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

Planning Using Hierarchical Constrained Markov Decision Processes

Seyedshams Feyzabadi · Stefano Carpin

the date of receipt and acceptance should be inserted later

Abstract Constrained Markov Decision Processes offer a principled method to determine policies for sequential stochastic decision problems where multiple costs are concurrently considered. Although they could be very valuable in numerous robotic applications, to date their use has been quite limited. Among the reasons for their limited adoption is their computational complexity, since policy computation requires the solution of constrained linear programs with an extremely large number of variables. To overcome this limitation, we propose a hierarchical method to solve large problem instances. States are clustered into macro states and the parameters defining the dynamic behavior and the costs of the clustered model are determined using a Monte Carlo approach. We show that the algorithm we propose to create clustered states maintains valuable properties of the original model, like the existence of a solution for the problem. Our algorithm is validated in various planning problems in simulation and on a mobile robot platform, and we experimentally show that the clustered approach significantly outperforms the non-hierarchical solution while experiencing only moderate losses in terms of objective functions.

1 Introduction

Markov Decision Processes (MDPs) are extensively used to determine robot control policies in situations where the state is observable. This method is appealing because it builds on a formal framework guaranteeing optimality in terms of expected cumulated costs while accounting for uncertain action outcomes. Among its limitations are the assumption that the state is observable and that a single cost function is considered when determining the optimal solution. When the state is not observable, partially observable MDPs (POMDP) can be used, and there exists a vast literature in this area.

Seyedshams Feyzabadi · Stefano Carpin
School of Engineering
University of California, Merced
5200 North Lake Rd.
Merced, CA, 95343
Tel.: +1-209-228-4152
E-mail: {sfeyzabadi,scarpin}@ucmerced.edu

In this paper we focus on the other problem, i.e., the situation where multiple costs should be simultaneously considered. As robots become more capable, they are tasked with missions of increasing complexity. Consequently, more and more often control policies considering multiple costs at once are needed. For example, one would like to simultaneously consider the time to complete a mission, consumed energy, undertaken risk, and so on. When multiple costs are given, constrained MDPs (CMDPs) can be used. With CMDPs one determines a policy optimizing with respect to one cost while satisfying bounds on the other costs. This approach guarantees an optimal solution, but the main drawback is that in many practical scenarios it may require to solve constrained linear programs with an extremely large number of variables, thus creating a computational bottleneck. The computational bottleneck entails both time complexity, as larger linear programs take more time to be solved, but also space complexity, since non-hierarchical solutions may generate problem instances requiring extremely large amounts of memory (specific examples will be given in Section 5). This is particularly problematic if a problem must be repeatedly solved to tackle dynamic changes in the environment. To overcome this problem, in this paper we present a hierarchical approach to solve large CMDPs (HCMDP), i.e., CMDPs requiring the solution of linear programs with a large number of variables. The idea is to partition the CMDP's state space into clusters of states to obtain a problem instance with a much smaller number of states. An optimal policy is computed for this reduced CMDP, and then used to determine a policy for the original CMDP. As we discuss in Section 2, similar ideas have been already explored for MDPs, but to the best of our knowledge their application to CMDPs is novel, and former methods include some limitations that we overcome. In particular we show that our solution provides guarantees in terms of preserving some properties of the initial CMDP, and in particular it ensures that the hierarchical method provides a solution whenever the original problem admits one, albeit optimality is in general not preserved. A problem arising when utilizing clustering techniques is how to define pertinent parameters for the newly created clustered states. Our solution relies on a Monte Carlo approach that can be broadly applied without imposing special requirements on the structure of the underlying state space or cost functions. To assess the impact of the technique we propose, the method is extensively validated both in simulation and on a mobile robot traveling more than 5.5km in an indoor environment, and we show that substantial performance improvements are obtained while experiencing only modest decreases in therm of solution quality.

The remainder of the paper is organized as follows. Section 2 discusses related work in the area of hierarchical planners. Background in MDPs and CMDPs is provided in section 3. Our method is presented in section 4, where we describe how an HCMDP is created, and we discuss the associated planner. Section 5 illustrates the performance of the planner in three different scenarios, namely matlab simulations, Gazebo simulations, and on a P3AT robot. Finally conclusions are given in section 6. Moreover, in the appendix we provide the proofs of two theorems stated in section 4.

2 Related work

MDPs have been extensively used to solve various types of decision making problems in robotics and beyond. We refer the reader to [5, 27] for general introductions to the topic, and to [22, 23, 28] for applications specific to robotics. For a comprehensive introduction to CMDPs, Altman's book is a standard reference [1]. Although MDPs have become

one of the standard tools used in numerous robotic applications, the use of CMDPs in robotics remains very limited. In fact, when it is necessary to simultaneously consider multiple costs, a common approach is to combine them into a single objective function, for example as a linear combination [25]. This method however is unappealing because the resulting objective function is an artificial measure not related to the physical quantities experienced by the robot while operating in its environment. Similarly, if one considers the approach of solving the constrained optimization problem using a Lagrangian relaxation, a key issue is selecting appropriate constants in the relaxation. Either case shows that optimizing putting a bound on the costs is a more intuitive method. Ding et al. have been among the few to use CMDPs to solve various planning problems in robotics [13, 14]. El Chamie and Açikmeşe use a CMDP formulation based on linear programming to control a swarm of robots while providing bounds on the costs [8]. Similarly, Boussard and Miura use a CMDP based method to solve a search problem where the objective is to locate as many items as possible under a time constraint. Recently, Carpin et al. used CMDPs to solve a rapid deployment problem whereby robots have to maximize the probability to reach a set of given locations while being subject to a given temporal deadline [7, 9].

Aiming at improving computational efficiency, numerous methods proposed hierarchical solutions for MDPs, sometimes with the objective of computing reusable policies that can be repeatedly utilized for multiple similar problem instances. Dai et al. proposed various methods in this area. Topological value iteration guarantees to find an optimal solution for the underlying MDP, but the method becomes inefficient if the state space features poor connectivity [10]. Aiming at overcoming some of these limitations, focused topological value iteration builds an accurate value function on regions of the state space by taking advantage of knowing the initial state. However, it still suffers when poor connectivity exists [11, 12]. A technique called "region-based decomposition" proposed by Hauskrecht et al. [18] decomposes the overall state space into several regions resembling the structure of the environment in which the robot moves, like rooms, corridors, and the like. The key of the method is in identifying entrances to these regions to reduce the size of the state space.

Barry et al. also presented more than one solution to the hierarchical MDP planning problem. A first method based on the strongest connected components was given in [3] and then improved by the DetH* algorithm [4]. The main limitation of these solutions, however, is that they are applicable only to solve factored MDPs.

The SPUDD method proposed by Hoey et al. aims at solving very large MDPs by saving value functions and policies as functions rather than using lookup tables [19]. The MAXQ method proposed by Diettrich also aims at solving MDPs with large state spaces, but it relies on human input in the clustering stage. Hierarchical methods have also been explored in the area of POMDPs. For example Pineau et al. [26] developed a method focusing on actions, and proposed a decomposition based on the feasibility of actions in different situation.

Other methods dealing with solutions for hierarchical models were presented in [2, 6, 26, 29]. All these methods, however, do not tackle the CMDP case.

To the best of our knowledge, the first paper proposing a hierarchical solution for CMDPs has been our previous work presented in [15]. The proposed method showed that significant performance gains could be obtained with this strategy while incurring in very limited losses in terms of the cost functions. However, the partitioning method we used in our original work relied on a preassigned clustering strategy that in general does not preserve feasibility when going from the original problem to its hierarchical

version. The clustering method we propose in this paper overcomes this limitation, and extends our former preliminary contributions presented in [15, 16].

3 Background on Constrained Markov Decision Processes

In this section we introduce the concepts and notation needed to formalize the problem we tackle in this paper. The reader is referred to [5, 27] for a thorough description of MDPs, and to [1] for CMDPs.

3.1 Markov Decision Processes

A finite MDP is defined by a quadruple $\mathcal{M} = (\mathcal{X}, U, P, c)$ where:

- $\mathcal{X}$ is a finite state space with n elements.
- for each $x \in \mathcal{X}$ the finite set $U(x)$ is the set of actions that can be executed in state x. From these n sets we define $U = \cup_{x \in \mathcal{X}} U(x)$ as the set of all actions. Furthermore, from $\mathcal{X}$ and U we define the state/action set $K = \{(x, u) \mid x \in \mathcal{X}, u \in U(x)\}$.
- $P : K \times \mathcal{X} \to [0, 1]$ is a probability mass function called *transition probability*. $P(x, u, y)$ is the probability of transitioning from state x to state y when executing action $u \in U(x)$. In the following we write this as P^u_{xy} for brevity. Note that since P is a probability mass function it satisfies the probability axioms.
- $c : K \to \mathbb{R}_{\geq 0}$ is a non-negative cost function. $c(x, u)$ is the cost incurred when executing action u while being at state x.

A deterministic policy is a function $\pi : \mathcal{X} \to U$ associating to each state x an action in $U(x)$. Note that according to this definition the policy is Markovian, i.e., it depends on the current state only, but not on past history. In the following we will exclusively consider Markovian policies since it is well known that they are optimal for the problems we consider. A finite MDP $\mathcal{M}$ and a policy π induce a stochastic process over the set of states $\mathcal{X}$ [5]. In the following we use the symbol X_i for the random variable representing the state at time i obtained starting from an initial state $x_0 \in \mathcal{X}$ and repeatedly applying π. It is well known that for the most commonly used cost criteria (e.g., finite horizon, discounted infinite horizon) deterministic policies are optimal [5]. In this paper we focus on the *infinite horizon total cost* function defined as

$$c(x, \pi) = \mathbb{E}\left[\sum_{t=0}^{\infty} c(X_t, \pi(X_t)) | X_0 = x\right] \tag{1}$$

where the expectation is taken with respect to the probability distribution over the set of realizations of the stochastic process X_i induced by π. Equation (1) defines the so-called cost-to-go, i.e., it is the expected cost incurred when starting from state $X_0 = x$ and following policy π. In the following we write $c(\pi)$ for the n-dimensional vector storing $c(x, \pi)$ for each state (also known as *value vector*). Our focus on the infinite horizon total cost stems from the observation that many robotic tasks have a duration that is a priori unknown and are not necessarily subject to a discounting factor. It is evident from Eq. (1) that without additional hypotheses the total cost could in general be infinite. One alternative that we followed in our past work [9, 16] is to require that the MDP is absorbing. In this paper we instead require a set of conditions that are

easier to verify. The state space $\mathcal{X}$ can be partitioned into two subsets $\mathcal{X}'$ and M such that:

1. $\forall x \in M \ : \ c(x,u) = 0$;
2. $P^u_{xy} = 0$ for each $x \in M$, $u \in U(x)$, and $y \in \mathcal{X}'$;
3. for each $x \in \mathcal{X}$ there exists one state $y \in M$ and a policy π such that in the Markov chain associated with policy π state y is accessible from x.

