thus, timeout will be reached before a solution is found. Scenarios not solved by our approaches on Table 6 fail for that reason.

Regarding makespan results from Table 7, MAP&R-LB-EC and SIW are the two planners that obtain the best results - they also solve most of the problems. Although the *Rooms* scenario might seem easier to solve by just looking at Table 1, as it works for any-shape robots, the grid of waypoints is bigger and harder to navigate from the planning point of view. The search space is very big. The reason of failing on the *Extremities* and *Maze* scenarios is due to the changes on the robots' paths caused by the collision avoidance or the topology of the scenario. Those scenarios contain narrower areas and large halls where only some robots can reach the end. Thus, robots might spend a lot of time looking for the correct path while at the same time avoiding the rest of the agents.

10 Related Work

Multi-Agent Planning (MAP) is an active topic within the planning community as shown by the organization of the 1st CoDMAP[3] and the wide range of planners that participated [32]. The planners vary from fully distributed to centralized among other features. Our approach uses first a distributed planner and if plan reuse is needed, a centralized phase.

A MAP approach that uses a preprocessing step is the automated agent decomposition for multi-robot task planning [10]. In that work there is a preprocessing step, prior to actual search, that exploits decompositions of the problem in domains with a lower level of interaction, boosting the final performance. ADP [9] is also related to that work. Our approach factorizes the problem regarding goals and agents involved, creating independent subtasks for each agent before starting to plan.

The methodology that uses morphological operations in order to build the Actuation Maps was previously introduced by [27]. Moreover, it has recently been shown that similar transformation can be used to obtain Actuation and Visibility Maps for any-shape robots as well [28].

Earlier in 1997, Ambite and Knoblock [1] worked in a post-processing technique which rewrites some planning rules and local search techniques to make efficient the planning process. Thus, they obtained a low-cost plan for problems that were hard to solve from the point of view of planning.

Another similar problem is the Team Orienting Problem, where robots maximize the number of covered waypoints visited, with constraints on the total travel time of each robot. In order to solve this problem and optimize the overall planning time, an algorithm was proposed that uses a three-tier graph, interleaving the search for optimal waypoint assignment, ordering of the waypoints and also considering feasible paths between waypoints while avoiding obstacles [33]. This algorithm guarantees optimal solutions, while in our work we focus on sub-optimal planning for multi-robot teams.

Another relevant planning problem in robotics is inspection, which searches for robot paths that can perceive a set of sensing targets. Neural network approaches have been proposed [11], and again a preprocessing method was used, answering visibility queries efficiently both in 2D and 3D scenarios [21]. Here the queries ask for visibility from specific positions. In our work we preprocess the environment to find the overall actuation capabilities of the robot from any reachable position.

There are similar environments to our problem defined in previous planning domains. One it is called *VisitAll*, whose aim is to visit all the waypoints presented in the problem by just navigating through them. It was used for the first time on 7th IPC [4]. In our domain we added an actuation action (with an associated actuation range), which was not considered in the *VisitAll* domain. Another similar problem is the Rovers domain, but in this case each agent (rover) can execute a bigger set of actions when it is placed on a waypoint. Some other examples were mentioned in the Introduction.

When robots with many degrees of freedom execute successive motions in the same environment, it usually requires many complex planning instances. By apply-

[3] http://agents.fel.cvut.cz/codmap/

[4] http://www.plg.inf.uc3m.es/ipc2011-deterministic

ing a preprocessing technique, it is possible to improve efficiency of path planning for those robots [22]

Finally, a very common robotic application, coverage path planning, has been widely studied [16]. However, the many cell decomposition-based strategies still do not objectively consider heterogeneity and thus we do not exactly know how it creates different feasible tasks for each robot when assigning goals.

11 Conclusions and Future work

In this paper we showed how to combine information from Actuation Maps with Multi-Agent Planning to solve a multi-robot path planning problem more efficiently skipping the computation of estimated cost during planning. We used Actuation Maps in a preprocessing step to determine the feasibility of pairs robot-goal and to extract an estimated cost. That cost is used later to avoid the computation of relaxed plans during Goal-Assignment. The environment map was discretized into a grid of waypoints. The goals were distributed thanks to a goal-allocation algorithm and unfeasible goals identified and discarded from the planning task. Then, the planning task was factorized for each robot. They generate their individual paths that result in a maximal space coverage in terms of actuation. We also proposed a new version of the MAP algorithm that is able to fix agents' interactions after the individual planning phase. On the experiments we have designed a total of eight scenarios, seven for circular robots and one for any-shape robots, which is another contribution to the paper.

