Hierarchical Decision Theoretic Planning for Navigation Among Movable Obstacles

Martin Levihn, Jonathan Scholz and Mike Stilman

Abstract In this paper we present the first decision theoretic planner for the problem of Navigation Among Movable Obstacles (NAMO). While efficient planners for NAMO exist, they are challenging to implement in practice due to the inherent uncertainty in both perception and control of real robots. Generalizing existing NAMO planners to nondeterministic domains is particularly difficult due to the sensitivity of MDP methods to task dimensionality. Our work addresses this challenge by combining ideas from Hierarchical Reinforcement Learning with Monte Carlo Tree Search, and results in an algorithm that can be used for fast online planning in uncertain environments. We evaluate our algorithm in simulation, and provide a theoretical argument for our results which suggest linear time complexity in the number of obstacles for typical environments.

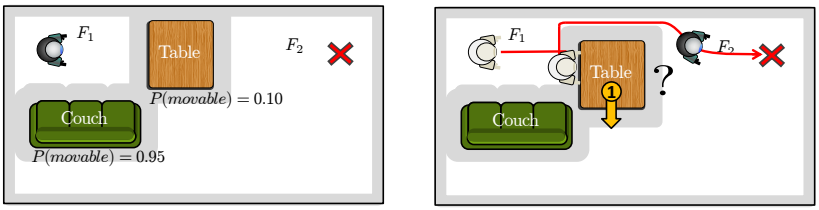
1 Introduction

There is great interest in robots that can safely navigate in common environments such as homes and offices. However, the presence of obstacles poses a serious challenge. Interacting with a single piece of clutter found in typical environments is difficult in itself, and the robot may need to manipulate many pieces of clutter to clear a goal safely. Even given a map of the room, how does the robot decide which path to take, or which object to move? This problem is referred to as Navigation Among Movable Obstacles (NAMO) [17, 18, 21]. NAMO is an important research area that is on the critical path to robot interaction in human environments.

There are two primary challenges in developing a practical algorithm for the NAMO domain: the exponential size of the search space and the inevitable inaccuracies in perception as well as actuation that occur on physical systems.

To understand the NAMO search space, consider navigating in a room with movable obstacles such as the ones depicted in Figure 1 and 2(a). If $\mathscr{C}_R = (x,y)$ represents the configuration of the robot base with resolution n in each direction of motion, then the number of possible robot configurations is $|\mathscr{C}_R| = O(n^2)$. This is true for each object as well, $|\mathscr{O}_i| = O(n^2)$. The full space of possible environment configurations is the product of these subspaces, $\mathscr{C}_R \times \mathscr{O}_1 \times \mathscr{O}_2 \times ... \times \mathscr{O}_N$, and therefore has $O(n^{2(N+1)})$ world states. In other words, the NAMO state-space is exponential in the number of objects it contains, with a base quadratic in map resolution.

Center for Robotics and Intelligent Machines, Georgia Institute of Technology, e-mail: {levihn, jkscholz,mstilman}@gatech.edu



(a) Configuration

(b) Solution obtained through a deterministic NAMO planner.

Fig. 1 The table wheels are likely to be locked, making it impossible for the robot to move the table. In contrast to deterministic planners, our proposed framework accounts for this probability.

Prior work on NAMO focused on handling the problem of *dimensionality*. It has not yet addressed the underlying issue of *uncertainty*. In reality, robots have noisy actuators and sensors, incomplete action models, and limited perceptual abilities.

To better understand why this might be a problem in an actual NAMO task, consider the example in Figure 1. Perhaps the robot knows that the shortest path to the goal involves moving the table, but it cannot see whether all the table wheels are unlocked. How might it weigh the costs of moving the table versus the couch? How would this answer be affected if it were given only a crude action model for manipulating the couch? These sorts of reasoning patterns are not expressible within the framework of deterministic search, without resorting to *ad hoc* heuristics.

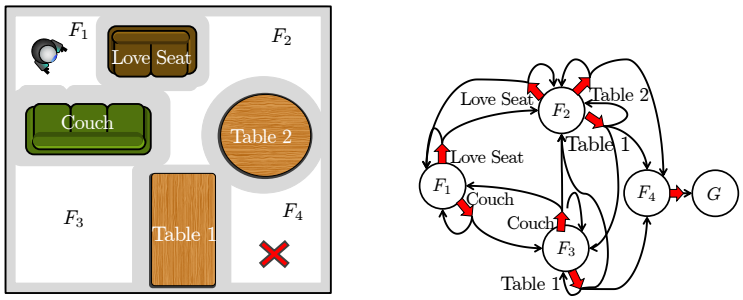
Leveraging ideas from decision theory, we have achieved a novel representation that formally addresses uncertainty in NAMO - it is useful, efficient and theoretically well-defined. By casting the NAMO problem as a hierarchical Markov Decision Process (MDP), we describe the first NAMO planner which can bias its decisions at *plan time* in order to compute policies that are *likely to succeed*.

This work is organized as follows: Section 2 gives an overview of the challenges of stochastic planning in NAMO, and our approach to addressing them. Following related work in Section 3, Section 4 reviews the relevant ideas in stochastic planning, which we combine to describe our algorithm in Section 5. Section 6 provides a theoretical and empirical evaluation of our results, and Section 7 concludes with final remarks.

