HCMDP: a Hierarchical Solution to Constrained Markov Decision Processes

Seyedshams Feyzabadi Stefano Carpin

***Abstract*— Constrained Markov Decision Processes offer a principled way to tackle sequential decision problems with multiple objectives. Although they could be very valuable in numerous robotic applications, to date their use has been quite limited. One of the reasons is that their solution requires to solve constrained linear programs with a large number of variables and this is computationally demanding, especially when considering dynamic environments. In this paper we propose a hierarchical approach to solve large CMDPs. States are clustered into macro states and relevant parameters like transition probabilities and costs are extracted with a Monte Carlo approach. Macro states are created with the objective of grouping together states with similar costs while preserving feasibility. We illustrate the value of our findings in a path planning scenario where the robot moves through an environment characterized by different risk levels. Our approach largely outperforms the non-hierarchical method and we also show how it prevails over methods based on fixed partitioning strategies.**

I. Introduction

In this paper we present a hierarchical method for the solution of constrained Markov decision processes (CMDPs). Markov decision processes (MDPs) continue to be extensively used to solve sequential decision problems where the state is observable and the outcome of actions is uncertain. When the state is not observable, computationally more demanding methods like partially observable MDPs (POMDPs) are used. However, in numerous situations state observability is a legitimate hypothesis. This is for example the case when the state is used to model the pose of a robot operating outdoor and equipped with appropriate sensors (e.g., GPS and gyro). While the MDP formulation considers a single objective function, in numerous engineering domains there are multiple objectives to be concurrently considered. When this is the case, one could combine them into a single objective function, see e.g., [15]. This approach is however problematic because the resulting objective function does not have an immediate physical meaning and it is influenced by the specific choice made when combining the various components together. Alternatively, one can optimize with respect to one objective function and introduce constraints on the others. This is the standpoint adopted in CMDPs. Determining the optimal policy for a CMDP requires to solve an often large constrained linear program and there is then interest in techniques to reduce the associated computational costs. To mitigate this problem, we embrace a hierarchical paradigm (HCMDP – hierarchical CMDP). As we discuss in Section II, similar ideas have been already explored for MDPs, but their application to CMDPs is missing and former methods include some limitations that we overcome. Our eventual objective is to use CMDPs to solve motion planning problems of unmanned ground vehicles (UGV) used in manufacturing environments (e.g., autonomous forklifts) and operating in human inhabited environments. In particular, we are interested in developing motion plans accounting for both efficiency and various safety measures. For example, the robot should at the same time optimize task specific measures (e.g., length of traversed path, consumed energy, etc.), while containing certain risk measures, like crossing through heavily trafficked areas. Note that in some instances it may be unavoidable to take risky actions in order to complete the task, but the planner should keep these episodes to a minimum. To this end, we use CMDPs to compute a solution accounting for these concurrent objectives.

The main idea behind HCMDP is to partition the state space of the original CMDP into clusters and to compute a policy over this smaller space. Then, the solution is projected back into the original state space, by solving a number of simpler CMDP instances. This method presents two problems. The first concerns *completeness*, i.e., depending on how the clusters are formed one may lose connectivity to the goal set. The second problem concerns *optimality*, i.e., the solution obtained through the hierarchical approach is not as good as the optimal solution in the original problem. Provided that the decrease in performance is bounded, the second problem can in general be tolerated, whereas care should be taken to avoid the first one. In this paper we show how it is possible to create clusters to guarantee feasibility. Additional problems connected to the hierarchical solution of CMDPs are related to the definition of costs and transition probabilities in the hierarchical CMDP. To estimate these quantities, we use a Monte Carlo based approach that is broadly applicable and does not hinge on specific properties of the underlying state space.

The remainder of the paper is organized as follows. Related work is discussed in Section II. The basic CMDPs theory is presented in Section III. Section IV illustrates HCMDP and in Section V we present various simulations

S. Feyzabadi and S. Carpin are with the School of Engineering, University of California, Merced, CA, USA.

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.

illustrating its properties. Finally, conclusions are offered in Section VI.

II. Related work

Literature in MDPs is vast. The reader is referred to [4] for a general introduction to MDPs and to [14], [16] for applications to robotics. A comprehensive treatise about CMDPs is given in [1]. Despite MDPs have been extensively used in robotics and the theory of finite CMDPs is well understood, the use of CMDPs in robotics has been somehow limited and most applications are found in other domains, e.g., communication networks. We posit that their limited use is a consequence of their computational complexity. In robotics, Ding et al. [11] used CMDPs to solve planning problems. More recently, we used CMDPs to solve the rapid multirobot deployment problem [5], [6].