The first two conditions establish that no more cost is accrued in M and that once the state enters M it remains there. The last condition implies the existence of a deterministic Markovian policy leading to M. Collectively, these conditions establish that there exists at least one Markovian policy π' such that $c(\pi')$ is finite. This guarantees that the infinite horizon total cost problem we define in the following admits a solution. Note that while in general M may consist of more than one state, since we assumed $c(x,u) = 0$ for each state in M, one can without loss of generality assume that M features a single state. This simplifies the formulation of the last condition, where accessibility is then requested from each state in $\mathcal{X}'$ to the only state in M. In the remainder of the paper we will assume that these three conditions hold. When this is the case, solving an infinite horizon total cost MDP requires to determine the policy minimizing the cost, i.e., to determine

$$\pi^* = \arg\min_{\pi} c(\pi)$$

where the minimum as intended componentwise for each state $x \in X$. As for other cost criteria, the optimal policy for this problem is a deterministic policy [5].

3.2 Constrained Markov Decision Processes

In an MDP a single cost $c(x,u)$ is incurred every time an action is executed. When multiple costs are defined, a CMDP approach can instead be used. In CMDPs one determines a policy minimizing one cost function while putting constraints on the others. Formally, a finite CMDP $\mathcal{C}$ is defined as $\mathcal{C} = (\mathcal{X}, U, P, c, d_i, D_i, \beta)$ where $\mathcal{X}, U, P, c$ are defined as for MDPs and:

- $d_i : K \to \mathbb{R}_{\geq 0}$ with $1 \leq i \leq k$ are k additional cost functions;
- D_i are k non negative bounds;
- β is a probability mass function defined over $\mathcal{X}$ giving the probability distribution for the random variable X_0, i.e., the initial state.

In a CMDP when action u is executed in state x, then each of the costs $c(x,u)$, $d_1(x,u), \ldots, d_k(x,u)$ is incurred. For each of them different cost criteria could be defined. Extending the framework we just introduced, we will consider infinite horizon total costs for all these functions. As for the MDP case, it will be necessary to introduce conditions to guarantee the existence of a solution. These are the same we considered for the MDPs, but the first condition is extended requiring that all costs are 0 in M (i.e., $d_i(x,a) = 0$ for each $1 \leq i \leq k$ and each $x \in M$.) For an absorbing CMDP and policy π the following $k+1$ total cost vectors are then defined:

$$c(\pi,\beta) = \mathbb{E}\left[\sum_{t=0}^{\infty} c(X_t, \pi(X_t))\right] \qquad d_i(\pi,\beta) = \mathbb{E}\left[\sum_{t=0}^{\infty} d_i(X_t, \pi(X_t))\right] \tag{2}$$

where as for the MDP case $c(\pi, \beta)$ is a vector storing the expected cost-to-go for each state $x \in X$. With a slight abuse of notation $d(\pi, \beta)$ is instead a scalar value obtained taking the expectation of the vector with respect to β.

In Eq.(2) expectations are taken with respect to both the policy π and the initial distribution β. This additional parameter is needed because the optimal policy for a CMDP in general depends on the initial distribution of states. The CMDP problem asks to determine a policy π^* solving the following constrained optimization problem:

$$\begin{aligned} &\pi^* = \arg\min_{\pi} c(\pi, \beta) \\ &\text{s.t. } d_i(\pi, \beta) \leq D_i \quad 1 \leq i \leq k. \end{aligned} \tag{3}$$

Given that in a CMDP there are $k+1$ costs, in the problem formulation there is a certain degree of subjectivity in deciding which cost should be minimized and which ones should instead be bounded in expectation. From a mathematical standpoint, any choice is technically correct, but from a practical perspective these choices are not equivalent and domain specific knowledge is often essential to make this choice. In some instances there is a natural hierarchy among the importance of the costs, or bounds may be simpler to derive for certain costs. For example, if one cost measures safety and the other energy consumption, it may be natural to optimize with respect to safety and to put a bound on energy consumption based on the batteries used by the robot. To corroborate this aspect, examples will be discussed in Section 5.

Although MDPs and CMDPs share many traits in their definitions, some important differences emerge when computing the optimal policies. First, the optimal policy for a CMDP is in general a stochastic policy, whereas optimal policies for MDPs are deterministic. Moreover, because the costs in Eq. (2) depend on the initial distribution β, the optimal policy depends on it, whereas the optimal policy for an MDP is independent from the initial state distribution.

The optimal policy for a CMDP is determined by solving a constrained linear program based on so-called occupancy measures. Let $K' = \{(x, u) \mid x \in \mathcal{X}', u \in U(x)\}$ be the state-action space restricted to the non absorbing states, and let $\rho(x, u)$ be a set of optimization variables associated to each element of K'. The optimization problem in Eq. (3) has a solution if and only if the following constrained linear program is feasible:

$$\begin{aligned} \min_{\rho} \quad & \sum_{(x,u)\in K'} \rho(x,u) c(x,u) \\ \text{s.t.} \quad & \sum_{(x,u)\in K'} \rho(x,u) d_i(x,u) \leq D_i \quad 1 \leq i \leq k \\ & \sum_{(y,u)\in K'} \rho(y,u)(\delta_x(y) - P^u_{yx}) = \beta(x) \quad \forall x \in \mathcal{X}' \\ & \rho(x,u) \geq 0 \quad \forall (x,u) \in K' \end{aligned} \tag{4}$$

where $\delta_x(y) = 1$ when $x = y$ and 0 otherwise. If the linear program is feasible, then the optimal solution $\rho(x, u)$ induces an optimal randomized policy π^* defined as

$$\pi^*(x,u) = \frac{\rho(x,u)}{\sum_{u\in U(x)} \rho(x,u)} \quad x \in \mathcal{X}', u \in U(x) \tag{5}$$

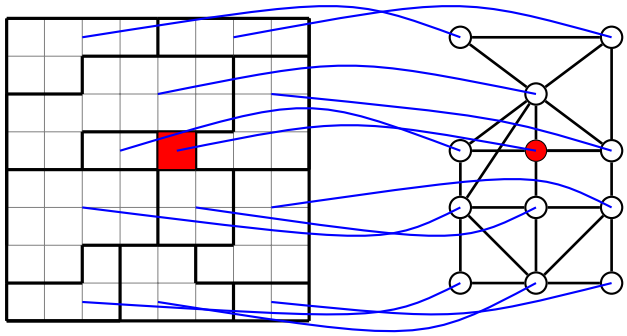

Fig. 1 The figure illustrates the idea behind the HCMDP approach. The left side shows a CMDP whose numerous states are represented by the small grid cells. On top of these states, irregularly shaped clusters are defined, as indicated by the thick line partitions. Clusters define a new HCMDP shown in the right panel. Neighboring relationships between clusters define connectivity in the HCMDP. The M states in the original CMDP (red grid cell) are mapped into M states in the hierarchical CMDP (red circle). Edges in the clustered graph are added between macro states sharing a border in the original state space.

where $\pi^*(x, u)$ is the probability of executing action u while in state x (see [1] for the proof.). The reader will note that this definition allows for randomized policies, as we formerly stated. The policy is not defined for states M, but this is irrelevant because no more costs are accrued when the state is in M, and the state cannot leave it once it enters.

While the CMDP formalism allows to tackle many different practical problems, one of the main limitations is given by the number of variables that may emerge while solving the associated linear program. In fact, one may easily end up with problem instances with tens of thousands of variables. In a fully static scenario this could sometimes be acceptable as the policy could be computed off-line. However, when the environment changes one may be required to recompute the policy online, and the computational requirements could become difficult to accommodate within a tightly timed control loop. In an effort to address these challenges, in the following we present a hierarchical method to expedite the solution while preserving feasibility.

4 HCMDP – Hierarchical Constrained Markov Decision Processes

Approximated methods based on hierarchical approaches have been proposed in literature for MDPs (see e.g., [5], vol II, Ch. 6), but to the best of our knowledge this paradigm has not been considered for CMDPs, with the exception of our recent works [15, 16]. The underlying idea for CMDPs is similar to the one used for MDPs, although there are some complications due to the necessity to consider multiple costs at once. By clustering the state space of the original CMDP we create an HCMDP with fewer states, compute a policy for the smaller instance, and then utilize this policy to derive a policy for the original CMDP (see Figure 1).

These ideas can be formalized as follows. Let $\mathcal{C} = (\mathcal{X}, U, P, c, d_i, D_i, \beta)$ be a CMDP. From $\mathcal{C}$ we extract an HCMDP $\mathcal{C}_H = (\mathcal{X}_H, U_H, P_H, c_H, d_{i,H}, D_{i,H}, \beta_H)$ with $|\mathcal{X}_H| << |\mathcal{X}|$. Next, we compute the optimal policy π^*_H for $\mathcal{C}_H$ and we use it to extract a policy π' for $\mathcal{C}$. While π^*_H is optimal for $\mathcal{C}_H$, in general π' will not be optimal $\mathcal{C}$ and in fact

one can anticipate that the gap between π' and π^* (the optimal policy for $\mathcal{C}$) widens as the size of the clusters shrinks, and viceversa narrows as the size of the clusters increases. However, our tenet is that the loss in optimality is limited and compensated by the significant gain in computational efficiency. Moreover, the hierarchical solution is also less memory intensive, i.e., by solving multiple smaller problems it requires less memory than the original non-hierarchical problem. Various subproblems must be tackled to build $\mathcal{C}_H$ from $\mathcal{C}$. In the following subsections we identify all of them and provide pertinent solutions.

4.1 Clustering

The first problem is how to compute $\mathcal{X}_H$ from $\mathcal{X}$. Elements of $\mathcal{X}_H$ will be in the following called *macro states* or *clusters*. There are evidently multiple ways to determine these clusters. The method we propose is based upon both an intuition and a formal requirement. The intuition is that clusters should be composed of *similar* states, where similarity is measured in terms of the cost function c. The formal requirement is that clusters must preserve the connectivity property we define in the following.

Definition 1 Let $\mathcal{C} = (\mathcal{X}, U, P, c, d_i, D_i, \beta)$ be a CMDP, and x, y be two states in $\mathcal{X}$. We say that x is connected to y and we write $x \rightsquigarrow y$ if there exists a sequence of states $s_1, \ldots, s_m \in \mathcal{X}$ and actions $u_1, \ldots u_{m-1} \in U$ such that $s_1 = x$, $s_m = y$, and $P^{u_i}_{s_i, s_{i+1}} > 0$ for $1 \leq i \leq m-1$.

Based on this definition we introduce the concept of connectivity preserving HCMDP.

Definition 2 Let $\mathcal{C} = (\mathcal{X}, U, P, c, d_i, D_i, \beta)$ be a CMDP and let $\mathcal{C}_H = (\mathcal{X}_H, U_H, P_H, c_H, d_{i,H}, D_{i,H}, \beta_H)$ be an HCMDP for $\mathcal{C}$. We say that $\mathcal{C}_H$ preserves the connectivity of $\mathcal{C}$ if the two following conditions hold:

1. $\mathcal{X}_H$ is a partition of $\mathcal{X}$;
2. For $z \in \mathcal{X}'$ and $y \in M$, let Z_H and Y_H be the states in $\mathcal{X}_H$ such that $z \in Z_H$ and $y \in Y_H$. If $z \rightsquigarrow y$, then $Z_H \rightsquigarrow Y_H$.