2 Overview

Our general strategy to handling uncertainty in NAMO is to construct an "approximate" MDP which closely resembles the real problem, but can be solved in real-time. To achieve this, we draw on two techniques from Reinforcement Learning literature. First, we hierarchically decompose the overall task to *create shorter subtasks*, and second we apply an online Monte-Carlo technique to *solve only the relevant portion of the subtask*.

The NAMO problem has two properties that make it a good candidate for hierarchical planning. First, there is a natural abstraction from the low-level state space into a small set of *abstract states*, which we call "free-space regions" to indicate that the robot can move freely without collision. Second, there are a small number of im-



(a) Free space components (b) Resulting high-level MDP

Fig. 2 Concept visualization. The proposed framework defines a hierarchy of MDPs. The high level MDP operates with abstract states representing free spaces and abstract actions reflecting the notion of "creating an opening" by manipulating a specific obstacle.

plied *abstract actions* for maneuvering in this abstract state space. Since each free space region is circumscribed by obstacles, we can define the abstract action "create an opening to neighboring free-space" for each obstacle. These two properties are the necessary components of an *abstract MDP* (Section 4), which is the basis for our algorithm. Fig. 2(b) shows this representation for the environment visualized in Fig. 2(a).

A hierarchical framework by itself, however, is insufficient for creating a tractable NAMO MDP: even the individual manipulation tasks are too large to solve completely. For this reason we incorporate a complementary approach called Monte-Carlo Tree Search (Section 4.3). MCTS is a technique for generating a policy for one state at a time, with runtime that depends on the length of the plan rather than the number of states. MCTS is well-suited for the individual manipulation tasks for two reasons. First of all, the abstract MDP only requires a manipulation policy from a few starting locations of the obstacle, which are known at plan-time. Second, since the abstract MDP divided the NAMO problem into smaller manipulation tasks, the overall policy length for any subtask is quite small. These two properties allow us to substitute MCTS in place of value-iteration for our obstacle manipulation planner, without compromising the hierarchical optimality for the overall algorithm.

3 Related Work

The proposed framework lies at the intersection of search-based algorithms for the NAMO domain and stochastic planning algorithms for general domains with uncertainty. This section provides an overview of these two topics and their relationship to the proposed work.

3.1 NAMO Planning

Navigation and manipulation planning poses a significant computational challenge even with *complete environment information*. Wilfong [23] first proved that deterministic NAMO with an un-constrained number of obstacles is NP-hard. Demaine [3] further showed that even the simplified version of this problem, in which only unit square obstacles are considered, is also NP-hard.

In [18], Stilman presented a planner that solved a subclass of NAMO problems termed LP_1 where disconnected components of free space could be connected independently by moving a single obstacle. The planner was able to solve the hard problems presented in [2] and was successfully implemented on the humanoid robot HRP-2 [17]. Our state space decomposition is mainly based on the concept of "free-space regions" introduced in [18]. However, the free-space concept in [18] was simply a rhetorical device, and has never actually been used in a NAMO algorithm. The work presented here takes direct advantage of the free-space representation. Subsequent work presented a probabilistically complete algorithm for NAMO domains [21]. However, all these methods solved NAMO assuming perfect knowledge of the environment and deterministic action outcomes.

Wu [7] and Kakiuchi [9] introduced the first extensions to NAMO in *Unknown Environments*. In [7] a planner was presented that could solve NAMO problems given only partial obstacle information. However, that planner is not capable of handling uncertainty, and instead assumes that everything within sensor range is ground truth. Further, actions are again assumed to be deterministic. [9] presented a system that executes NAMO in unknown environments on the humanoid robot HRP-2. However, the authors took a reactive behavior-based approach, rather than attempting full decision theoretic planning.

3.2 Decision Theoretic Planning

As discussed in the previous section, there are two basic forms of uncertainty that may affect a NAMO planner: uncertainty in the *action outcome*, and uncertainty in the world *state*. In the decision theory planning literature, these types of uncertainty are typically modeled in different ways. The first is captured by a *probabilistic action model*, and is the basis for the Markov Decision Process (MDP). The second requires augmenting the MDP with a *probabilistic observation model*, into the so-called Partially Observable MDPs (POMDP).

POMDPs have been applied separately to *navigation* planning by Koenig and Pineau [12] [14] and *grasping manipulation* by Hsiao [6]. While both domains are related to NAMO, they focus on the configuration space of a single robot or a single object. In contrast, the NAMO domain requires the robot to reason about the full set of objects in its workspace. Existing robot planners that use POMDPs are generally restricted to low-dimensional configuration spaces. This constraint holds even when applying the most recent approximate POMDP solvers such as point-based value iteration [14] and belief compression [15].

Although explicit belief-space planning with the POMDP model offers some theoretical advantages, it is not strictly required in order to handle perceptual uncertainty. In POMDPs, entropy from both the action and observation models ultimately manifests as stochasticity in the feedback the agent gets from the environment. Thus, it is always possible to collapse any POMDP into the simpler MDP rolling the observation uncertainty into the transition model. We take this approach in this paper for the sake of computational efficiency.

Several techniques have been developed for extending the MDP model to hierarchical tasks. Among the most well known include the options framework [19], hierarchies of abstract machines (HAM)[13], and the MAX-Q framework [4]. All three

of these approaches rely on a generalization of the MDP to incorporate non-atomic, or "semi-markov" actions. The resulting model is referred to as a semi-markov decision process, or SMDP [5].