Hierarchical solutions to MDPs have been proposed in the past with the objective of improving computational efficiency and to compute policies that could be reused for similar subproblem instances. Dai et al. proposed multiple approaches to create hierarchical models of large MDPs [7]. Topological value iteration guarantees to find the optimal solution of an MDP. Its partitioning method, however, becomes inefficient when applied to state spaces with poor connectivity. Focused topological value iteration [8], [9] improves the former method by building the value function on a subset of the state space including the starting point. The partitioning problem, however, persists. Barry et al. proposed their first hierarchical approach based on strongest connected components [2], and then improved that with DetH* [3]. Their methods, however, are focused on the solution of factored MDPs. SPUDD [13] was introduced to solve very large MDPs and saves values/policies as functions rather than using a numeric lookup table. MAXQ [10] is also used to solve MDPs in hierarchical form, but it relies on human input to define the clustering of states. In our previous work [12] we proposed a first hierarchical solution to CMDP problems. Our findings showed that a significant speedup could be obtained with modest losses in terms of objective functions. However, our previous partitioning method was based on a predetermined schema that does not guarantee feasibility in the hierarchical problem. In this paper we overcome this limitation.

III. Background

We shortly recap MDPs and CMDPs. The reader is referred to [4] and [1] for detailed introductions to the problem. Throughout the paper we exclusively consider finite models.

A. Markov Decision Processes

A finite MDP is defined by a quadruple (X, U, P, c) where:

- X is a finite state space with n elements.
- U the finite set of actions. $U(x)$ is the set of actions that can be taken in state $x \in X$.
- $P : X \times U \times X \to [0, 1]$ is the transition probability function. $P(x, a, y)$ is the probability of moving from state x to state y when applying action $a \in U(x)$. Being a probability, this function satisfies the normalization requirements, and is written as P^a_{xy} for brevity.
- $c : X \times U \to \mathbb{R}_{\geq 0}$ is a cost function. $c(x, a)$ is the cost incurred when applying action a while being at state x.

Note that both $P(x, \cdot, y)$ and $c(x, \cdot)$ are defined only for actions in $U(x)$, and this fact will be tacitly assumed in the following. A policy $\pi : X \to U$ associates an action to a state, and it is known that for MDPs there is no loss of optimality assuming that π is deterministic. Given a start state $x_0 \in X$, the policy π induces a stochastic process over the set of states, where X_{i+1} is the random variable representing the state reached from X_i by applying action $\pi(X_i)$. Different cost models have been considered in literature. In this paper we focus on the infinite horizon total cost, whose definition hinges on the definition of absorbing MDP. An MDP is absorbing if the state space X can be partitioned into X' and M such that for each policy π:

1) $\sum_t P^\pi(X_t = x) < \infty$;
2) $P^a_{xy} = 0$ for each $x \in M$ and $y \in X'$,

where $P^\pi(X_t = x)$ is the probability that $X_t = x$ while following policy π. If we assume that $c(x, a) = 0$ for each $x \in M$, then for an absorbing MDP we can the define the infinite horizon total cost of a policy π as

$$c(\pi) = \mathbb{E}\left[\sum_{t=0}^{\infty} c(X_i, \pi(X_i))\right]$$

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. Note that this cost exists because of the absorbing property, i.e., we are guaranteed that for each policy the state will eventually enter and remain in M, where no additional cost is accrued. Solving an MDP means to determine the policy π^* minimizing the cost just defined. Dynamic programming provides various approaches to compute the solution, e.g., value iteration and policy iteration.

B. Constrained Markov Decision Processes

CMDPs extend MDPs by introducing additional costs, and imposing constraints on them. A CMDP is defined by (X, U, P, c, d_i, D_i) where X, U, P, c are defined as for MDPs and:

- $d_i : X \times U \to \mathbb{R}_{\geq 0}$ with $1 \leq i \leq k$ are k additional cost functions.
- D_i are k non negative bounds.