Our approach greatly reduces the GA time, and because GA is one of the bottlenecks of MAP, we were able to also reduce the overall planning time when preprocessed information was provided to the MAP algorithm. The gains in performance depend greatly on the topology of the environment and the characteristics of each robot.

Experiments also show that when solving big size multi-agent problems using planning, it is very helpful to first factorize the problem into subtasks. Thus, the total planning time will be smaller than when trying to solve the whole problem at once. Also, factorization is essential when working on problems that explicitly involve agents' interactions. Experiments on collision avoidance show the importance of task factorization and the topology of the scenario in order to successfully fix collisions. When problems scale and complexity increases, centralized MAP approaches perform worse than distributed approaches.

As future work, we would like to extend the preprocessing technique to other domains and consider different -robot or agent- models. Our approach can be easily extended to path planning tasks or real-time strategy videogames. We gave some examples of the former such as surveillance tasks or search and rescue tasks. The latter domain could be interesting when designing bots that play automatically. Our approach could improve the player/bot performance when extracting information from the map to decide which goals are more relevant to achieve first. We also want to improve the performance of fixing interactions. Plan reuse works well when collisions only affect to a couple of actions. For biggest plan modifications plan-reuse is not enough.

Acknowledgements This work has been partially supported by MICINN projects TIN2014-55637-C2-1-R and TIN2017-88476-C2-2-R, and also financed by the ERDF – European Regional Development Fund through the Operational Programme for Competitiveness and Internationalisation - COMPETE 2020 Programme within project ≪POCI-01-0145-FEDER-006961≫, and by National Funds through the FCT – Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) as part of project UID/EEA/50014/2013, and FCT grant SFRH/BD/52158/2013 through Carnegie Mellon Portugal Program.

References

1. Ambite JL, Knoblock CA (1997) Planning by rewriting: Efficiently generating high-quality plans.
2. Borrajo D (2013) Plan sharing for multi-agent planning. In: Nissim R, Kovacs DL, Brafman R (eds) Proceedings of the ICAPS'13 DMAP Workshop, pp 57–65
3. Borrajo D, Fernández S (2015) MAPR and CMAP. In: Proceedings of the Competition of Distributed and Multi-Agent Planners (CoDMAP-15), Jerusalem (Israel), URL http://agents.fel.cvut.cz/codmap/results/CoDMAP15-proceedings.pdf
4. Borrajo D, Fernández S (2018) Efficient approaches for multi-agent planning. KAIS 14:253–302
5. Borrajo D, Veloso MM (2012) Probabilistically reusing plans in deterministic planning. In: In Proceedings of ICAPS'12 Workshop on Heuristics and Search for Domain Independent Planning
6. Borrajo D, Roubíčková A, Serina I (2015) Progress in case-based planning. ACM Comput Surv 47(2):35:1–35:39, DOI 10.1145/2674024, URL http://doi.acm.org/10.1145/2674024
7. Brafman RI, Domshlak C (2013) On the complexity of planning for agent teams and its implications for single agent planning. Artificial Intelligence 198:52–71, DOI http://dx.doi.org/10.1016/j.artint.2012.08.005
8. Conitzer V (2010) Comparing multiagent systems research in combinatorial auctions and voting. AMAI 58(3-4):239–259, DOI 10.1007/s10472-010-9205-y
9. Crosby M (2015) Adp an agent decomposition planner. Proceedings of the CoDMAP-15 p 4
10. Crosby M, Rovatsos M, Petrick RP (2013) Automated agent decomposition for classical planning. In: ICAPS
11. Faigl J (2010) Approximate solution of the multiple watchman routes problem with restricted visibility range. IEEE transactions on neural networks 21(10):1668–79
12. Fikes RE, Nilsson NJ (1971) Strips: A new approach to the application of theorem proving to problem solving. In: Proceedings of IJCAI, pp 608–620, URL http://dl.acm.org/citation.cfm?id=1622876.1622939
13. Foulser D, Li M, Yang Q (1992) Theory and algorithms for plan merging. Artificial Intelligence 57(2-3):143–181
14. Fox M, Long D (2003) PDDL2.1: An extension to PDDL for expressing temporal planning domains. JAIR 20
15. Fox M, Gerevini A, Long D, Serina I (2006) Plan stability: Replanning versus plan repair. In: Proceedings of ICAPS'06, pp 212–221