The primary difference between these approaches is whether they involve *simplifying* or *augmenting* the original MDP. The goal of the options framework is to introduce abstract actions without compromising the finer-grained planning of the original MDP. Therefore options-planning does not offer any *representational* abstraction: all planning is done in the state space of the original MDP. In HAM, the system designer specifies a hierarchical collection of state machines for solving subtasks. This state machine presents an abstract interface for the high-level task, which can again be solved as an SMDP. The drawback of these approaches which makes them inappropriate for NAMO is that the resulting SMDPs are either too low-level (options), or too high-level (HAM).

The third approach, MAX-Q, strikes a balance between Options and HAM. It is well-suited for NAMO because it does not require pre-defined subtask policies, but still utilizes an abstract planning representation. However, MAX-Q still assumes that each subtask is solvable using standard *dynamic programming* methods [4]. These algorithms are based on the theoretical results for the SMDP, which scales at best linearly in the size of the (non-abstract) state space [8]. This prohibits a direct application of the MAX-Q framework to the NAMO domain, and is the main technical challenge of this paper.

This challenge is addressed using an alternative to dynamic programming for solving MDPs, referred to as "sparse sampling" or "monte-carlo tree search" (MCTS). The MCTS literature describes a family of algorithms which were introduced to provide MDP solvers which scale *independently of the size of state space* [10]. However, MCTS remains exponential in the depth of its search tree, which for NAMO can often require tens or hundreds of primitive actions. Several heuristics have been developed for MCTS, including UCT [11] and FSSS [22], which are significantly more sample efficient than vanilla MCTS. While these algorithms are good candidates for a subtask planner, they can not solve the overall NAMO problem due to the large depth, branching factor, and sparsity of rewards in the core MDP.

4 Preliminaries

This section highlights the concepts underlying our approach. Following the definition of an MDP, both MAX-Q and MCTS will be explained. While substantially different, MAX-Q and MCTS are techniques developed for handling large state spaces in an MDP. To our knowledge, these techniques have never been combined. Consequently this section will treat them separately while the following sections will demonstrate how they can be efficiently combined to achieve a practical planner.

4.1 Markov Decision Processes

The Markov Decision Process (MDP) is a model for stochastic planning, and the foundation of our algorithm. We define an MDP $M = (S, A, T^a_{ss'}, R^a_s, \gamma)$ for a finite set of states S, a finite set of actions A, a transition model $T^a_{ss'} = P(s'|s,a)$ specifying the probability of reaching state s' by taking action a in state s, a reward model

$R_s^a = r(s,a)$ specifying the immediate reward received when taking action a in state s, and the discount factor $0 \leq \gamma < 1$.

The standard technique for solving MDPs is *Value Iteration* [16], which is used to find the optimal policy $\pi^* : f(s) \rightarrow a$ which maps states to actions in order to maximize the expected long-term reward, or value $V(s)$ for all states $s \in S$. Generic value iteration (VI) has a polynomial runtime in the number of states (with fixed error ε).

4.2 MAX-Q Value Function Decomposition

The MAX-Q framework is a technique within the reinforcement learning literature which describes how a *value function* for an MDP may be decomposed according to a *task hierarchy*. To understand this idea, consider a two-level hierarchy composed of a high-level policy π_0, defined over a set of subtask policies $\pi_i, i \in \{1,n\}$, which are in turn defined over primitive actions. That is, $\pi_0 : f(s) \rightarrow \pi_i$, and $\pi_i : f(s) \rightarrow a$.

The key insight behind MAX-Q is that the value functions for the subtask policies $V_{\pi_i}(s)$ contain all the information needed to represent the value function of the parent policy $V_{\pi_0}(s)$. This is true because, according to the Bellman equation for SMDPs, the value of executing subtask π_i in state s is simply the sum of the (discounted) reward accumulated by π_i itself, and the future expected reward of whichever state π_i terminates in:

$$Q(s,\pi_i) = R(s,\pi_i) + \sum_{s',\tau} P(s',\tau|s,\pi_i)\gamma^\tau V(s') \tag{1}$$

where γ^τ ensures that the value of s' is appropriately discounted according to the time τ that π_i took to terminate [1].

The first term in this expression, $R(s,\pi_i)$, is precisely the information encoded by $V_{\pi_i}(s)$:

$$R(s,\pi_i) = V_{\pi_i}(s) = \mathbb{E}_{\pi_i}\left[\sum_{t=1}^{\tau} \gamma^t R(s,a)\right] \tag{2}$$

Therefore, planning in the high-level task simply involves using the values of the subtasks as immediate rewards for their execution in the high-level policy.

In addition to an analysis of compositional value functions, the full MAX-Q framework also describes several model-free learning algorithms. In this paper, however, we are instead interested in model-based planning, since we assume the robot has knowledge of the effects of its actions. Our algorithm therefore differs in some important details, but shares the primary data structure (a hierarchical Q-function) and general format of "MAX-QQ" [4].

4.2.1 Types of optimality

So far, we have described a *bottom-up* representation of value functions, in which values of subtasks are "projected up" to parent tasks. This approach provides space efficiency, as well as the opportunity to divide and conquer: each subtask can be learned separately and combined to learn the parent task. Dietterich [4] refers to the class of policies which can be represented by such a model as *recursively optimal*, meaning all policies are optimal given optimal solutions to their subtasks.

The drawback to this approach, which makes it inappropriate for the NAMO domain, is that subtasks are learned without knowing their context in the overall plan. This is a problem, for example, if moving two obstacles provides the same reward, but only one of them opens a path to the goal. In general, tasks with sparse rewards tend to require either additional *shaping rewards* for the subtasks, or a solution strategy that includes *top-down* information [1].