When action a is applied in state x, in addition to $c(x, a)$ each of the additional costs $d_i(x, a)$ is incurred as well. The definition of an absorbing CMDP follows the definition of an absorbing MDP, but we additionally require that $d_i(x, a) = 0$ for each $x \in M$. Given an absorbing CMDP and a policy π the following costs are then defined:

$$c(\pi, \beta) = \mathbb{E}\left[\sum_{t=0}^{\infty} c(X_i, \pi(X_i))\right]$$

$$d_i(\pi, \beta) = \mathbb{E}\left[\sum_{t=0}^{\infty} d_i(X_i, \pi(X_i))\right]$$

where β indicates the mass distribution for X_0, i.e., $\beta(x) = \Pr[X_0 = x]$. This additional parameter is needed because the solution of a CMDP in general depends on the initial distribution of state. The CMDP problem is to determine a policy π^* solving the following optimization problem:

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

Despite their similarities, there are three main differences between MDPs and CMDPs. First, the optimal policy for a CMDP may require randomization, whereas deterministic policies are sufficient to achieve optimality in MDPs. Second, the optimal solution depends from the initial distribution β, whereas for MDPs the optimal policy is independent from the initial state. Third, MDPs are normally solved using dynamic programming but this is not true for CMDPs. The optimal policy for a CMDP is found solving a constrained linear program defined as follows. Let $\mathscr{K} = \{(x, a) | x \in X', a \in U(x)\}$ be the state-action space, and and $\rho(x, a)$ a set of optimization variables associated to each element in $\mathscr{K}$. Then the optimization defined in Eq. 1 has a solution if and only if the following linear program is feasible:

$$\begin{aligned} \min_{\rho} \quad & \sum_{(x,a)\in\mathscr{K}} \rho(x, a) c(x, a) \\ \text{s.t.} \quad & \sum_{(x,a)\in\mathscr{K}} \rho(x, a) d_i(x, a) \leq D_i \quad 1 \leq i \leq k \\ & \sum_{(y,a)\in\mathscr{K}} \rho(x, a)(\delta_x(y) - P^a_{yx}) = \beta(x) \quad \forall x \in X' \\ & \rho(x, a) \geq 0 \quad \forall (x, a) \in \mathscr{K} \end{aligned} \tag{2}$$

where $\delta_x(y) = 1$ when $x = y$ and 0 otherwise. If the linear program is unfeasible, then the CMDP cannot be solved, i.e., no policy can satisfy the constraints. If the linear program is feasible, then the optimal solution $\rho(x, a)$ induces an optimal randomized policy π^* defined as

$$\pi^*(x, a) = \frac{\rho(x, a)}{\sum_{a\in A(x)} \rho(x, a)} \quad x \in X', a \in U(x) \tag{3}$$

where $\pi^*(x, a)$ is the probability of executing action a while in state x. Note that the policy is only defined for states in X' and can be arbitrarily defined for states in M because of the assumption that the CMDP is absorbing.

The linear program associated with a CMDP can easily include a very large number of optimization variables $\rho(x, a)$ and its solution becomes therefore computationally demanding (see Section V for some examples). This problem is particularly daunting when CMDPs are used plan the actions of of a robot operating in a dynamic environment, and changes in the surroundings require the repeated solution of the large linear program. Starting from these motivations, in the following we present a hierarchical method to accelerate the solution while preserving feasibility and limiting performance loss.

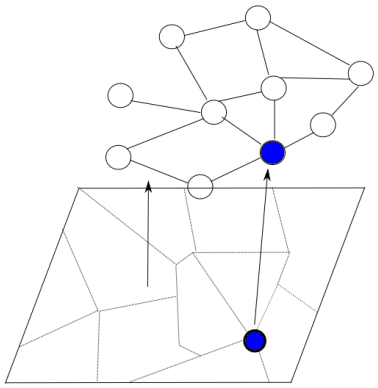

Fig. 1: HCMDP Approach. The lower layer represents the states of the MDP, whereas the higher layer represents the macro-state graph. The blue area shows the set M and is preserved in the two layers.

IV. HCMDP – Hierarchical Constrained Markov Decision Processes

Hierarchical methods to solve MDPs have been considered since quite some time (see e.g., [4], vol II, Ch. 6). The principle can be applied to CMDPs too, although to the best of our knowledge this has not been explored yet, with the exception of our recent work [12]. The overarching idea is to create a smaller CMDP (i.e., an instance with fewer states) through clustering of states, compute a policy for the smaller instance, and then utilize this high level strategy to infer a policy for the original problem (see Figure 1). More formally, starting from a CMDP (X, U, P, c, d_i, D_i), we extract a HCMDP $(X_H, U_H, P_H, c_H, d_{i,H}, D_{i,H})$ with the objective that $|X| << |X_H|$. Next, we compute the optimal policy π^*_H for the HCMDP and we use it to extract a policy π' for the original CMDP. Note that while π^*_H is guaranteed to be optimal for the HCMDP, in general π' will not preserve optimality for the non-hierarchical CMDP. Our proposition, however, is that the loss in optimality is compensated by the gain in computational efficiency.