16. Galceran E, Carreras M (2013) A survey on coverage path planning for robotics. Robotics and Autonomous Systems 61(12):1258–1276
17. García-Olaya A, de la Rosa T, Borrajo D (2011) Using the relaxed plan heuristic to select goals in oversubscription planning problems. In: Lozano JA, Gámez JA, Moreno JA (eds) Advances in Artificial Intelligence, pp 183–192
18. Gerkey BP, Matarić M (2004) A formal analysis and taxonomy of task allocation in multi-robot systems. International Journal of Robotics Research 23(9):939–954
19. Hoffmann J, Nebel B (2001) The FF planning system: Fast plan generation through heuristic search. JAIR 14:253–302
20. Howey R, Long D, Fox M (2004) VAL: Automatic plan validation, continuous effects and mixed initiative planning using PDDL. In: ICTAI 2004, pp 294–301
21. Janousek P, Faigl J (2013) Speeding up coverage queries in 3D multi-goal path planning. ICRA (1):5082–5087, DOI 10.1109/ICRA.2013.6631303
22. Kavraki L, Latombe JC (1994) Randomized preprocessing of configuration for fast path planning. In: Proceedings of the International Conference on Robotics and Automation, IEEE, pp 2138–2145
23. Krulwich B (1992) Planning for soft goals. In: Hendler J (ed) Artificial Intelligence Planning Systems, pp 289 – 290, DOI https://doi.org/10.1016/B978-0-08-049944-4.50047-1, URL https://www.sciencedirect.com/science/article/pii/B9780080499444500471
24. Luis N, Borrajo D (2014) Plan Merging by Reuse for Multi-Agent Planning. In: Proceedings of the 2nd ICAPS Distributed and Multi-Agent Planning workshop (DMAP)
25. Mali AD (2000) Plan merging & plan reuse as satisfiability. In: Proceedings of the 5th European Conference on Planning: Recent Advances in AI Planning. ECP'99, pp 84–96
26. Muise C, Lipovetzky N, Ramirez M (2015) MAP-LAPKT: Omnipotent multi-agent planning via compilation to classical planning. In: Competition of Distributed and Multiagent Planners, URL http://www.haz.ca/papers/muise_CoDMAP15.pdf
27. Pereira T, Veloso M, Moreira A (2015) Multi-robot planning using robot-dependent reachability maps. In: Robot 2015: Second Iberian Robotics Conference, Springer, pp 189–201
28. Pereira T, Veloso M, Moreira AP (2016) Visibility maps for any-shape robots. In: IROS'16, the IEEE/RSJ International Conference on Intelligent Robots and Systems, vol 29, pp 428–459
29. Pereira T, Luis N, Moreira A, Borrajo D, Veloso M, Fernandez S (2018) Heterogeneous multi-agent planning using actuation maps. In: 2018 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC), pp 219–224, DOI 10.1109/ICARSC.2018.8374186
30. Richter S, Westphal M (2010) The LAMA planner: Guiding cost-based anytime planning with landmarks. JAIR 39:127–177
31. Smith DE (2004) Choosing objectives in over-subscription planning. In: ICAPS
32. Štolba M, Komenda A, Kovacs DL (eds) (2015) Proceedings of the Competition of Distributed and Multi-Agent Planners (CoDMAP-15)
33. Thakur D, Likhachev M, Keller J, Kumar V, Dobrokhodov V, Jones K, Wurz J, Kaminer I (2013) Planning for opportunistic surveillance with multiple