We take the top-down approach by using the value of the target free-space region as a reward for the manipulation policy that clears the appropriate obstacle. This is equivalent to Dietterich's solution for the model-free case, in which the completion functions (the right-most term of Eq. 1) are treated as subtask terminal state rewards [4]. Policies that are learned in this fashion are referred to as *hierarchically optimal*, meaning that the overall policy is optimal given the constraints of the imposed hierarchy [4].

4.3 Monte-Carlo Tree Search

MCTS was invented as a way of allowing near-optimal planning for MDPs with large or infinite state spaces. The main idea is to relax the goal of computing a full policy, and instead focus on computing the optimal policy for a single state – the state the agent is in. In their original work on *sparse sampling*, Kearns et al. showed that it was possible to obtain ε-optimal Q-value estimates for the current state from a set of sampled transitions, and that the number of samples C per state was independent of $|S|$ [10].

MCTS works by building a search tree from the current state, selecting actions according to some search policy π_s, and sampling transitions from the transition model $T^a_{ss'}$ for each action. This tree generates a set of sampled rewards, which can be backed up to the root node to obtain a Q estimate according to:

$$Q^d_{SS'}(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q^{d-1}_{SS'}(s',a') \tag{3}$$

$Q^d_{SS'}(s,a)$ refers to the expected value of taking action a in state s, and following the optimal policy for $d-1$ subsequent steps (so $Q^1_{SS'}(s,a) = R(s,a)$). Equation 3 defines a simple recursion for using forward simulation to select actions, despite uncertainty in the action model.

4.3.1 The effective horizon

There is one additional result from the MCTS literature, regarding search depth, which we exploit in our hierarchical NAMO algorithm. Kearns et al. proved in [10] that MCTS could achieve ε-optimality with an $O((|A|C)^H)$ running time, where H is the *effective horizon* of the problem. Based on the fact that rewards in the distant future have little effect on the Q-values of the current state, this proof bounds H according to ε and R_{max} (the maximum possible reward):

$$H = \lceil log_\gamma(\frac{\varepsilon(1-\gamma)^2}{4V_{max}})\rceil, V_{max} = R_{max}/(1-\gamma) \tag{4}$$

Equation 4 states that the effective horizon *increases* with the value of the maximum possible reward that can be achieved. As we show in Section 5, this relation can be exploited in a hierarchical setting to have the MCTS manipulation planners spend time in proportion to the value of their target free-space region.

Neither MAX-Q nor MCTS alone are appropriate for the NAMO domain. The applicability of MAX-Q is hindered by requiring an exact solution for each subtask. MCTS in turn is limited by the branching factor and sparsity of rewards in the NAMO domain. In the next section we present a new algorithm that solves these shortcomings by combining the two approaches.

5 Algorithm

In this section we outline how to construct an MDP for the NAMO problem, and how to solve it by combining the tools from MAX-Q and MCTS described above. Recall that a standard MDP has an action model associated with each state. For the NAMO domain, this implies a *displacement model* for each object, which depends on the action a being executed, and the target obstacle o:

$$\delta x, \delta y \sim P(\delta x, \delta y | a, o) \tag{5}$$

In our case, however, we represent action models for a discrete set C of *object categories*. This allows us to incorporate observation uncertainty. Now instead of knowing the action model for an obstacle with certainty, the robot has a belief distribution $P(c|o)$ over categories of action models. The actual displacement model is consequently the marginal over the categories:

$$\delta x, \delta y \sim \sum_c P(\delta x, \delta y | a, c) P(c|o) \tag{6}$$

$P(c|o)$ can be used to encode any uncertainty the robot might have about the category of object it is manipulating. For example, $P(c_{ut}|o)$ could be the posterior probability of obstacle o being an unlocked table c_{ut} given some sensor data D: $P(c_{ut}|D) \propto P(D|c_{ut})P(c_{ut})$.

Our algorithm (Algorithm 1), therefore takes the following input:

1. The set O of obstacles present in the workspace
2. Distributions $P(c_i|o_j)$ representing the probability of obstacle o_j belonging to category c_i
3. Motion models indicating 2D object displacements, $P(\delta x, \delta y | a_{ll}^1, c_1) \ldots P(\delta x, \delta y | a_{ll}^k, c_m)$, indexed by action and object category
4. $\mathscr{C}_{goal}$ The robot's goal configuration in the workspace

It outputs:

1. A high-level policy π_0 indicating which obstacle to move for each free space
2. A set of low-level partial policies Π_1 indicating the manipulation actions to execute for each obstacle

5.1 The NAMO MDP

The proposed NAMO MDP has a two-level hierarchy, with navigation between regions as the high-level task, and object manipulation as the low-level task. Here we define both MDPs, and their hierarchical semantics. Recall that an MDP is defined as $M = (S, A, T^a_{ss'}, R^a_s, \gamma)$, which leaves four properties to define at each level of the hierarchy (γ can be viewed as a parameter of the algorithm).

States and Actions: The fundamental state space in the NAMO problem is the set of possible configurations $\mathscr{C}_W$ of the robot and environment. We define the low-level MDP M_{ll} in terms of these states S_{ll}, and a set of primitive actions A_{ll} for manipulating obstacles on the map. For simplicity, we used axis-aligned manipulations as action primitives in our simulation, but in practice these would typically be replaced with a more appropriate choice for physical systems, such as those introduced in [18].