Note that because of the requirement that $\mathcal{X}_H$ is a partition of $\mathcal{X}$, the states Z_H and Y_H are unique and the second condition is then well posed. Moreover, observe that the connectivity preserving property is a function of $\mathcal{C}_H$ as a whole and not just $\mathcal{X}_H$, because connectivity depends not only on states, but also on transition probabilities. To define the clustering algorithm, for $S \subset \mathcal{X}$, $x \in \mathcal{X}$, and $u \in U(x)$ it is convenient to define the following operators:

$$\begin{aligned}
&\mathrm{Pre}(S) = \{x \in \mathcal{X} \mid \exists u \in U(x) \wedge \exists y \in S \wedge P^u_{xy} > 0\} \\
&\mathrm{Post}(x, u) = \{Y \in \mathcal{X}_H \mid \exists y \in Y \wedge P^u_{xy} > 0\} \\
&\mathrm{Post}(S) = \{Y \in \mathcal{X}_H \mid \exists y \in Y \wedge \exists x \in S \wedge P^u_{xy} > 0\}.
\end{aligned}$$

Note that $\mathrm{Pre}(S)$ is a subset of states of $\mathcal{X}$, whereas $\mathrm{Post}(x, u)$ and $\mathrm{Post}(S)$ are sets of macrostates of $\mathcal{X}_H$. Starting from these operators, Algorithm 1 shows our proposed approach to clustering. The method depends on two parameters, namely the maximum cluster size MS and a constant δ used to define whether two states have similar cost. A single cluster is initially created with the set M (line 1). Then the algorithm loops until all states in $\mathcal{X}$ have been assigned to a state in $\mathcal{X}_H$ (loop starting at line 2). At

each iteration, each of the existing clusters is examined (line 3) and the states that can reach one of the clusters in one step with non-zero probability are identified (set S at line 4). For each element in S (loop starting at line 5) every possible action is evaluated (line 6) to determine to which macrostate the action would lead to (line 7). The state is assigned to a cluster H (line 10) only if two conditions are simultaneously verified, i.e., cluster H includes less than MS states, and the average cost of cluster $C(H)$ is similar to the cost $c(s, u_s)$ incurred to move from s into H (test at line 9). As soon as a state is assigned to a cluster, it is not considered anymore (line 11). If state s cannot be assigned to any cluster (line 12), then a new cluster including just s is created (line 13) and added to the set of clusters (line 14). At the end, an optional merging step described in the following is performed (line 15) and the set of clusters is returned (line 16.)

Algorithm 1: Clustering Algorithm

```
   Data: 𝒳, U, P
   Result: 𝒳_H: Set of macro-states
 1 𝒳_H ← {M} ;
 2 while There exist states not assigned to any cluster do
 3 |   foreach C ∈ 𝒳_H do
 4 |   |   S ← Pre(C) ∩(𝒳 \ ∪𝒳_H) ;
 5 |   |   foreach s ∈ S do
 6 |   |   |   for u_s ∈ U(s) do
 7 |   |   |   |   Y_s ← Post(s, u_s);
 8 |   |   |   |   for H ∈ Y_s do
 9 |   |   |   |   |   if c(H) ≈ c(s, u_s) ∧ |H| < MS then
10 |   |   |   |   |   |   assign s to cluster H;
11 |   |   |   |   |   |   break out of the two inner for loops;
12 |   |   |   if s not assigned yet then
13 |   |   |   |   create new cluster M_s = {s} and assign s to it;
14 |   |   |   |   add M_s to 𝒳_H;
15 𝒳_H ← merge(𝒳_H) ;
16 return 𝒳_H;
```

Note that because of the third hypothesis we formerly made (accessibility of M from each state), Algorithm 1 terminates. This is because the hypothesis ensures that at every iteration at least one state is assigned to a cluster (either an existing one or a new one). Therefore, after a finite number of iterations the clustering algorithm terminates. As previously stated, the clusters determined by Algorithm 1 depend on MS and δ and their impact on the clusters should now be clear. A too large value for δ leads to cluster with heterogeneous values for c, while small values for δ lead to the creation of many small clusters. Similar conclusions can be made for MS. In the experimental section we will analyze the sensitivity to MS. The negative effect of a poorly chosen δ can be mitigated by MS (when δ is too large) or by the merging algorithm described next (when δ is too small).

Merging. The cluster size influences the performance of the algorithm, because too large clusters diminish the effectiveness of the algorithm, whereas too many small clusters induce large approximations. The MS parameter is used to counter the first problem, i.e., the creation of too large clusters. The merging algorithm presented here counters the other problem, i.e., the presence of too many small clusters. To this end, small clusters are merged together, where the definition of *small cluster* is defined by

a new parameter mS (minimum size). Algorithm 2 shows how merging is performed. All clusters are examined (line 2), and if they are smaller than mS (line 3) they are combined with one of their neighboring clusters. To this end, all neighbors are determined (line 4) and the one with the most similar cost is picked (line 4). If the combined size does not exceed the upper limit MS (line 6) then the clusters are merged (line 7). The process is repeated until no more clusters can be merged (loop starting at line 1). The algorithm terminates returning the new set of merged macrostates (line 9.) Note that the merging algorithm does not merge the macro state M.

Algorithm 2: Merge Algorithm

Data: $\mathcal{X}_H$: Set of all macro-states excluding the macro state $\{M\}$
Result: $\mathcal{X}_H$: New set of macro-states
1 **repeat**
2 **for** $M \in \mathcal{X}_H$ **do**
3 **if** $|M| < mS$ **then**
4 M_{adj}= Post(M) ;
5 $M_c \leftarrow \arg\min_{M_c \in M_{adj}} |c(M_c) - c(M)|$;
6 **if** $|M_c| + |M_{adj}| < MS$ **then**
7 merge M and M_c and update $\mathcal{X}_H$;
8 **until** *no more states are merged*;
9 **return** $\mathcal{X}_H$;

4.2 Hierarchical Action Set

Once the set of macro states $\mathcal{X}_H$ has been created, the set of actions U_H for the HCMDP easily follows. For $Y \in \mathcal{X}_H$, the action set $U_H(Y)$ is identified considering all macro states that can be reached in one step from the states in $\mathcal{C}$, i.e.,

$$U_H(Y) = \{Z \in \mathcal{X}_H \mid \exists y \in Y \wedge z \in Z \wedge u \in U(y) \wedge P^u_{yz} > 0\}. \tag{6}$$

It is important to note that according to this definition $U_H(Y)$ is generated starting from the original action set U, but its elements are actions that are not in U. In particular, each action in $U_H(Y)$ is a macrostate in $\mathcal{X}_H$.

4.3 Transition probabilities, costs, and initial probability distribution

The definition of an HCMDP is completed by the definition of the transition probabilities P_H, costs (c_H and $d_{i,H}$), bounds ($D_{i,H}$), and initial probability distribution β_H. With the objective of creating a method applicable independently from the structure of the underlying state space, we opt for a Monte Carlo approach whereby transition probabilities and costs are estimated through sampling.[1] In this section, in the interest of space, we only describe how transition probabilities can be estimated through sampling, since the same idea can be applied to estimate the costs.

[1] To the best of our knowledge no method has been proposed to analytically estimate costs and probabilities.

Given $M_1, M_2 \in \mathcal{X}_H$ and $M_3 \in U(M_1)$, we aim at estimating $P^{M_3}_{M_1,M_2}$. It is worthwhile recalling that although M_3 is a macro state too, in this context it is an action applied from macro state M_1 and it shall be interpreted as the action aiming at moving from macro state M_1 to macro state M_3. As per the definition of $U_H(M_1)$ given in Eq. (6), $P^{M_3}_{M_1,M_2}$ is non zero only if M_1 and M_2 are adjacent, where adjacency is defined by Eq.(6). The probability estimation method is as follows. Let $B_{M_1,M_3} = \{y \in M_3 \mid \exists x \in M_1 \wedge \exists u \in U(x) \wedge P^u_{xy} > 0\}$, i.e., the boundary between M_1 and M_3. We then build a graph $G = (V, E)$ where $V = M_1 \cup B$ and an edge is added between vertices $v_1, v_2 \in V$ whenever $P^u_{v_1,v_2} > 0$ for some $u \in U(v_1)$. Then, for each state $x \in M_1$ we compute the shortest path from x to B, where shortest is defined in terms of number of edges. These paths define a policy[2] π over M_1 to move from M_1 to M_3. Note that this policy is not optimal. Next, using a uniform probability mass function over the states in M_1 we randomly select one state $x \in M_1$ and simulate policy π. Following π the state eventually leaves M_1 to enter either M_3 or some other macro state. Through repeated executions, these policy simulations allow to estimate $P^{M_3}_{M_1,M_j}$ for each M_j.

Hierarchical costs c_H and $d_{i,H}$ are estimated using a similar approach. The difference in this case is that the average is taken over the overall costs accrued during the simulation of an application of action M_i from state M_j. For the hierarchical bounds $D_{i,H}$ we use the same costs in the original CMDP. Finally, β_H is built from β by adding up the probabilities of the states within each macro state, i.e., for each macro state Y_H we define $\beta_H(Y_H) = \sum_{x \in Y_H} \beta(x)$.

4.4 Hierarchical planning

The hierarchical planner operates in two steps and Algorithm 3 illustrates them. The algorithm takes as input a CMDP $\mathcal{C}$, a starting state s and a set of goal states M. Since we assumed that a starting state $s \in \mathcal{X}$ is provided as input, the initial distribution β is zero everywhere except in s. An HCMDP $\mathcal{C}_H$ is built as per our previous description (line 1), and an optimal policy is determined (loop starting at line 3). At each iteration the linear program is solved (line 4), and an optimal policy π^*_H for $\mathcal{C}_H$ is computed. If a policy cannot be found (line 5), the bounds $D_{i,H}$ are increased until the associated linear program is solved (line 6) In particular, each of the $D_{i,H}$ is increased by a fixed percentage. In our examples presented later on, the increment is 10% and there is an obvious tradeoff between selecting a larger or a smaller increase, since a smaller increase will give a sharper bound, but multiple small increases may be necessary before a satisfactory bound is determined and the problem solved. The rationale behind increasing the bounds, is that the hierarchical problem may be unsolvable due to the approximations induced in computing the costs through Monte Carlo sampling (in particular, overestimation of the costs $d_{i,H}$). It shall however be noted that these increased bounds may end up being higher than the original ones. While we believe that for simple problem settings it may be possible to derive quantitative bounds, this appear to be not trivial for the general case, and is left for future work. After a solution is found (line 8), the optimal policy for the hierarchical CMDP is determined (line 9). This concludes the first stage of the algorithm. In the second stage, starting at line 10,

[2] The set of paths define a policy because for each vertex they identify an edge to traverse along the shortest path, and by construction this edge is associated with an action.

a policy for the original CMDP is extracted solving a sequence of smaller CMDPs to move from one macro state to the next. To be specific, the smaller CMDPs are built as follows. Assume the state in the original CMDP is $\mathcal{C}$ is $s \notin M$ (otherwise the problem is already solved and the second loop at line 11 terminates.) This state belongs to exactly one macro state $Y_H \in \mathcal{X}_H$ because $\mathcal{X}_H$ is a partition of $\mathcal{X}$ (line 12.) The optimal policy π^*_H defines the action $Z_H \in U(Y_H)$ to execute, i.e., it identifies the next macro state to move into (line 14.) By construction, this macro state shares a boundary with Y_H, i.e., there exist a set of states $GoalSet \subset \mathcal{X}$ in the original CMDP reachable from some state in Y_H with a single transition (line 15). Therefore, from the original CMDP we extract a smaller CMDP whose state set is $Y_X \cup GoalSet$ and we compute a policy π_L to reach $GoalSet$ (line 16). Once the policy is computed, it is executed (line 18) until the state exists Y_H (either entering $GoalSet$ or another macro state). From there, the optimal policy π^*_H provides a new action to execute and the cycle terminates when the goal set M is reached.