Despite these principles are well accepted, no general purpose solution has been proposed so far, and this approach relies on the solution of various intermediate problems. The first concerns *clustering*, i.e., how to build X_H from X. In the following we present a method that is guaranteed to preserve connectivity. Next, it is necessary to define the action set U_H for the clustered state space. The last modeling problem is the definition of transition probabilities between clusters of states, and the computation of the state/action costs. To solve this problem we adopt a Monte Carlo approach. The final problem is how to infer a policy π' over X starting from π^*_H. In the following we discuss how we solve each of these issues.

A. Clustering

The first problem we tackle is how to compute X_H from X. X_H must be a partition of X and its elements will be called clusters or macro states. The key idea to effective partitioning is to group together states with *similar* characteristics, where similarity is in this case measured in terms of the cost function c. As mentioned in the introduction, care must be taken when creating macro states in order to preserve connectivity. This concept can be formalized as follows. We say that x is connected to y and we write $x \rightsquigarrow y$ if there exists a sequence of states $s_1, \ldots, s_n \in X$ and actions $a_1, \ldots a_{n-1} \in U$ such that $s_1 = x$, $s_n = y$ and $P^{a_i}_{s_i, s_{i+1}} > 0$ for $1 \leq i \leq n-1$. Let $z \in X'$ and $y \in M$ be two states in the original CMDP, such that $z \rightsquigarrow y$, and let $Z_H, y_h \in X_H$ be the two macro states containing z and y in the hierarchical CMDP. Then, we require that $Z_H \rightsquigarrow Y_H$. Furthermore, given $S \subset X$, $x \in X$, and $a \in U(x)$ we define the following sets

$$\text{Pre}(S) = \{x \in X | \exists a \in U(s) \wedge \exists y \in S \wedge P^a_{xy} > 0\}$$
$$\text{Post}(x, a) = \{Y \in X_H | \exists y \in Y \wedge P^a_{xy} > 0\}$$
$$\text{Post}(S) = \{Y \in X_H | \exists y \in Y \wedge P^a_{xy} > 0\}.$$

Note that $\text{Pre}(S)$ is a subset of states of X, whereas $\text{Post}(x, a)$ and $\text{Post}(S)$ are sets of macrostates of X_H. Algorithm 1 illustrates our clustering approach. The algorithm starts by creating a cluster with the absorbing set (line 1). Then, it repeatedly iterates over the set of states that have not been assigned to a cluster, but can reach one of the existing clusters in one step with non-zero probability (line 4). States are then assigned to a cluster if the size of the cluster does not exceed a maximum size (parmeter MS), and the average cost of a cluster is similar to $c(s, a_s)$, i.e., the cost incurred in order to move from s into the cluster (lines 6 to 14). We define $c(H)$ as the average of the costs for all the state/action pairs in H. If a state cannot be assigned to any of the existing clusters, then a new cluster with a single state is created (lines 15 to 19), and the process is iterated. At the end, an optional merging step described in the following is performed (line 23). Note that in the current description of the algorithm we assumed that all states are connected to each other and therefore they eventually are all assigned to a cluster. To enforce this property, one can preliminarily identify and eliminate states belonging to components not connected to M. The behavior of the clustering algorithm is defined by two parameters, namely the maximum size of each cluster MS, and the threshold δ used for the test whether $c(H) \approx c(s, a_s)$. The impact of these parameters is evident. Large values for δ lead to clusters of states with dissimilar values for c, thus defying the effectiveness of the clustering approach. On the other hand a too little value leads to too many clusters. In the experimental section we will analyze the sensitivity to MS. The negative effect of a poorly chosen δ is controlled by MS (when δ is too large) or by the merging step (when δ is too small).

1) Merging: Too large clusters negatively impact the performance of the algorithm and the parameter MS is introduced to control this aspect. However, too many small clusters also negatively impact the performance. To overcome this problem, small clusters can be combined together during a merging step. A parameter mS (minimum size) is used to trigger the merging process. Algorithm 2 shows how merging is performed. Clusters smaller than mS are combined with neighboring clusters. When multiple neighbors exist, the one with the most similar cost is picked, provided that it does not exceed the maximum size MS.