For the high-level MDP M_{hl}, we define the state space S_{hl} to be the set of free-space regions implied by the starting obstacle configuration. In our implementation we identify these regions by performing seeded wavefront expansions on $\mathscr{C}_W$ to determine disjoint regions. During this operation we also save the list of obstacles bounding each free space region into an "adjacency list" $\mathscr{L}_r$, since these represent the possible manipulation targets for that state. The action space A_{hl} is the union of all possible manipulation sub-tasks for each region, where sub-task $a^{ijk}_{hl} = \pi_{ijk}$ means "open a path from state i to state j by manipulating obstacle k". The set of possible obstacles k for each starting state i are obtained from the adjacency list.

Transitions and Rewards: Following the discussion of MAXQ in Section 4.2.1, the rewards in M_{ll} and M_{hl} are defined to reflect their hierarchical relationship. Values for R_{hl} represent expectations over the reward accumulated by subtasks in A_{hl} (Eq. 2), and individual subtasks a_i receive the value of their final state in M_{hl} as a terminal reward (Section 4.2.1). This should be intuitive, since the high-level policy needs to know the actual outcome of executing each possible subtask, and each subtask needs to know its *context* in the high-level policy in order to know how important different obstacles are. Initially, only the state $s \in S_{hl}$ containing the goal configuration has a reward, set according to the robot's utility function.

The transition probabilities in M_{hl} directly represent the expected outcome of executing a subtask π_{ijk} in some state s^i_{hl}. For example, the manipulation sub-task π_{ijk} terminates in state j if it successfully opens a path from i to j, and terminates in state i otherwise. Therefore, the transition probability $P(s' = j | s = i, \pi_{ijk})$ is 1 if and only if π_{ijk} terminates in j. In addition, this suggests that transition model T_{hl} is sparse: the probabilities are automatically zero for all states that do not share an obstacle in their adjacency lists. Finally, the transition model for the low-level MDP T_{ll} is sampled directly from the displacement model (Eq. 6), encoding the domain uncertainty. This completes the construction of the NAMO MDP.

Note that this construction is an instance of what Dietterich refers to as a funnel abstraction [4]: the value all possible robot configurations (positions) within the target free space get mapped to a single value: the value of that region. This is the basic abstraction from which the hierarchical MDP obtains its savings.

Input: O: Obstacles, $P(c_1|o_1)\dots P(c_m|o_n)$: Distributions over categories,
$P(\delta x,\delta y|a^1_{ll},c_1)\dots P(\delta x,\delta y|a^p_{ll},c_m)$: motion models estimates for action primitives and categories, $\mathscr{C}_{goal}$: Goal Configuration
Output: π_0: High Level Policy, Π_1: Set of Low Level Policies

$(F,\mathscr{L}_r) \leftarrow$ get_freespaces(O);
$M_{hl}(S_{hl},A_{hl},T_{hl},R_{hl}) \leftarrow (F,\{\},\mathbf{[0]},\mathbf{[0]})$;
$\pi_0 \leftarrow \emptyset$; $\Pi_1 \leftarrow \emptyset$;
`// determine high level MDP definition:`
foreach $f_i \in F$ **do**
  if f_i *contains* $\mathscr{C}_{goal}$ **then**
    $\forall a\ R[f_i,a] \Leftarrow$ utility of reaching the goal;
  end
  foreach $o_k \in \mathscr{L}_r$ *adjacent to* f_i **do**
    foreach $f_j \in F$ **do**
      if o_k *adjacent to* f_j **then**
        $A_{hl} \Leftarrow A_{hl} \cup \{a^{ijk}_{hl}\}$;
        $T[s^i_{hl},s^j_{hl},a^{ijk}_{hl}] \Leftarrow 0.5$; `// connectivity; actual uncertainty encoded at lower level, adds to 1 due to self transition for failure case`
      end
    end
  end
end
`// run value iteration:`
while *error not within* ε **do**
  foreach $s^i_{hl} \in S_{hl}$ **do**
    $v \Leftarrow 0$;
    foreach $a^{ijk}_{hl} \in A_{hl}$ **do**
      $h \Leftarrow \lceil log_\gamma(\frac{(\varepsilon(1-\gamma)^2)/4}{s^j_{hl}.v}\rceil)$;`// dynamic horizon`
      $(q_k,\pi_{ijk}) \Leftarrow$MCTS$(a^{ijk}_{hl},h,\sum_c P(\delta x,\delta y|a^1_{ll},c)P(c|o_k),\dots,\sum_c P(\delta x,\delta y|a^p_{ll},c)P(c|o_k)$;
        `//` a^{ijk}_{hl} `provides necessary information about obstacle and free-spaces to connect`
      if $q_k > v$ **then**
        $v \Leftarrow q_k$;
        $\pi_0(s^i_{hl}) \Leftarrow a^{ijk}_{hl}$; $\Pi_l(o_k) \Leftarrow \pi_{ijk}$;
      end
    end
    $s^i_{hl}.v \Leftarrow v$;
  end
end
return π_h,Π_l;

Algorithm 1: Proposed framework.

5.2 Solution

Section 5.1 described a two-level hierarchy of MDPs defined in terms of each other: the rewards for M_{ll} were obtained from the values of states in M_{hl}, and the values of states in M_{hl} were defined based on outcomes of subtasks in M_{ll}. With this formula-

tion, values in both M_{ll} and M_{hl} reflect the true transition probabilities and rewards for manipulation actions.