Algorithm 3: HCMDP Planning

```
   Data: CMDP C = (X, U, P, c, d_i, D_i, β), s, M
 1 Build HCMDP C_H = (X_H, U_H, P_H, c_H, d_{i,H}, D_{i,H}, β_H);
 2 Solved ← false;
 3 while Not Solved do
 4 |   Solve LP associated with HCMDP;
 5 |   if LP unfeasible then
 6 |   |   Increase each bound D_{i,H} of ΔD_{i,H} > 0;
 7 |   else
 8 |   |   Solved ← true;
 9 Extract optimal aggregate policy π*_H (Eq. 5);
10 x ← s;
11 while x ∉ M do
12 |   Determine state Y_H containing s ;
13 |   if Y_H ≠ M then
14 |   |   Z_H ← π*_H(Y_t) ;
15 |   |   GoalSet ← Frontier(Y_H, Z_H);
16 |   |   π_L ←SolveLocalCMDP(x, GoalSet) ;
17 |   |   repeat
18 |   |   |   Follow policy π_L and update x ;
19 |   |   until x exits Y_H;
```

We conclude this section stating two theorems whose proofs are given in the appendix of the paper.

Theorem 1 *Let $\mathcal{C} = (\mathcal{X}, U, P, c, d_i, D_i, \beta)$ be a CMDP and let $\mathcal{C}_H = (\mathcal{X}_H, U_H, P_H, c_H, d_{i,H}, D_{i,H}, \beta_H)$ be an HCMDP built from $\mathcal{C}$ with the method described in this section. If the number of samples tends to infinity, then $\mathcal{C}_H$ preserves the connectivity of $\mathcal{C}$.*

Theorem 2 *Let $\mathcal{C} = (\mathcal{X}, U, P, c, d_i, D_i, \beta), s, M$ be the input to Algorithm 3. Then, as the number of samples tends to infinity, Algorithm 3 builds a solvable HCMDP.*

Note that the second theorem states the existence of a policy, but does not state anything about its optimality. Theorems 1 and 2 only guarantee asymptotic convergence in the number of samples, but do not provide indications on the rate of convergence as a function of the number of samples. This is consistent with results in existing

literature for motion planning, like the probabilistic roadmap method [21], rapidly exploring random trees [24], and the more recent optimal method RRT* [20]. In some instances, knowledge of certain parameters characterizing the environment, or making simplifying assumptions about its structure, it may be possible to derive relationships between the number of samples and the rate of convergence. However, for more general cases this is an open question and subject to future work.

5 Experimental Validation

In this section we investigate the viability of the method we presented. In particular, we aim at assessing the trade off between the computational gains we obtain using a hierarchical solution and the gap between the optimal non-hierarchical solution and the sub-optimal hierarchical solution. Moreover, we also experimentally determine the sensitivity to the parameters influencing the clustering method we presented. The algorithm is evaluated in three different scenarios, namely a matlab based simulation, a Gazebo based simulation, and an implementation on a Pioneer P3AT robot operating in an indoor environment. Each of the three different setups offer complementary features.

5.1 Matlab Simulations

We start our study with numerous matlab simulations allowing to quickly evaluate how the performance changes as the parameters are modified. Three different methods are compared. The first is the baseline non-hierarchical CMDP solver producing optimal policies solving the linear program given in Eq. (4). This method will be indicated as NH (non-hierarchical) in the tables and charts. The second is the method we presented in [15] that relies on a fixed structure for partitioning of the state space. In particular, for mobile robot tasks it relies on a state space clustering leading to a rectangular decomposition of the environment (see Figure 7). In tables and charts this method will be indicated as "Fixed". The third is the method we propose in this paper. All methods are subject to the same costs and transition probabilities.

All examples we will consider in the following feature two costs and then cannot be considered with a plain MDP solver. The first cost c is the cumulative risk accrued along a trajectory based on a preassigned risk map. The second cost d is the Euclidean path length. Note that although we consider two costs only, all solving approaches can handle an arbitrary number of costs. The first test environment is shown in Figure 2 and it models an outdoor environment in which the robot has to navigate between selected couples of start/goal locations. The heat map shows the risk map, with higher risk locations indicated by warmer colors. The sub figure on the right shows two different solutions obtained imposing different constraints on the path length. Note that the white trajectory traverses areas with higher risk because it is subject to a more stringent bound on path length, so it cannot afford to take the same detour generated by the red trajectory. This map is referred to as "terrain map" in the following. Figure 3 shows the clusters created by Algorithm 1 when processing this map.

The second test environment is depicted in Figure 4 and it is based on an indoor factory-like environment. The associated risk map is shown as well, where the darker areas with highest risk are associated with the obstacles. This map is referred to as

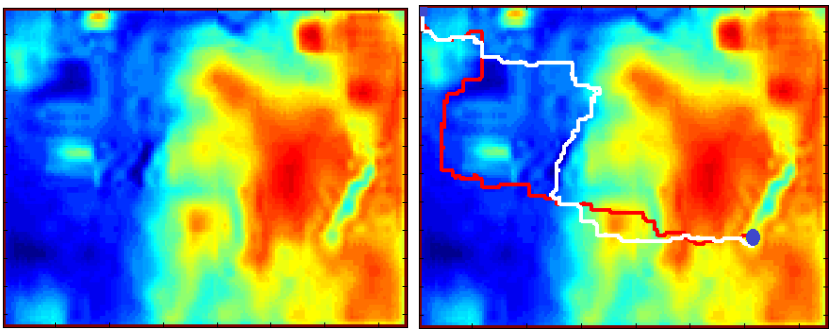

Fig. 2 Sample terrain map where warmer colors represent high risk areas to be avoided. The right picture shows two different solutions obtained with different constraints.

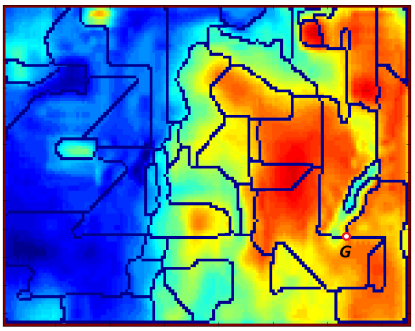

Fig. 3 Macro states created by Algorithm 1 on the terrain map environment. The goal state is evidenced by the letter G and a red circle as it constitutes a standalone cluster.

"maze" in the following. In both maps the cost d (Euclidean distance) is set to 1 for every state/action pair.

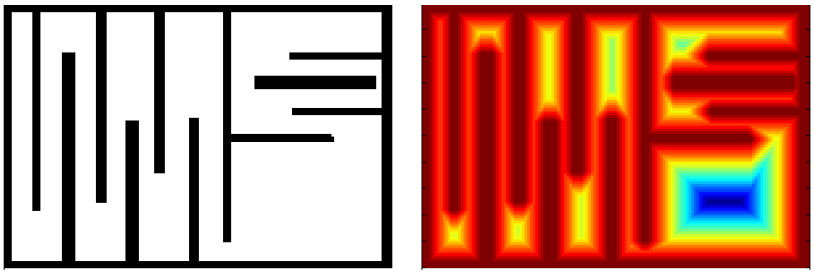

Fig. 4 Maze map where fixed partitioning fails. The left panel shows the map whereas the right panel displays the associated risk map.

Both environments are defined over a grid in which we assume 4-connectivity and we correspondingly define four actions for each state when possible.[3] Each action succeeds

[3] For states close the boundary or to an obstacle, the action set is adjusted by removing actions that would violate these constraints.

with probability 0.8. When the action fails, any of the nearby neighbors can be reached with equal probability. Three different performance measures are used in the following to compare the different algorithms. The first is the time spent to determine the policy. The second is the expected value of the risk objective function c, and the last is the value for the additional constrained cost d (path length). All algorithms are coded in matlab and rely on the built in `linprog` function to solve the linear programs. To ensure a fair comparison, since the hierarchical methods solve multiple instances of smaller linear programs, the displayed time is the cumulative sum of the time spent to solve all linear programs needed to compute a solution. The non-hierarchical method instead solves just one large linear program and the time spent to solve it is what we use for comparison.

Table 1 displays the time spent to solve 12 different problem instances on the terrain map environment. Each instance is defined by a different couple of start/goal locations. The top two rows display the performance of the non hierarchical and fixed resolution planner whereas the bottom twelve rows display the time spent by the HCMDP method for different combinations of its parameters. In particular, X/Y means that the maximum cluster size MS is X, and the number of samples used to estimate probabilities and costs is $Y\%$ of the number of states in the clusters. The prefix *nm* stands for *non merged*, and indicates that the merging step at the end of Algorithm 1 was not performed. Throughout this section, the δ parameter is set as the average difference for the c function between each node and all of its neighbors. The table shows that the performance difference between the non hierarchical method and the hierarchical strategies considered (Fixed and HCMDP) varies between a factor of 17 (case 3) and 1000 (case 4).

Alg	1	2	3	4	5	6	7	8	9	10	11	12
NH	107	94	54	1303	613	43	860	86	743	151	59	106
Fixed	3.19	2.91	2.2	1.3	0.8	1.79	2.3	1.5	2.5	0.89	1.89	1.7
100/10	4.59	5.49	3.89	1.72	2.72	2.20	2.43	1.82	3.21	1.43	3.39	2.55
100/30	7.04	5.68	4.02	1.82	1.79	7.33	2.48	3.02	3.59	1.79	3.06	3.59
100/50	6.17	8.45	3.79	1.70	1.65	7.48	2.37	5.48	5.14	2.58	4.34	4.21
nm/100/50	5.3	3.48	3.15	1.76	1.56	2.54	11.8	2.87	2.51	1.92	2.93	2.76
200/10	4.9	3.53	4.06	2.57	1.08	3.47	2.58	3.53	2.9	1.02	3.44	4.57
200/30	3.89	3.21	3.74	2.84	1.49	2.72	2.98	4.01	3.72	1.28	2.84	3.81
200/50	4.75	4.50	4.45	2.38	1.64	2.36	2.48	3.78	5.21	1.21	3.51	3.77
nm/200/50	4.03	3.44	3.02	1.31	1.76	2.63	1.84	4.83	4.07	1.49	2.45	2.20
400/10	7.17	6.02	5.25	5.64	1.71	8.12	3.17	3.76	3.31	2.26	3.45	3.85
400/30	6.41	6.83	5.05	2.52	2.31	8.06	5.43	2.90	4.08	2.42	7.73	4.86
400/50	6.80	5.65	5.41	5.14	2.48	8.09	3.06	2.56	4.39	2.3	3.47	3.89
nm/400/50	5.32	3.98	9.72	6.28	7.16	11.0	4.9	5.92	4.5	2.32	3.83	4.62

Table 1 Time spent (seconds) by the various algorithms for the terrain map on 12 different instances.