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 a_s ∈ U(s) do
7                  Y_s ← Post(s, a_s);
8                  for H ∈ Y_s do
9                      if c(H) ≈ c(s, a_s) ∧ |H| < MS
                       then
10                         assign s to cluster H;
11                         break out of two nested for
                           loops;
12                     end
13                 end
14             end
15             if s not assigned yet then
16                 create new cluster M_s = {s};
17                 add M_s to X_H;
18                 assign s to cluster M_s;
19             end
20         end
21     end
22 end
23 X_H ← merge(X_H);
24 return X_H;
```

B. Hierarchical Action Set

The set of clustered states induces a new set of actions U_H for the HCMDP. For a macrostate $Y \in X_H$, the action set $U_H(Y)$ is created considering all macrostates that can be reached in one step. Hence, $U_H(Y)$ is:

$$U_H(Y) = \{Z \in X_H | \exists y \in Y \wedge z \in Z \wedge a \in U(y) \wedge P^a_{yz} > 0\}.$$

Note that U_H is induced by X_H and the original action set U, but it is made of actions that were not part of U.

C. Transition probabilities and costs

The final step to complete the construction of HCMDP is the definition of transition probabilities P_H, costs c_H and $d_{i,H}$, and bounds $D_{i,H}$. To the best of our knowledge, no principled methods have been proposed to infer these quantities using an analytic approach valid irrespectively

Algorithm 2: Merge Algorithm

Data: X_H: Set of all macro-states
Result: X_H: New set of macro-states
repeat
  for $M \in X_H$ **do**
    if $|M| < mS$ **then**
      M_{adj}= Post(M) ;
      $M_c \leftarrow \arg\min_{M_c \in M_{adj}} |c(M_c) - c(M)|$;
      if $|M_c| < MS$ **then**
        merge M and M_c and update X_H;
      end
    end
  end
until *no more states are merged*;
return X_H;

of the structure of the underlying state space. In order to create a method that can be applied to arbitrary state spaces, we therefore opt for a Monte Carlo based approach in which transition probabilities and costs are estimated through sampling. The advantage of this approach is found in its broad applicability. The known disadvantage is that it does not lend itself to an easy characterization of its performance in terms of analytic bounds. We only describe the estimation method for the transition probabilities, since the idea is similar for the costs. For macro states M_1 and M_2 and action $M_3 \in U(M_1)$ we want to estimate $P^{M_3}_{M_1,M_2}$. This probability is defined only if M_1 and M_2 are adjacent, i.e., $M_1 \in U(M_2)$ and $M_2 \in U(M_1)$. The estimation process is as follows. We select one state $x \in M_1$ and one state $y \in M_2$. In both cases a uniform random distribution is used. Then we compute the single source shortest path from y using breadth first search. The graph used by BFS has all states in M_1 and M_2 and has and edge between vertices $a \in M_1$ and $b \in M_2$ whenever there is an action $c \in U(a)$ such that $P^c_{ab} > 0$. Single source shortest path identifies all vertices in M_1 reachable from y with a non-zero probability. This set includes x due to the way we built the macro states. By reversing the direction of the edges in the graph we then obtain a policy to go from x to y. Next, a simulation starts, that is we simulate the state evolution from x following the deterministic policy we just computed and using the transition probabilities in the original CMDP. Due to the underlying uncertainty in state evolution, it is possible that while following this policy the state will end in M_2 or in a different adjacent macro state. The probability $P^{M_3}_{M_1,M_2}$ is then estimated by taking the ratio between the number of times the state eventually reaches M_3 and the overall number of attempts. Note that the set of simulations provide also $P^{M_3}_{M_1,M_j}$ with $j \neq 2$ by counting how many times the evolution ends in M_j rather than M_2.

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. Finally, for the hierarchical bounds $D_{i,H}$ we use the same costs in the original CMDP. This combination of transition probability definition and clustering guarantees connectivity. The proof, omitted for brevity, follows the connectivity proof given in [3].