This suggests an iterative scheme in which we alternate between updates to the high-level policy π_0 given the current values for subtasks $V_{\pi_{ijk}}$, and updates to the individual subtasks π_{ijk} given the values in V_{π_0}. However, computing these values requires actually solving the associated MDPs, which was shown to be intractable for M_{ll} in Section 1, since $S_{ll} = \mathscr{C}_W$.

Fortunately, sparse-sampling planners provide a way of obtaining approximate solutions to M_{ll}, and make assumptions which are compatible with our hierarchical approach (Section 4.3). Therefore, our actual algorithm performs the same alternating policy-iteration scheme described above, but with the substitution of an MCTS planner in place of value-iteration for the manipulation MDPs. These MCTS planners compute Q-values in $\mathscr{C}_W \times A_{ll}$, and sample transitions from the displacement model defined in Eq. 6. Sampling terminates when an opening is achieved or the maximum search depth, defined by the horizon H (Eq. 4), is reached. For a general MCTS planner, H is defined as a function of R_{max}. In combination with a hierarchical policy iteration, however, it can be used to explicitly force the computation effort for each subtask planner to scale in proportion to its estimated utility. This is achieved using a *dynamic horizon*. Dynamic horizon recomputes the H-value, Eq. 4, of each MCTS planner *separately* based on the value of the target state in S_{hl}. The overall effect of this operation is that π_i does not waste time sampling actions past the point where rewards can significantly affect the Q-value of the subtask.

The following section will argue the usefulness of this complete framework.

6 Evaluation

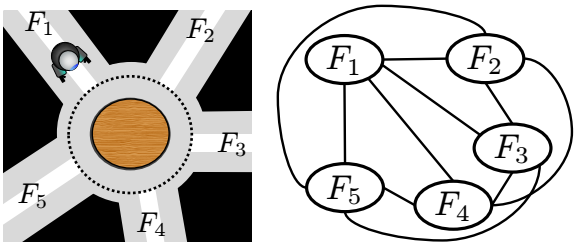
This section provides theoretical and empirical evidence that our algorithm has a run-time which is linear in the number of obstacles, for non-degenerate environments.

6.1 Theoretical Analysis

The following analysis is based on relating the sparsity of the free-space transition matrix T_{hl} to the *adjacency graph* G_A over free space regions. The sparsity of T_{hl} directly controls the complexity of value iteration.

Definition 1. The adjacency graph $G_A = (V,E)$ is defined to have vertices $v_i \in V$ which uniquely represent free space regions F_i, and edges $e(i,j) \in E$ connecting adjacent free space regions. That is, $e(i,j) \in E \Leftrightarrow adjacent(F_i,F_j)$, with $adjacent(F_i,F_j) = true$ iff F_i and F_j are disconnected through a single obstacle.

Figures 3 and 5 show examples of workspaces and their associated adjacency graphs. A graph G is planar if it can be drawn on the plane with no edges crossing except at common vertices. Kurarowski's theorem states that a graph is planar if and only if it does not contain a subgraph that is a subdivision of K_5 or $K_{3,3}$, where K_5 denotes a complete graph with five vertices, and $K_{3,3}$ denotes a complete bipartite graph on six vertices [20]. Note that G_A in typical environments will fulfill this definition. The contrary would require that either:



(a) Environment (b) Resulting adjacency graph G_A.

Fig. 3 Example of G_A not being planar.

a) A set of five free spaces are all adjacent to each other, separated by only a single obstacle.
b) A set of three free spaces are all adjacent to a disjoint set of three free spaces, but not adjacent to each other

Figure 3 shows one of these degenerate cases.

Further, recall that by definition of T_{hl}, the only next-states with non-zero transition probability from state s_{hl} are states representing free spaces adjacent to s_{hl} or s_{hl} itself (failure case). This gives rise to the following lemma:

Lemma 1. *T_{hl} has on average a constant number of next-states with non-zero probability if G_A is planar.*

Proof. First, it is trivially true that on average, rows of T_{hl} contain a constant number of self-transitions: $\frac{|S_{hl}|}{|S_{hl}|} = 1$. The remaining non-zero entries in T_{hl} are a direct mapping of $G_A = (V,E)$. The number of next-states with non-zero probability for s_{hl} is equal to the degree d of the associated vertex in G_A. This suggests that the average number of next-states with non-zero probability in T_{hl} is equal to the average degree of G_A.

We now bound the average degree of G_A to show that the number of non-self transitions in T_{hl} is at most 6. First, recall Euler's formula, which places a constraint on the number of possible edges e, vertices v, and faces f in a planar graph:

$$v - e + f = 2 \tag{7}$$

Now consider the set of all possible "edge-face pairs" $p \in P$, (for $v > 2$) where an edge is added to P for each face in which it appears. Observe that since an edge can contribute to at most 2 faces, we have $|P| \leq 2e$. In addition, each face must have at least 3 edges, implying $|P| \geq 3f$. Plugging this into Eq. 7, we have $e \leq 3v - 6$. We can now compute the average degree d_{avg} of G_A:

$$d_{avg} = \frac{d_{total}}{\#vertices} = \frac{\sum_{i=1}^{v} d_i}{v} = \frac{2e}{v} \leq \frac{2(3v-6)}{v} = 6 - \frac{12}{v} < 6 \tag{8}$$