Table 1 also shows that, as expected, HCMDP is slower than the fixed method since it performs a more sophisticated partitioning, but the performance gap is in general modest, though it widens as expected as the number of samples in the Monte Carlo estimation grows.

Another interesting to assess the performance gain is by looking at the number and size of the linear programs solved by the non-hierarchical and hierarchical methods. The terrain map environment generates a constrained linear program with more than

65,000 variables, and the maze environment creates instances with more than 41,000 variables. The number of variables is (roughly) given by the number of states multiplied by four, since each variable corresponds to a state/action pair (x, u) and there are four or less actions per state (states close to the boundary or to the obstacles have less than four actions.) Table 2 instead displays the size and number of subproblems generated by the hierarchical methods for the twelve problems considered in the terrain environment. Each row shows two numbers, with the top one being the number of variables of the largest linear program solved, and the bottom number showing the average number of problems solved. Both numbers are averaged over one hundered runs, and rounded to the closest integer. Note that the percentage of samples does not influence these numbers and is therefore not displayed. These numbers justify the large difference in time between non-hierarchical and hierarchical methods and motivate this line of research.

Alg	1	2	3	4	5	6	7	8	9	10	11	12
100	392 10	380 8	354 8	380 4	382 3	384 5	376 6	348 4	376 5	356 3	380 5	384 6
nm/100	356 14	348 13	316 10	284 6	358 6	348 8	304 8	320 9	320 7	340 8	312 9	276 6
200	756 6	780 5	800 3	800 3	800 2	760 5	780/4 4	756 3	800 4	740 3	704 2	760 4
nm/200	740 8	680 8	656 5	632 3	740 4	556 8	724 4	712 6	680 8	644 4	720 3	716 6
400	1560 4	1524 4	1460 3	1388 3	1512 2	1560 3	1540 3	1452 2	1524 3	1512 3	1420 2	1420 3
nm/400	1320 5	1272 6	1156 4	1420 3	1260 3	1340 2	1410 5	1316 3	1220 4	1340 8	1360 6	1360 4

Table 2 Size of the largest linear program (top number) and number of supbproblems (bottom number) for each algorithm and problem considered in the terrain map problem.

While the time to solve a problem instance is an important aspect, hierarchical methods would not be very useful if the performance increase comes at the cost of a inferior performance in terms of costs of the functions being optimized. Figure 5 and 6 analyze how these costs vary across the 12 different cases we considered. In particular, Figure 5 shows the average cost for the primary c cost (risk), whereas Figure 6 plots the average d cost (path length). In both instances averages are obtained over 100 independent runs (variances are small, and so averages are representative.)

Figure 5 shows that as expected the non-hierarchical method achieves in general the best performance in terms of minimizing the expected c cost. In some cases (e.g., 2 and 8) some instances of the HCMDP solver obtains a better cost. This is due to the fact that if the HCMDP planner cannot solve a certain problem instance for given constraints on the d cost, it will increase the D bounds in an attempt to make the linear program feasible. Hence, in some instances, taking advantage of the increased bound for the d cost, it is possible to lower the c cost as well because the additional budget for the traversed length may give the possibility to avoid risky areas. The reader could also observe that in many instances the Fixed partitioning method works competitively with both the non hierarchical method and HCMDP. However, as it will be evidenced discussing the maze environment, this strategy is in general prone to failure, although in the terrain map this is not evident since there are no obstacles.

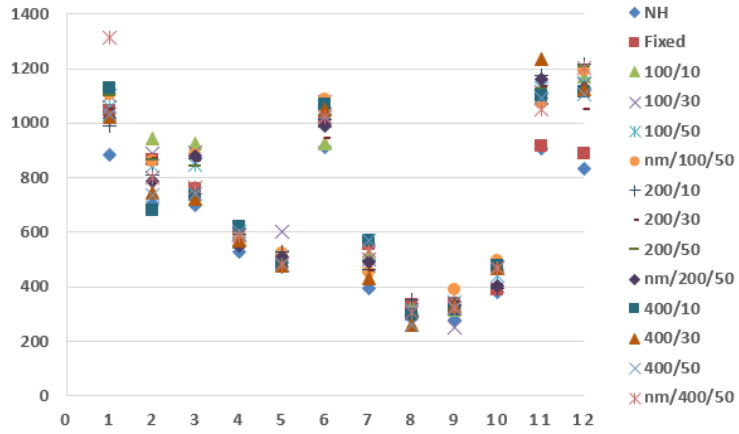

Fig. 5 Average c cost (risk) over 100 independent runs for the terrain map environment.

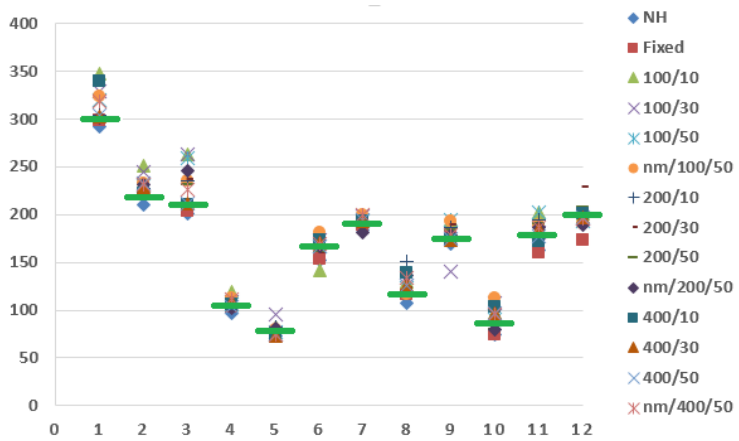

Fig. 6 Average d cost (path length) over 100 independent runs for the terrain map environment. Horizontal green bars show the value of the constraint in the original, non-hierarchical problem formulation.

Figure 6 shows instead the average for the constrained path length cost d. Here we see that the variations are more contained, demonstrating that only in few instances the bound needs to be lifted for making the problem feasible.

We next consider the maze map. This environment is interesting because it shows that approaches relying on clustering methods not considering the underlying map are in general doomed to fail. Figure 7 shows the policy produced by the algorithm we formerly presented in [15]. The dotted lines show the clusters boundaries, whereas the red arrows illustrate the computed policy. The example shows that in some macro states the policy suggests transitions that cannot be executed because of an obstacle cutting through. This problem could be overcome reducing the size of the clusters, but for any fixed clustering strategy one can devise an environment leading to unfeasible policies. For this reason, adaptive clustering algorithms considering the underlying structure of the state space are needed. These include the one we presented, as well as [4].

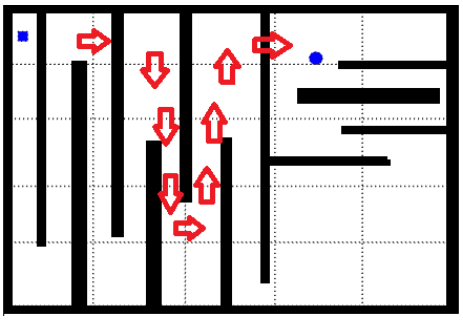

Fig. 7 Example showing how fixed partitioning fails. Partitions are shown using dotted lines. The hierarchical CMDP induces the policy shown by the red arrows that cannot be executed due to the obstacles cutting through the macro states.

Alg	1	2	3	4
100/10	5.63	4.64	3.41	2.22
100/30	3.08	3.89	3.20	2.10
100/50	5.69	4.54	3.76	2.54
100/70	5.69	5.22	3.92	2.44
100/90	5.73	4.80	3.78	2.45
200/10	8.52	7.95	5.05	8.48
200/30	9.89	5.44	6.97	7.76
200/50	9.08	8.88	5.66	6.74
200/70	9.62	7.51	6.37	6.28
200/90	9.07	9.11	5.84	7.31

Table 3 Time spent (seconds) by the various algorithms for the maze map on 4 different instances.

Given that the fixed-clustering approach is not a viable solution for this environment, and having shown in the previous section that the non-hierarchical solution is largely outperformed by our method, the analysis of the maze environment is then restricted to the HCMDP algorithm. Similarly to Table 1, Table 3 show the time spent (in seconds) to compute the policy as the size of the cluster and the number of samples is varied. The table confirms that for this metric the size of the cluster is the most relevant parameter, whereas there is less sensitivity to the number of samples. Finally, Figures 8 and 9 show the performance for the functions c and d. The findings confirm what previously observed, i.e., that the performance loss is contained.

One last question to be addressed is how to pick the parameters for the HCMDP planner. Looking at Table 1, for example, we see that the performance varies with the maximum size of the cluster (MS) and the number of samples. The second parameter has rather straightforward interpretation, since the number of samples is a fixed percentage of the number of states in the macrostate, and there is an obvious tradeoff, in the sense that more samples imply higher accuracy but require more time. The choice for the cluster size, instead, is less intuitive. With the objective of getting some experimental insights on how to select this value, we performed a series of experiments where we varied the size of the cluster and observed its impact on the computational time and the quality of the solutions. Figure 10 shows the results for three different problem instances in both the terrain and the maze problem. Note that since the problems have different state spaces, results are normalized expressing in both cases MS as the percentage of the number of states rather than as an absolute number. It is interesting

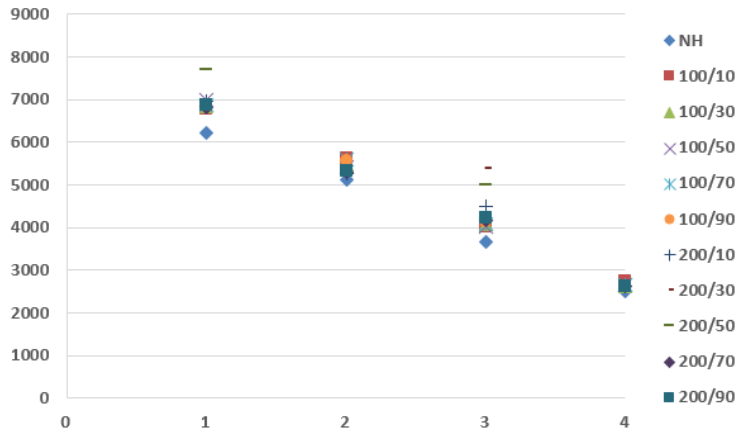

Fig. 8 Average c cost (risk) over 100 independent runs for the maze environment.