D. The hierarchical planner

Algorithm 3 sketches the structure of the hierarchical planner. The input to the algorithm is the original CMDP, the start state s, and the set of goal states M. The algorithm starts by computing the hierarchical structure (line 1) and then tries solving the associated linear program. If the linear program can not be solved, the various bounds are increased (line 6) and the problem is solved again. We assume all costs are increased, but one could also opt for a different increase strategy, e.g., increasing one at the time. 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 $d_{i,H}$). Once a strategy for the hierarchical CMDP is found, a sequence of smaller CMDPs is solved to move from one macro state to the next (line 15 to 23). The desired sequence of macro states to follow is given precisely by the policy for the hierarchical problem and exploits our definition of actions for the macro states. In particular, the action set for a macro state Y_H is given by the set of macro states sharing a boundary with it, so that the goal set computed at line 18 can be easily computed. The cycle terminates when the goal set is reached.

Algorithm 3: Algorithmic Sketch

Data: CMDP $= (X, U, P, c, d_i, D_i), s, M$
1 Build HCMDP $(X_H, U_H, P_H, c_H, d_{i,H}, D_{i,H})$;
2 $Solved \leftarrow false$;
3 **while** *Not Solved* **do**
4 Solve LP associated with HCMDP;
5 **if** *LP unfeasible* **then**
6 Increase each bound $D_{i,H}$ of $\Delta D_{i,H}$;
7 **end**
8 **else**
9 $Solved \leftarrow true$;
10 **end**
11 **end**
12 Extract optimal aggregate policy π^*_H (Eq. 3);
13 $x \leftarrow s$;
14 **while** $x \notin M$ **do**
15 Determine state Y_H containing s;
16 **if** $Y_H \neq M$ **then**
17 $Z_H \leftarrow \pi^*_A(Y_t)$;
18 $GoalSet \leftarrow$ Frontier(Y_H, Z_H);
19 $\pi_L \leftarrow$SolveLocalCMDP$(x, GoalSet)$;
20 **repeat**
21 Follow policy π_L and update x
22 **until** x *reaches GoalSet or exits* Y_H;
23 **end**
24 **end**

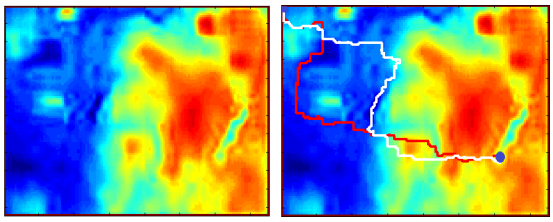

Fig. 2: Sample terrain map. Warmer colors represent high risk areas to be avoided. The right picture shows two different solutions obtained with different risk constraints.

V. Experiments and Results

In this section we experimentally compare three different approaches to solve CMDP problems. The first is the non-hierarchical method that solves the linear program given in Eq. 2. The second is the method we presented in [12] that relies on a fixed structure partitioning of the state space. The third is the method we described in the previous sections, and we study its sensitivity to some of its parameters.

As stated in the introduction, our objective is to study high-level motion strategies for autonomous vehicles used in industrial environments while pursuing multiple objectives. *High-level* in this case means that we abstract from the motion primitives of the vehicle. These can be accounted for in the transition probabilities between states and the associated costs. According to this approach, we focus on planar environments subdivided in regular grids. In this study we assume that cost c is a measure of *risk* and we consider a single additional cost d representing traveled distance. Our method, however, can include an arbitrary number of additional costs. According to this choice, the problems we study aim at determining paths between couple of points that minimize the overall expected risk while bounding the expected length of the traversed path. We consider two different test cases. The first is shown on the left side of Figure 2. Warmer colors are used to identify cells associated with higher values for the risk cost c. This map is referred to as *terrain* in the following. The second map is shown in the left side of Figure 3 where black areas are non traversable obstacles and the white area represents free space. Risk is defined as proximity to obstacles which and is shown in the middle panel of the same figure. This map is referred to as maze in the following. In both maps the cost d is set to 1 for every state/action pair.