Consequently, T_{hl} has on average at most 6 next-states with non-zero probability. □

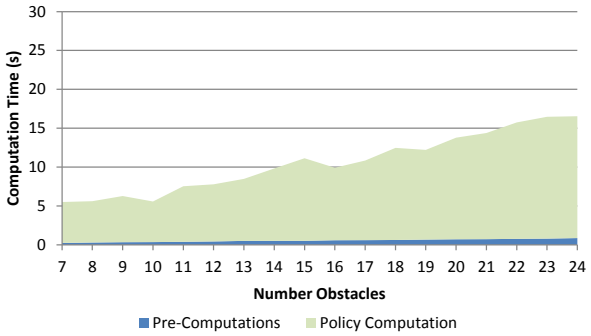


Fig. 4 Obtained average computation times as a function of the number of obstacles. The graph suggest linear complexity in the number of obstacles.

Lemma 2. *The run-time of the proposed framework is linear in the number of obstacles $|O|$ if the adjacency graph G_A is planar.*

Proof. First, consider M_{hl}. The number of states $|S_{hl}|$ is linear in $|O|$ (worst case $2|O|$ for infinite obstacles intersecting at a star pattern). Value iteration is performed over M_{hl}. Since the error ε is a free parameter, the complexity of value iteration is typically separated into the cost-per-iteration and the expected number of iterations. The number of iterations required by VI to achieve an ε-error bound is $N = \lceil log(2R_{max}/\varepsilon(1-\gamma))/log(1-\gamma)\rceil$ [16], which is independent of S_{hl}. The inner loop of value iteration is quadratic in $|S|$ the worst case, due to the need to compute an expectation over the entire state-space in each Bellman update [8]. However, since T_{hl} has on average a constant number of next-states with non-zero probability for planar G_A (Lemma 1), the expected per-iteration cost of value iteration reduces to $6 \times |S_{hl}|$, yielding an overall complexity for M_{hl} of $O(N|S_{hl}|)$.

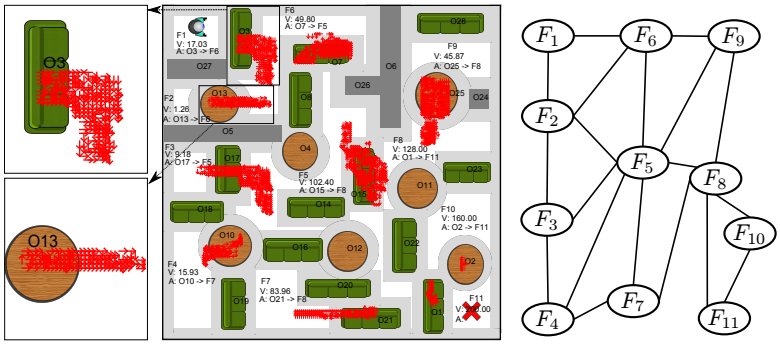
Finally, each action evaluation in M_{hl} requires MCTS sampling. As discussed in Section 4, the complexity of MCTS is independent of the state space size [10]. It is consequently constant over $|O|$, which implies that the overall algorithm is linear in $|O|$. □

6.2 Empirical Analysis

To evaluate the proposed framework we have analyzed it using the implementation discussed in Section 5. All the experiments were run on a Intel Core i7 (2.1GHz) PC.

6.2.1 Runtime

We have evaluated the framework on 1000 randomly generated NAMO environments. The size of the map was sampled uniformly to be between 250x250 and 400x400 grid cells. The number of obstacles for each map was uniformly sampled to be between 7 and 24, each obstacle in turn having random position, shape (rectangular or ellipsoid) and size (minimum 15x15 cells, maximum 65x65 cells occupation). Motion models $P(\delta x, \delta y | a_{ll}, c)$ were represented using Gaussians with their mean and standard deviation randomly varying for different categories. The category estimates $P(c|o)$ for each object were randomly assigned with the constraint of



(a) Obtained solution with zoom in on two local policies. (b) Adjacency graph.

Fig. 5 Example environment. V is the states value and A denotes the action to execute. E.g. the free space containing the robot has a value of 17.03 and the best action to execute is to manipulate obstacle $O3$ to gain access to free space $F6$. Low level policies visualized as a vector field. Static obstacles visualized in gray.

summing to 1 for each obstacle. The generated maps had an average of more than 70% cells occupied.

Figure 4 summarizes the computation time as a function of the number of obstacles. Pre-computations include the generation of the configuration space representation and determination of free space regions. While we show computation time as a function of number of obstacles, note that there are many other contributing factors, such as the particular configuration and number of free spaces, the complexity of planning to create openings etc. Figure 4 averages over these factors, and consequently resulted in a high standard deviation of up to 11.1s.

The empirical results are consistent with the theoretical analysis of our algorithm, and support applicability of the approach to practical environments.

6.2.2 Example

Fig. 5 shows an example environment and the solution obtained by our proposed framework. The high level policy is indicated in text, and the low level policy is visualized as a vector field for each obstacle. Only low level policies corresponding to obstacles that are chosen by the high level policy are visualized to preserve a clear view.

6.3 Online Execution

If the robot successfully executes a low-level policy in its workspace, it, in general, has *merged* two free spaces rather than transitioned between them. This alters the state-space of M_{hl}. While future work will investigate the applicability of policy iteration [16] to take advantage of the locality of the change, we address this by re-executing the proposed algorithm after each object manipulation. Given the linear runtime of the algorithm, this has not presented a significant disadvantage. However, as the algorithm does not consider all possible future free space configurations, the Q-values for a specific M_{hl} do not represent the true long term expected reward of those regions, but rather an approximation based on the current world configuration.