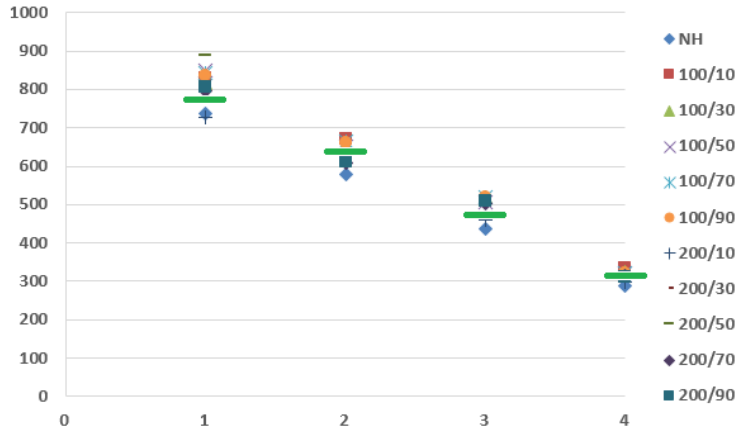

Fig. 9 Average d cost (path length) over 100 independent runs for the maze environment. Horizontal green bars show the value of the constraint in the original, non-hierarchical problem formulation.

to notice that for both problems and all cases considered, picking MS equal to 1% of the number of states in the non-hierarchical problem seems to give the best result.

Moreover, in Figures 11 and 12 we evaluate how the cluster size influences the objective function (risk) and the additional constrained costs (path length). Experimental data show that the impact of MS on these quantities is limited. Therefore, it is reasonable to conclude that setting MS equal to 1% of the size of the state space seems to be a good starting point. Of course, when domain specific knowledge is available, better choices could be made. An analytic investigation on how to select the cluster size is subject to future work.

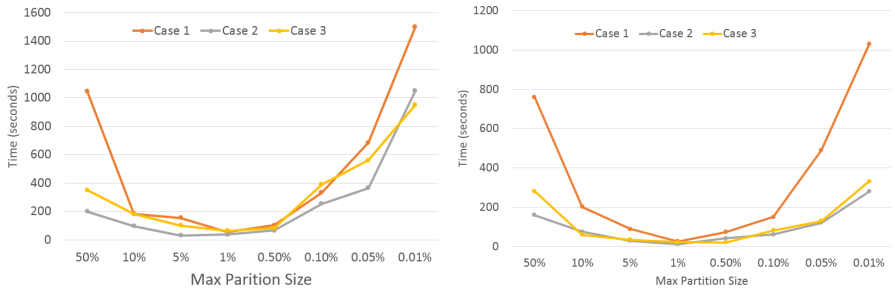

Fig. 10 Time to compute the solution with HCMDP as a function of the cluster size MS. The left figure shows the trends for the terrain problem, and the right figure shows the trends for the maze problem.

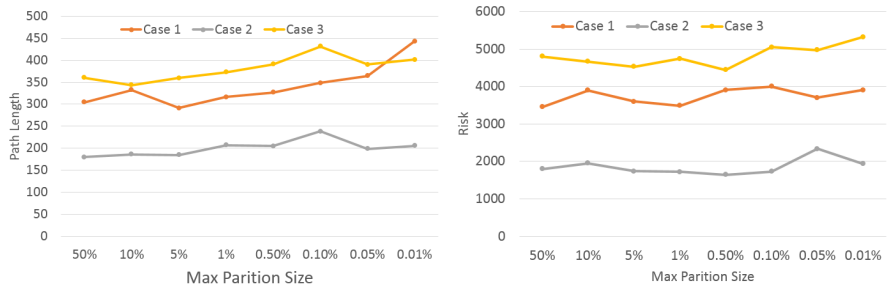

Fig. 11 Time to compute the solution with HCMDP as a function of the cluster size MS. The left figure shows the trends for the terrain problem, and the right figure shows the trends for the maze problem.

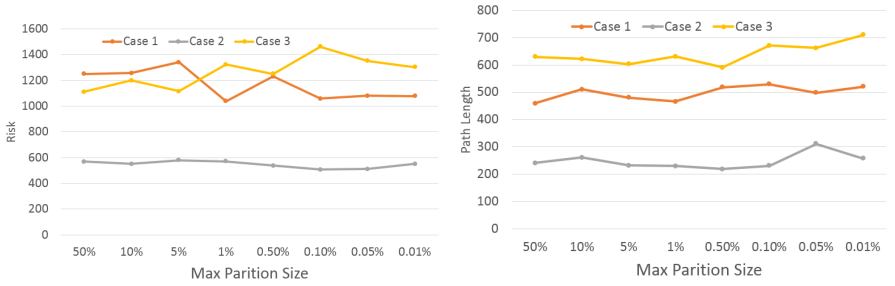

Fig. 12 Time to compute the solution with HCMDP as a function of the cluster size MS. The left figure shows the trends for the terrain problem, and the right figure shows the trends for the maze problem.

5.2 Gazebo Simulations

With the objective of eventually validating the HCMDP planner on a real robotic platform, as an intermediate step we developed and tested the planner using ROS and the Gazebo simulator. The simulated robotic platform exactly matches the robot we use in the real world experiments, i.e., a P3AT equipped with odometry, sonar, and the SICK LMS proximity range finder (see Figure 18). The software architecture used both in Gazebo and on the real robot relies on standard ROS nodes. In particular,

we use a particle filter for localization [28] and we therefore assume that a map of the environment is available. In both cases the map was preliminarily built driving the robot around manually and using the GMapping SLAM algorithm [17] available in ROS. Figure 13 shows a rendering of the simulated environment (top), the map built by the SLAM algorithm (bottom left) and the associated risk map for the primary cost c (bottom right). The simulated map is $21.7m \times 20m$ and is discretized into square cells of size $0.5m \times 0.5m$. In each cell we assume the availability of four actions (up, down, right, left). This assumption is consistent with our software design, where navigation to a given target point relies on the ROS navigation stack. In particular, the ability to execute an action (say "up") is not influenced by the current orientation of the robot, and therefore the state is represented just by the cell where the robot is located (without considering its orientation). According to the framework we presented, for each state x (i.e., grid cell) the policy computed by the planners returns an action $\pi(x)$ that can be translated to a point in one of the nearby grid cells and then fed to the navigation stack to move the robot there.

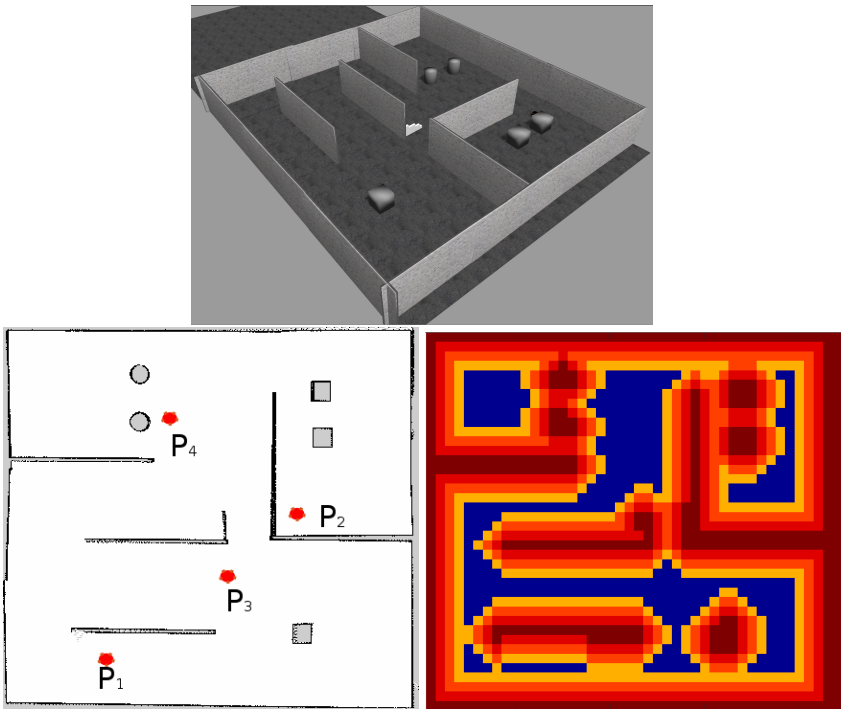

Fig. 13 Simulated environment used for the Gazebo simulations and associated risk map.

An essential step to setup a planner based on an MDP approach is to estimate the transition probabilities, i.e., P^u_{xy}. To estimate these quantities, each action is executed 100 times and transition probabilities are derived by counting. To explore the sensitivity of the policies to transition probabilities, two surfaces with different friction coefficients were used. In the first case the probability of executing a motion with success is 0.66

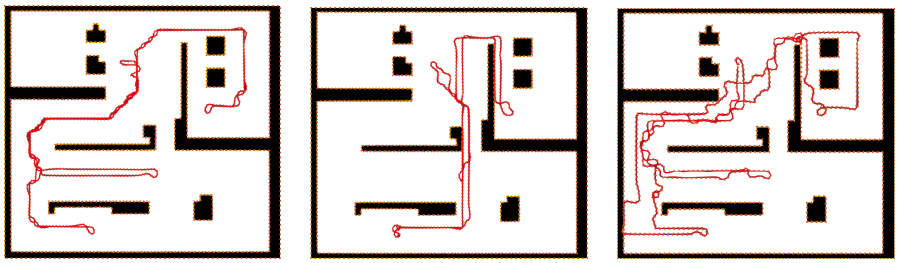

Fig. 14 Sample paths obtained by the three planners solving the same problem instance: Left: MDP, Center: CMDP, Right: HCMDP

whereas in the second case it is 0.51. When a motion does not succeed, a uniform probability over adjacent states (excluding the target state) is used.

Figure 13 (bottom left) shows four different points in the map used as start/goal locations to compute the policy. Four different cases were considered. The first starts at P_1 and ends at P_2; the second starts at P_2 and ends at P_3; the third starts at P_3 and ends at P_4; and the fourth starts at P_4 and ends at P_1. Policies were computed using three different planning strategies, namely MDP, CMDP, and HCMDP. For the MDP planner the objective is to minimize the cumulative c cost according to the risk map shown in figure 13. MDP is included as an additional term of comparison, although it solves a simpler problem because it considers just the primary cost c (risk) and ignores the additional cost d (Euclidean distance). Figure 14 shows some examples of the resulting paths. Every test was repeated 20 times to estimate average behaviors. The leftmost panel in figure 14 shows that the MDP solution takes the safest path since it almost entirely lies inside the low risk areas (refer to figure 13). This path is however longer. This is consistent with the objective function we setup, and to the lack of constraints on path length. The middle panel shows that, due to the constraints on path length, the planner produces a path that cuts through some higher risk areas to meet the bound. Finally, the rightmost panel shows that the HCDMP planner displays a behavior that is somehow intermediate, in the sense that sometimes it favors a safer subpath, and sometimes it instead selects a shorter path. This is very evident when considering point P_2. In such case the HCMDP planner produces a path that approaches P_2 like the CMDP planner (shorter but riskier), but then leaves the point along the same path determined by the MDP planner. This is due to the fact that when leaving P_2 the hierarchical planner has to relax the constraints to achieve feasibility, and therefore it can produce a path with higher length but lower risk.