Being defined over a grid, for both environments we assume 4-connectivity and we correspondingly define 4 actions for each state.[1] In the state transition model each action succeeds (i.e., the robot moves towards the desired next state) with probability 0.8. For example, if x has four neighbors and we consider the action $a =$*up*, we have $P^a_{xy} = 0.8$ for the state y above x. The remaining 0.2 probability is uniformly spread over x and the three states *left*, *right*, *bottom*. We consider three different performance measures. The first is the time spent to find the solution, the second is the value of the objective function c (risk), and the third is the value of the of additional constrained cost d (path length). The code is implemented in Matlab and linear programs are solved using the built in `linprog` function. To perform a fair time comparison, for the hirarchical methods we log the cumulative time spent inside all the calls to `linprog`.

Table I shows the performance on the terrain environment for 12 different problem instances, where a problem instance is defined as a couple of start and end points (their coordinates are given in figure 4 and 5 on the x axis). The first row of the table shows the timing for the non-hierarchical method (NH) while the second row displays the results for the hierarchical method using fixed partitioning. Successive rows show the results for the algorithm we propose while varying the parameter MS and the sampling rate in the Monte Carlo process. To be specific, 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.

To put the performance of the non-hierarchical method into perspective, we notice that the terrain environment generates a constrained linear program with more than 65000 variables whereas the maze environment induces a linear program with more than 41000 variables. These numbers explain the large difference in time between the non-hierarchical and hierarchical methods, and justify this line of research.

Next, we consider the value obtained for the objective function c (risk) and the bounded additional cost d (path length). Results are plotted in Figure 4 and 5 averaging the results of 100 independent runs. The first fact to observe is that the different methods achieve similar performance with regard to the constrained cost d (see Figure 5). This demonstrates that each solution tends to use almost all the allotted budget for d, as specified by the bound D. Considering Figure 4 one observes that, as expected the, non hierarchical method provides the best solution while the other methods in general behave similarly, with a slight better performance for hierarchical solutions using larger clusters. Moreover, instances not using the merging step behave in general worse. Two additional observations are in order. Fixed partitioning appears in general very competitive for this benchmark. However, in some instances it totally fails (like in maze), so it cannot be considered as a generally applicable solution. Second, in some instances (e.g., the second test case in Figure 4) it appears that the non-hierarchical method is outperformed by one of the hierarchical methods. This counterintuitive results is due to the fact that if the linear program for the hierarchical problem is unfeasible, the constraint D is increased (see Algorithm 3). This additional budget for the traversed length may give the possibility to avoid risky areas (high values of c) by taking a detour that is now allowed because of the increased D bound. This aspect is shown in the right panels of Figures 2 and 3. There we show how paths with lower constraints on length are forced

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

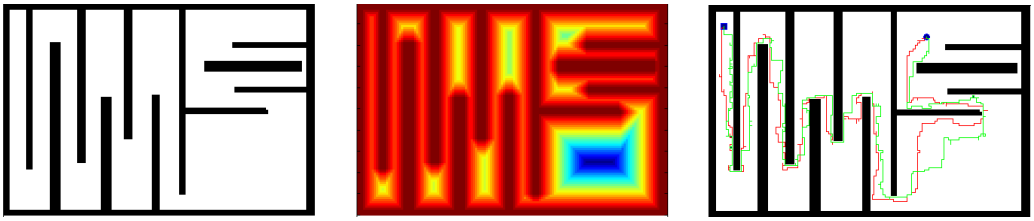

Fig. 3: Maze map where fixed partitioning fails. Left picture shows the map. Middle picture is the risk map associated with the map where the hotter colors illustrate riskier area. The right picture shows two different solutions.

Alg	1	2	3	4	5	6	6	8	9	10	11	12
NH	107	94.8	54.7	1303.4	613.8	43.9	860.2	86.6	743.9	151.4	59.3	106.2
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	11.84	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	11.03	4.9	5.92	4.5	2.32	3.83	4.62

TABLE I: Time spent by the various algorithms for the terrain map on 12 different instances.

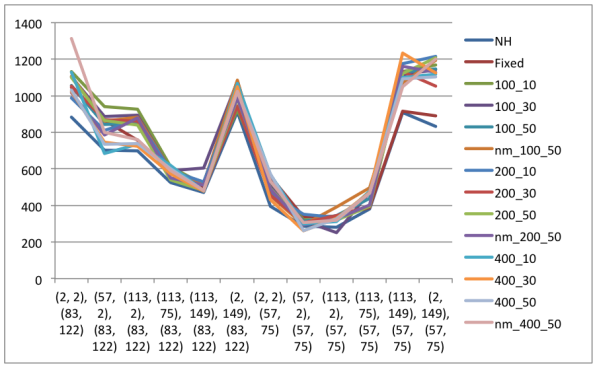

Fig. 4: Average c cost (risk) over 100 independent runs for the terrain environment.

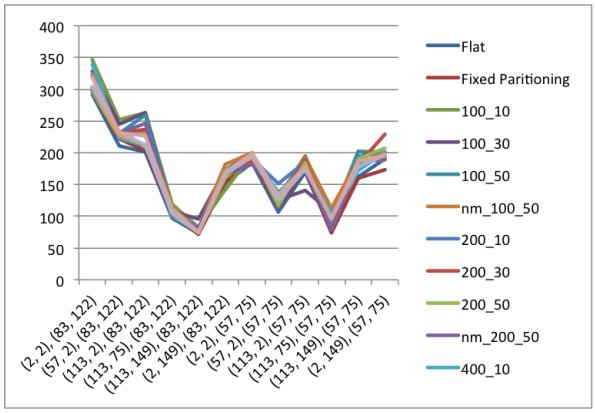