In general, this approximation is sound as long as actions are reversible. Future work will examine the expected loss and convergence properties under this assumption.

7 Conclusion

We have presented the first decision theoretic planner for the problem of Navigation Among Movable Obstacles. This planner is able to overcome the exponential complexity of the NAMO domain, while incorporating uncertainty, by employing a novel combination of ideas from sparse-sampling and hierarchical reinforcement learning.

Our algorithm can be viewed from the top-down as a value-iteration scheme in the free-space representation of the NAMO problem. From this perspective, the primary difference with classical value iteration is that the Bellman updates issue calls to a low-level MCTS planner for evaluating the action rewards, rather than querying an atomic reward function. In this fashion, values spread through the free-space MDP as they typically would in value iteration. From a bottom-up perspective, we can also view the role of the free-space MDP as a data structure for learning a shaping reward for manipulation subtasks. Thus, the high-level MDP over free space serves to constrain the set of possible actions (and hence the branching factor), as well as the depth of manipulation plans. We expect that this new perspective on the NAMO problem will generate additional insight into robot planning in human environments.

Acknowledgments

This research was supported by NSF grant IIS-1017076. The authors thank Kaushik Subramanian and Jacob Steinhart for many insightful discussions.

References

[1] A.G. Barto and S. Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 13(4):341–379, 2003.

[2] P. Chen and Y. Hwang. Practical path planning among movable obstacles. In *In Proceedings of the IEEE International Conference on Robotics and Automation*, pages 444–449, 1991.

[3] E. Demaine, J. O'Rourke, and M. L. Demaine. Pushpush and push-1 are np-hard in 2d. In *In Proceedings of the 12th Canadian Conference on Computational Geometry*, pages 211–219, 2000.

[4] T. Dietterich. An overview of maxq hierarchical reinforcement learning. *Abstraction, Reformulation, and Approximation*, pages 26–44, 2000.

[5] R.A. Howard. *Dynamic probabilistic systems*, volume 317. John Wiley & Sons New York, 1971.

[6] Kaijen Hsiao, Leslie Pack Kaelbling, and Toms Lozano-prez. Grasping pomdps. In *in Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 4685–4692, 2007.

[7] H.Wu, M. Levihn, and M. Stilman. Navigation among movable obstacles in unknown environments. In *IEEE/RSJ Int. Conf. On Intelligent Robots and Systems (IROS 10)*, October 2010.

[8] L.P. Kaelbling, M.L. Littman, and A.W. Moore. Reinforcement learning: A survey. *Arxiv preprint cs/9605103*, 1996.

[9] Y. Kakiuchi, R. Ueda, K. Kobayashi, K. Okada, and M. Inaba. Working with movable obstacles using on-line environment perception reconstruction using active sensing and color range sensor. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 1696 –1701, 2010.

[10] Michael Kearns, Yishay Mansour, and Andrew Y. Ng. A sparse sampling algorithm for near-optimal planning in large markov decision processes. *Machine Learning*, 49:193–208, 2002.

[11] L. Kocsis and C. Szepesvari. Bandit based monte-carlo planning. *Machine Learning: ECML 2006*, pages 282–293, 2006.

[12] Sven Koenig and Reid G. Simmons. Xavier: A robot navigation architecture based on partially observable markov decision process models. In *Artificial Intelligence Based Mobile Robotics: Case Studies of Successful Robot Systems*, pages 91–122. MIT Press, 1998.

[13] R. Parr and S. Russell. Reinforcement learning with hierarchies of machines. *Advances in neural information processing systems*, pages 1043–1049, 1998.

[14] Joelle Pineau, Geoff Gordon, and Sebastian Thrun. Point-based value iteration: An anytime algorithm for pomdps, 2003.

[15] Nicholas Roy, Geoffrey Gordon, and Sebastian Thrun. Finding approximate pomdp solutions through belief compression. Technical report, 2003.

[16] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition, 2009.

[17] M. Stilman, K. Nishiwaki, S. Kagami, and J. Kuffner. Planning and executing navigation among movable obstacles. In *IEEE/RSJ Int. Conf. On Intelligent Robots and Systems (IROS 06)*, pages 820 – 826, October 2006.

[18] Mike Stilman and James J. Kuffner. Navigation among movable obstacles: Real-time reasoning in complex environments. In *Journal of Humanoid Robotics*, pages 322–341, 2004.

[19] R.S. Sutton, D. Precup, and S. Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1):181–211, 1999.

[20] Norishige Chiba Takao Nishizeki. *Planar graphs: theory and algorithms*. Elsevier Science Ltd, 1988.

[21] J. Van Den Berg, M. Stilman, J. Kuffner, M. Lin, and D. Manocha. Path planning among movable obstacles: a probabilistically complete approach. *Workshop on Algorithmic Foundation of Robotics (WAFR VIII)*, pages 599–614, 2008.

[22] T.J. Walsh, S. Goschin, and M.L. Littman. Integrating sample-based planning and model-based reinforcement learning. In *Proceedings of AAAI*, number 1, 2010.

[23] G. Wilfong. Motion planning in the presence of movable obstacles. In *SCG '88: Proceedings of the fourth annual symposium on Computational geometry*, pages 279–288, New York, NY, USA, 1988. ACM.