robots. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, pp 5750–5757
34. Torreño A, Onaindia E, Komenda A, Štolba M (2017) Cooperative multi-agent planning: A survey. ACM Comput Surv 50(6):84:1–84:32, DOI 10.1145/3128584, URL http://doi.acm.org/10.1145/3128584
35. Veloso MM, Pérez MA, Carbonell JG (1990) Nonlinear planning with parallel resource allocation. In: Proceedings of the DARPA Workshop on Innovative Approaches to Planning, Scheduling, and Control, Morgan Kaufmann, pp 207–212
36. Vidal V (2014) YAHSP3 and YAHSP3-MT in the 8th International Planning Competition. In: In 8th International Planning Competition (IPC-2014), pp 64—-65

Authors' biography

Nerea Luis is a PhD student at the Universidad Carlos III de Madrid. Her thesis is about combining multi-agent planning and plan-reuse to easily adapt to different scenarios. In 2016, she was a visitor student at Manuela Veloso's CORAL group for 4 months in Carnegie Mellon University (USA). Her current research interests are multi-agent planning, machine learning and robotics. She was awarded with the Women Techmakers scholarship by Google.

Tiago Pereira is a Ph.D. candidate in the CMU-Portugal Program, being a student in the Electrical and Computer Engineering Department both at Carnegie Mellon University and Faculty of Engineering of University of Porto. He is also a research assistant at INESC TEC. He received a B.S and M.S. in Electrical and Computer Engineering from University of Porto in 2012. His research interests include motion planning and coordination of heterogeneous multi-robot systems.

Susana Fernández is currently an Associate Professor at the Universidad Carlos III de Madrid, since 2007. She received the B.A. degree in Physics (Universidad Complutense de Madrid, 1989), and a M.Phil. degree in "Logic, Text and Information Technology" (University of Dundee, Scotland, 1991). She worked for 10 years as a software engineer in two private companies, INDRA and Eliop, S.A. She received the Ph.D. degree in Computer Science (Universidad Carlos III de Madrid, 2006). Her current research interest is in Artificial Intelligence, particularly automated planning, machine learning and multi-agent systems. She has participated in several competitive projects related to automated planning, problem solving or agents. She has published more than 20 papers.

António Moreira is currently an Associated Professor in Electrical Engineering, developing his research within the Robotic and Intelligent Systems Centre of INESC TEC (Centre Coordinator). He graduated with a degree in Electrical Engineering from the University of Porto in 1986. He then pursued graduate studies at the University of Porto, completing a M.Sc. degree in Electrical Engineering - Systems in 1991 and a Ph.D. degree in Electrical Engineering in 1998. From 1986 to 1998 he also worked as an assistant lecturer in the Electrical Engineering Department of the University of Porto. His main research areas are Process Control and Robotics.

Daniel Borrajo received his Ph.D. from Universidad Politécnica de Madrid (1990). Currently, he is a Professor of Computer Science at Universidad Carlos III de Madrid. He has published over 200 journal and conference papers, mainly in the fields of problem solving methods (heuristic search and automated planning) and machine learning. He has been the Program Co-chair of the International Conference of Automated Planning and Scheduling (ICAPS'13), Conference co-chair of the Symposium of Combinatorial Search (SoCS'12, SoCS'11) and ICAPS'06, and PC member (including Area chair and Senior PC) of the main conferences on Artificial Intelligence (e.g., IJCAI, AAAI, ICAPS, ...). His current research interests lie in goal reasoning, multiagent planning, and machine learning applied to planning.

Manuela Veloso is the Herbert A. Simon University Professor in the School of Computer Science at Carnegie Mellon University. She is the Head of the Machine Learning Department. She researches in Artificial Intelligence, Robotics, and Machine Learning. She founded and directs the CORAL research laboratory, for the study of autonomous agents that Collaborate, Observe, Reason, Act, and Learn, www.cs.cmu.edu/~coral. Professor Veloso is AAAI, Fellow, ACM Fellow, AAAS Fellow, and IEEE Fellow, and the past President of AAAI and of RoboCup. Professor Veloso and her students research a variety of autonomous robots, including mobile service robots and soccer robots. See www.cs.cmu.edu/~mmv for further information, including publications.