Figures 15 and 16 show the first set of results, where the HCMDP planner was tested using different values for the maximum cluster size and sampling rate. Consistently with the expectations, Figure 15 shows that the MDP achieves the lowest risk, whereas it is again shown that tuning the parameters it is possible to obtain similar performance between CMDP and HCMDP. Figure 16 also confirms our previous observations with regard to variations in the secondary cost. Based on these two sets of results, it appears that the most convenient setup for the HCMDP planner is obtained when the maximum cluster size is set to 200 and the sampling rate is 90%. Therefore in the following we will just consider these parameters.

Figure 17 shows an additional set of results where the friction coefficient is set to be much lower and then the robot is subject to more slippage when moving (and

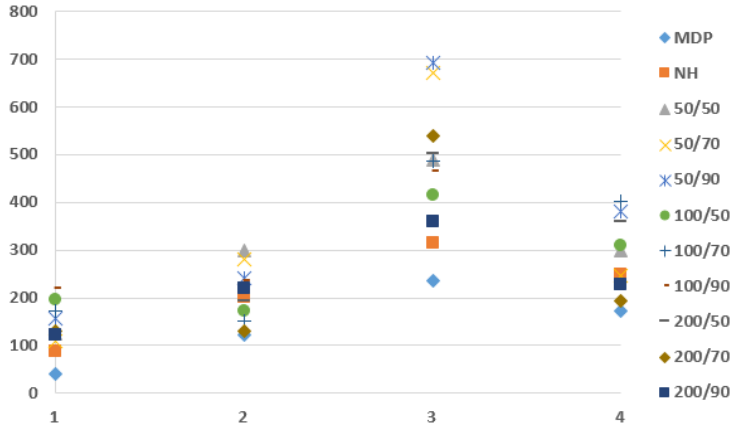

Fig. 15 Average c cost (risk) over 20 independent runs for intermediate friction.

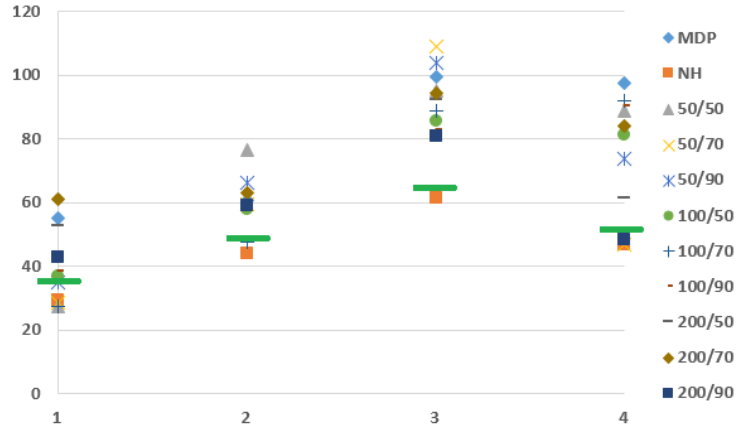

Fig. 16 Average d cost (path length) over 20 independent runs for intermediate friction. Horizontal green bars show the value of the constraint in the original, non-hierarchical problem formulation.

then more uncertainty). The figure confirms the conclusions we formerly derived, thus showing insensitivity to the transition probability rates.

5.3 Real Robot Experiments

To conclude, we executed the same three algorithms to control a P3AT robot (see figure 18) navigating inside one of the university buildings. Figure 19 shows the map of the environment preliminarily obtained using the GMapping algorithm. The maximum dimensions of the environment are $42m \times 71m$, and the map was split in square cells of size $0.5m \times 0.5m$. Consistently with the previous experiments, Figure 20 displays the risk map associated with the map. As for the Gazebo simulations, transition probabilities were extracted through repeated simulation of the various actions. Figure 19 also shows four points used to define the start and goal locations for the various

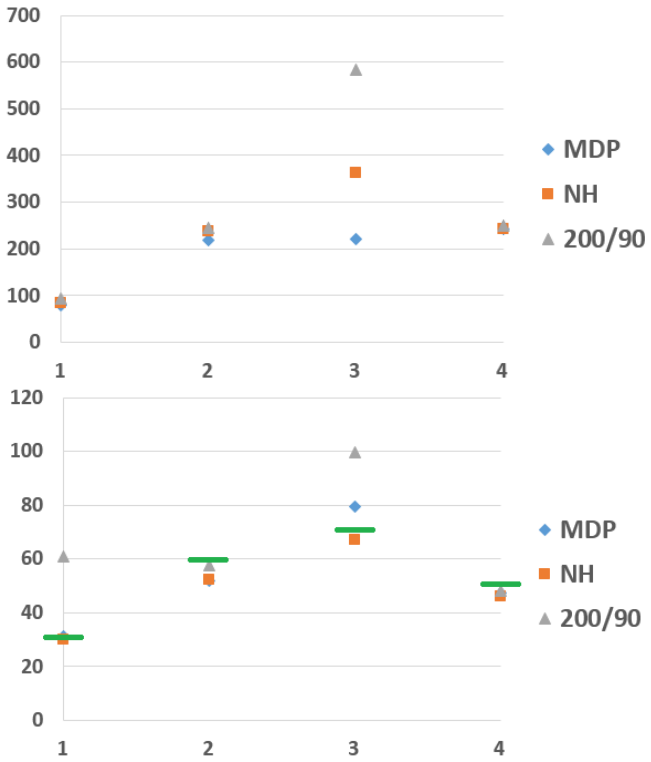

Fig. 17 Top: average c cost (risk). Bottom: average d cost (path length). Charts display the average of 20 independent runs in an environment with reduced friction between the wheels and the soil. Horizontal green bars show the value of the constraint in the original, non-hierarchical problem formulation.

Fig. 18 The P3AT robot used to test the HCMDP planner in the real world.

missions. To determine averages, every mission was executed 10 times, and over the various runs, the robot drove more than 5.5km inside the building totaling over more than 25 hours of autonomous operation. Figure 21 shows four paths produced by the HCMDP planner while computing a policy from point P_1 to P_2, then to P_3, afterwards to P_4, and ending at P_1. Columns in Figure 22 represent these four paths respectively.

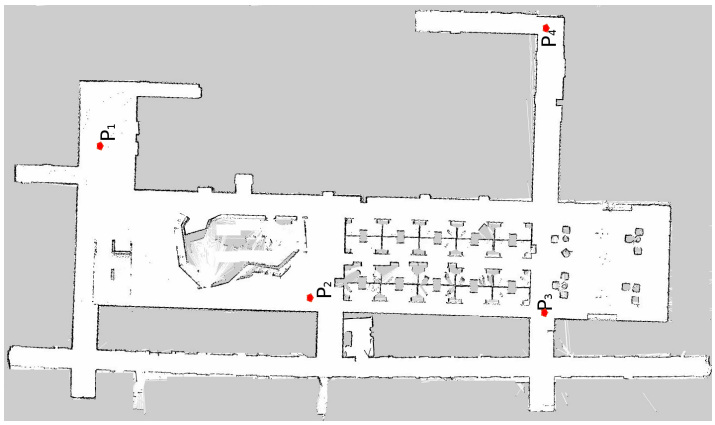

Fig. 19 Map of the university building used to the the HCMDP planner.

Fig. 20 Risk map associated with Figure 19.

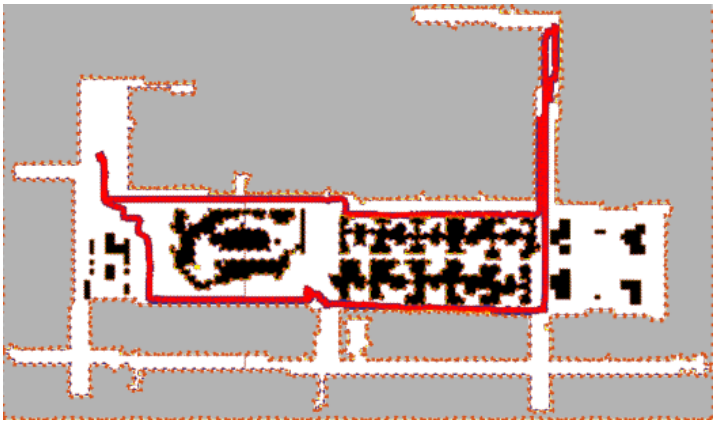

Fig. 21 Four paths produced by the HCMDP planner while computing a policy from point P_1 to P_2, then to P_3, afterwards to P_4, and ending at P_1 (see also Figure 19).

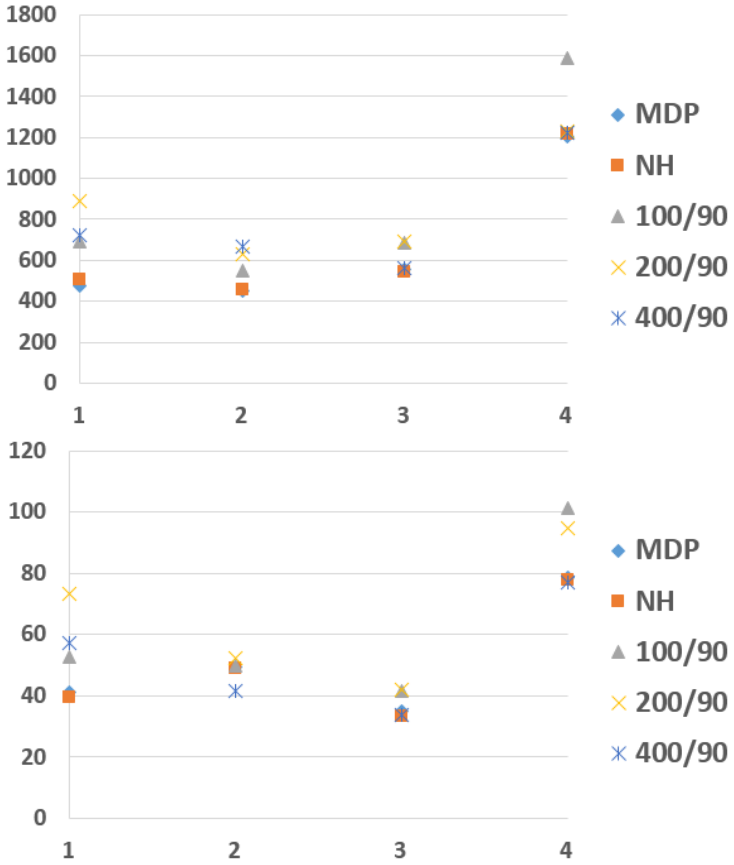

Fig. 22 Results obtained averaging 10 runs with the real robot. Top: average c cost (risk). Bottom: average d cost (path length).

As we did in the previous cases, Figure 22 displays the average primary cost c (risk) and the average secondary cost d (path length). Three different versions of the HCMDP planner were compared, where differences were in the size of the clusters (100, 200, 400), whereas the percentage of samples was fixed at 90. Experiments with the real robot confirm the conclusions we anticipated in Matlab and Gazebo simulations. In particular, referring to the bottom figure we outline that the path length constraint was 45 in the first case, 55 in the second, 40 in the third, and 85 in the fourth. It can be observed that the planner with cluster size 400 is rather competitive with the non hierarchical version for both costs.