Fig. 5: Average d cost (path length) over 100 independent runs for the terrain environment.

through risky areas.

We next consider the maze environment. In this case the fixed partitioning method fails because it produces a policy that does not consider the obstacles in the environment (see Figure 6). While this example is specific to the fixed partitioning method we considered, it is easy to show that for any fixed partitioning strategy one can build a state space instance that will be clustered into macro states 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 [3].

Having assessed that our proposed algorithm outperforms the non-hierarchical approach, and that fixed clustering methods are inadequate, the analysis of the maze environment is therefore restricted to the algorithm we proposed. Table II

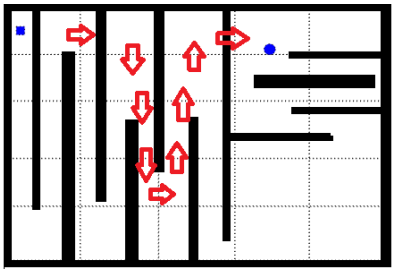

Fig. 6: Example showing how fixed partitioning fails. Partitions are shown using dotted lines. The hierarchical CMDP induces the policy shown by the red arrows that cannot be executed due to the obstacles cutting through the macro states.

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 II: Time spent by the various algorithms for the maze map on 4 different instances (in seconds).

shows the time spent solving the various instances of linear programs, as we did in Table I. The table confirms that for this metric the size of the cluster is the most relevant parameter, whereas variations are minor when one considers a fixed cluster size and then varies the number of samples.

Finally, Figures 7 and 8 show the performance for the functions c and d. The findings confirm what previously observed, i.e., that the performance loss is contained.

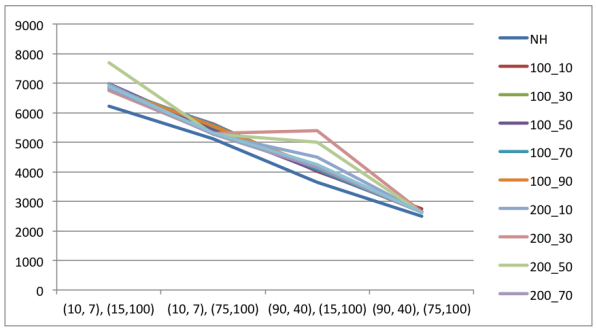

Fig. 7: Average c cost (risk) over 100 independent runs for the maze environment.

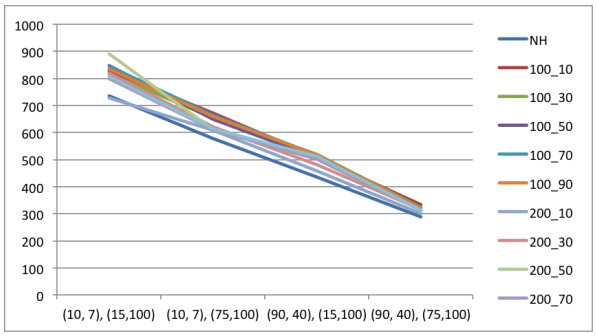

Fig. 8: Average d cost (path length) over 100 independent runs for the maze environment.

VI. CONCLUSIONS

In this paper we have presented a hierarchical solution to CMDPs. We believe that the CMDP framework is valuable to the robotics community because it offers a principled solution to multiobjective sequential decision problems and hierarchical solutions are necessary to expedite the solution of CMDPs. Our method hinges on two steps. A partitioning of the states guaranteed to preserve connectivity, and an estimation of parameters based on Monte Carlo sampling. Our experimental validation confirms that HCMDP offers large gains in terms of computational time while incurring in moderate losses in the objective functions. In the future we aim at deriving bounds for the performance loss. Moreover, by taking advantage of multi-core architectures we will accelerate the estimation of parameters with Monte Carlo sampling by running multiple instances in parallel.

REFERENCES

[1] E. Altman. *Constrained Markov decision processes*. CRC Press, 1999.

[2] J. Barry, L. P. Kaelbling, and T. Lozano-Pérez. Hierarchical solution of large markov decision processes. Technical report, MIT, 2010.

[3] 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.

[4] D. P. Bertsekas. *Dynamic programming and optimal control*, volume 1,2. Athena Scientific Belmont, MA, 2005.

[5] 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.

[6] 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.

[7] 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.

[8] P. Dai, Mausam, and D. S. Weld. Focused topological value iteration. In *International Conference on Automated Planning and Scheduling*, 2009.

[9] P. Dai, Mausam, D.S. Weld, and J. Goldsmith. Topological value iteration algorithms. *Journal of Artificial Intelligence Research*, 42(1):181–209, 2011.

[10] T. G. Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *arXiv preprint cs/9905014*, 1999.

[11] 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.

[12] 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.

[13] 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.

[14] S. M. LaValle. *Planning algorithms*. Cambridge university press, 2006.

[15] T. M. Moldovan and P. Abbeel. Risk aversion in markov decision processes via near optimal Chernoff bounds. In *NIPS*, pages 3140–3148, 2012.

[16] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. MIT Press, 2006.