6 Conclusions and Future Work

This paper proposed HCMDP, an algorithm to solve CMDPs using a hierarchical approach. The technique presented in this paper is valuable because it offers an efficient way to solve sequential stochastic decision making problems considering more than one cost. While CMDPs offer an optimal solution to these problems, their computational

burden limits their applicability in real world applications. HCMDP overcomes this problem while at the same time ensuring that valuable properties (e.g., feasibility) are maintained. The algorithm has been extensively validated in simulation and on a real robot, and the results show significant reductions in the computational time paired with modest gaps in the expectations of the cost functions.

Various interesting open questions remain. The first is whether it is possible to derive formal bounds for the performance gap between the policy produced by the non-hierarchical CMDP and HCMDP. This gap is likely to be a function of parameters like the maximum cluster size and the number of samples used in the Monte Carlo estimation, but at the moment it appears that tight bounds may be not easy to determine. The second question pertains to the choice of the parameters. It would be interesting to derive intuitions on how they should be selected for a given problem instance, perhaps in terms of an heuristic, or through an iterative guessing process. Another question is related to the estimation of the parameters for the HCMDP. Within the Monte Carlo framework, estimation can be performed in many different ways, and the one we explored in this paper is just one of the possible choices. A more in depth study would most likely determine better strategies, either in terms of performance or estimation accuracy.

On the experimental side, at the moment we have contrasted the HCMDP method only with the optimal non-hierarchical alternative. Of course, there is a range of sub-optimal methods one could use to expedite the solution of sequential stochastic decision problems, in particular for the unconstrained MDP scenario. As part of our future work, more numerical comparisons between sub-optimal solvers will be done.

Finally, with the emergence of massive multi-core architectures, it would be interesting to experiment how performance scales if one uses multiple cores to perform the Monte Carlo estimation and to solve the intermediate CMDP instances.

Appendix

Proof of Theorem 1. Definition 2 establishes two conditions for saying that an HCMDP preserves connectivity. The first requires that $\mathcal{X}_H$ is a partition of $\mathcal{X}$. Algorithm 1 never considers a state twice, i.e., once a state has been assigned to a cluster it will not be considered again for assignment (line 4). Moreover, the main loop ensures that all states in $\mathcal{X}$ are assigned to a cluster. Therefore, $\mathcal{X}_H$ is a partition of $\mathcal{X}$.

We next turn to the second condition. Let $z \in \mathcal{X}'$ and $y \in M$ be two states such that $z \rightsquigarrow y$. By definition this means that there exists a sequence of states $\mathcal{S} = s_1, s_2, \ldots, s_n$ such that for each each $1 \leq i \leq n-1$ $P^{u_i}_{s_i,s_{i+1}}$ for some $u_i \in U(s_i)$ and $s_1 = y$ and $s_n = z$. Since $\mathcal{X}_H$ is a partition of $\mathcal{X}$, this sequence of states is associated with a sequence of macrostates $Z_H = S_1 \ldots S_n = Y_H$ such that $s_i \in S_i$ for each i. Note that in general there could be some repeated elements in the sequence of macrostates. Let $S_1, \ldots S_k$ $(k \leq n)$ be the sequence obtained removing subsequences of repeated macrostates.[4] First note that this sequence includes at least two elements. This is true because we started assuming $y \notin M$ while $z \in M$. According to Algorithm 1 all and only the states in M are mapped to an individual macrostate (line 1), so y cannot be in the same macrostate as z. Next, consider two successive elements in the sequence of

[4] This means that if $S_i = S_{i+1}$ we remove the latter from the sequence and we reiterate this step until $S_i \neq S_{i+1}$ for all symbols left in the sequence.

macrostates, say S_i and S_{i+1}. By construction, there exist two successive states in $\mathcal{S}$, say s_j and s_{j+1}, such that $s_j \in S_i$ and $s_{j+1} \in S_{i+1}$. Since these two states are part of $\mathcal{S}$, there exists one input $u_j \in U(s_j)$ such that $P^{u_j}_{s_j,s_{j+1}} > 0$. As per Eq.(6), this implies that an action S_{j+1} is added to the set of actions $U(S_j)$. Next, consider the method described in subsection 4.3, and in particular the definition of the boundary B between two macro states. It follows that $s_{j+1} \in B_{S_i,S_{i+1}}$. The algorithm further continues computing the *shortest path* between each state in S_i and B, where the shortest path is computed over the induced graph G. For s_i the path trivially consists of a single edge to s_{i+1} (or some other vertex in B that is also one hop away from s_i.) Next, the algorithm randomly selects one vertex from S_j using a uniform distribution and executes the policy to reach B. Let m be the total number of Monte Carlo samples generated. Then the probability that the estimate of $P^{S_{i+1}}_{S_i,S_i+1} = 0$ is bounded from above by

$$(1-\gamma)^{k_1}(1-P^{u_j}_{s_j,s_{j+1}})^{k_2}$$

where $\gamma = \frac{1}{S_i}$, k_1 is the number of times s_j was not sampled and k_2 is the number of times s_j was sampled ($k_1 + k_2 = m, k_{1,2} \geq 0$). This proves that as the total number of samples m grows, the estimate for $P^{S_{i+1}}_{S_i,S_i+1}$ will be eventually be positive. This reasoning can be repeated for each couple of successive macro states, thus showing that $Z_H \rightsquigarrow Y_H$, and this concludes the proof.

Proof of Theorem 2. We start observing that Algorithm 3 builds and solves a sequence of HCMDPs. Each is a CMDP with a suitable set of parameters and at every iteration the constrained linear program given in Eq. (4) is solved. Theorem 1 guarantees that state M_H is accessible from every macrostate, and therefore there exists at least one policy π' for which $c(\pi')$ is finite. Let us next consider the inequality constraints in Eq. (4). If the linear program is not feasible, then each bound $D_{i,H}$ is increased by $\Delta D_{i,H}$ (line 6.) By construction, all costs $d_{i,H}(x,u) \geq 0$ for each state/action pair (x,u). Let $n_s = |\mathcal{K}'_H|$ be the number of state/action pairs in the HCMDP, $d_{max} = \max_{(x,u)\in K'_H}\{d_{i,H}(x,u)\}$ the largest among the additional costs, and $D_{min} = \min\{\Delta D_{i,H}\}$ the smallest among the increments in line 6. Therefore after at most $\lceil \frac{n_s d_{max}}{D_{min}} \rceil$ iterations all inequality constraints become feasible.

Acknowledgments

This paper extends preliminary results presented in [16]. This work is supported by the National Institute of Standards and Technology under cooperative agreement 70NANB12H143. Any opinions, findings, and conclusions or recommendations expressed in these materials are those of the authors and should not be interpreted as representing the official policies, either expressly or implied, of the funding agencies of the U.S. Government.

References

1. E. Altman. *Constrained Markov decision processes.* CRC Press, 1999.
2. A. Bai, F. Wu, and X. Chen. Online planning for large mdps with maxq decomposition. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 3*, pages 1215–1216, 2012.

3. J. Barry, L. P. Kaelbling, and T. Lozano-Pérez. Hierarchical solution of large markov decision processes. Technical report, MIT, 2010.
4. J. L. Barry, L. P. Kaelbling, and T. T. Lozano-Pérez. DetH*: Approximate hierarchical solution of large markov decision processes. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2011.
5. D. P. Bertsekas. *Dynamic programming and optimal control*, volume 1,2. Athena Scientific Belmont, MA, 2005.
6. J. Bouvrie and M. Maggioni. Efficient solution of markov decision problems with multiscale representations. In *Communication, Control, and Computing (Allerton), 2012 50th Annual Allerton Conference on*, pages 474–481. IEEE, 2012.
7. S. Carpin, M. Pavone, and B.M. Sadler. Rapid multirobot deployment with time constraints. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1147–1154, 2014.
8. M. El Chamie and B. Açikmeşe. Convex synthesis of optimal policies for Markov Decision Processes with sequentially-observed transitions. In *Proceedings of the American Control Conference*, pages 3862–3867, 2016.
9. Y.-L. Chow, M. Pavone, B.M. Sadler, and S. Carpin. Trading safety versus performance: rapid deployment of robotic swarms with robust performance constraints. *ASME Journal of Dynamic Systems, Measurement and Control*, 137, 2015.
10. P. Dai and J. Goldsmith. Topological value iteration algorithm for markov decision processes. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1860–1865, 2007.
11. P. Dai, Mausam, and D. S. Weld. Focused topological value iteration. In *International Conference on Automated Planning and Scheduling*, 2009.
12. P. Dai, Mausam, D.S. Weld, and J. Goldsmith. Topological value iteration algorithms. *Journal of Artificial Intelligence Research*, 42(1):181–209, 2011.
13. X. C. Ding, B. Englot, A. Pinto, A. Speranzon, and A. Surana. Hierarchical multiobjective planning: From mission specifications to contingency management. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 3735–3742. IEEE, 2014.
14. X. C. Ding, A. Pinto, and A. Surana. Strategic planning under uncertainties via constrained markov decision processes. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 4568–4575, 2013.
15. S. Feyzabadi and S. Carpin. Risk aware path planning using hierarchical constrained markov decision processes. In *Proceedings of the IEEE International Conference on Automation Science and Engineering*, pages 297–303, 2014.
16. S. Feyzabadi and S. Carpin. HCMDP: a hierarchical solution to constrained markov decision processes. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 3791–3798, 2015.
17. G. Grisetti, C. Stachniss, and W. Burgard. Improved techniques for grid mapping iwht Rao-Blackwellized particle filters. *IEEE Transactions on Robotics*, 23(1):36–46, 2007.
18. M. Hauskrecht, N. Meuleau, L. P. Kaelbling, T. Dean, and C. Boutilier. Hierarchical solution of markov decision processes using macro-actions. In *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*, pages 220–229. Morgan Kaufmann Publishers Inc., 1998.
19. J. Hoey, R. St-Aubin, A. J. Hu, and C. C. Boutilier. SPUDD: Stochastic planning using decision diagrams. In *Proceedings of Uncertainty in Artificial Intelligence*, pages 279–288, 1999.

20. S. Karaman and E. Frazzoli. Sampling-based Algorithms for Optimal Motion Planning. In *International Journal of Robotics Research*, 30(7):846–894, 2011.
21. L. E. Kavraki, P. Švetska, J. C. Latombe, and M.H. Overmars. Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces. In *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.
22. M. J. Kochenderfer. *Decision Making Under Uncertainty: Theory and Application*. MIT press, 2015.
23. S. M. LaValle. *Planning algorithms*. Cambridge university press, 2006.
24. S. M. LaValle, J. J. Kuffner. Randomized Kinodynamic Planning. In *International Journal of Robotics Research*, 20(5):378–400, 2001.
25. T. M. Moldovan and P. Abbeel. Risk aversion in markov decision processes via near optimal Chernoff bounds. In *NIPS*, pages 3140–3148, 2012.
26. J. Pineau, N. Roy, and S. Thrun. A hierarchical approach to pomdp planning and execution. In *Workshop on hierarchy and memory in reinforcement learning (ICML)*, volume 65, page 51, 2001.
27. M.L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley-Interscience, 2005.
28. S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. MIT Press, 2006.
29. N. A. Vien and M. Toussaint. Hierarchical monte-carlo planning. In *AAAI*, pages 3613–3619, 2015.