Supplementary Material for "Task-Oriented Active Perception and Planning in Environments with Partially Known Semantics"

Mahsa Ghasemi [1] **Erdem Arinc Bulgur** [2] **Ufuk Topcu** [2]

Exact Verification of a Markov Chain with Partial Semantics

As stated in Theorem 1, the complexity of verifying a Markov chain with a probabilistic labeling function is exponential in both the number of states and the number of atomic propositions. This complexity is due to the fact that there may be $2^{|\mathcal{S}||\mathcal{AP}|}$ possible environment configurations that can not be handled simultaneously due to incompatibility of probabilities from the MDP and the DFA (as discussed in Example 1).

We now propose an algorithm for exact verification of a Markov chain with probabilistic labels against a reachability formula. The algorithm may further be extended to verification of general co-safe temporal logic formulas, by considering the product of the Markov chain with the corresponding DFA. The proposed algorithm relies on constructing a computation graph in a top-down manner and then, tracing back the reachability probabilities in a bottom-up method. The computation graph ensures that the truth assignments of the state properties are consistent in each of its trajectories. The computation graph is essentially a directed tree if one disregards the edges at the same level of the tree. Nodes of the graph accumulate reachability probabilities associated with their subgraph conditioned on the history captured by the trajectory from the root to that node. Edges of the graph show probabilistic branching either due to the stochastic transitions of the Markov chain or the uncertain transitions over the DFA resulting from uncertain perception. The next example explains the graph construction and reachability evaluation for a sample Markov chain.

Example 2 (Exact quantitative verification using a computation graph)**.** *Let the fully-connected Markov chain in Figure 5 represent an induced Markov chain over an MDP by a policy π. Moreover, let $\mathcal{AP} = \{p\}$ be the set of atomic*

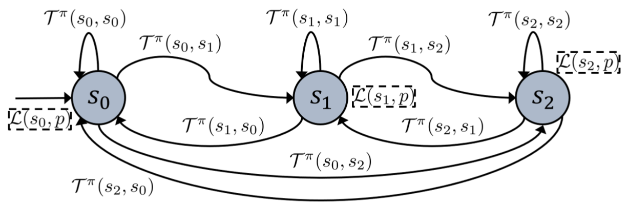

$\mathcal{T}^\pi(s_0, s_0)$
$\mathcal{T}^\pi(s_0, s_1)$
$\mathcal{T}^\pi(s_1, s_1)$
$\mathcal{T}^\pi(s_1, s_2)$
$\mathcal{T}^\pi(s_2, s_2)$
$\mathcal{L}(s_2, p)$
s_0
s_1
$\mathcal{L}(s_1, p)$
s_2
$\mathcal{L}(s_0, p)$
$\mathcal{T}^\pi(s_1, s_0)$
$\mathcal{T}^\pi(s_2, s_1)$
$\mathcal{T}^\pi(s_0, s_2)$
$\mathcal{T}^\pi(s_2, s_0)$

Figure 5. A 3-state fully-connected induced Markov chain with probabilistic labeling over property p.

propositions, and $\varphi = \Diamond p$ be the reachability specification. For a state s_i, the uncertain perception assigns a probability of $\mathcal{L}(s_i, p)$ for the property p to be true. Given this knowledge, the quantitative verification aims to compute the expected probability of realizing the specification, i.e., $\mathbb{E}_{\mathcal{L}\sim Dist(\mathcal{L})}\left[Pr\left(\mathcal{M}^\pi \models \varphi\right)\right]$.

To that end, we construct a computation graph, shown in Figure 6, that captures all the trajectories over the Markov chain as well as truth value assignments of the atomic propositions. The graph starts from a root node that is connected to a set of nodes representing the set of initial states (may be a singleton set). Then, each node branches into two possible truth assignments of property p, with corresponding probabilities as the labels of the outgoing edges. At this level, one truth assignment has taken place. If property p holds at the current state, the corresponding node satisfies the reachability formula. Otherwise, we must account for the next transition. Therefore, the node that assigns $False$ to p, branches into a number of nodes representing the possible next states on the Markov chain. In this example, since the Markov chain is fully-connected, each node will have 3 children. Such edges have a label that designates the transition probability dictated by the Markov chain. Again, different truth assignments are considered for each new node. However, notice that the node whose property already has a truth value will no longer branch. But instead, it may transition to other nodes at the same level with probabilities given by the Markov chain. This procedure is continued until the whole graph is constructed. In particular, the graph ends once all states have been assigned with truth values over the set of atomic propositions. A key factor in generating

[1]Department of Electrical and Computer Engineering, University of Texas at Austin [2]Department of Aerospace Engineering and Engineering Mechanics, University of Texas at Austin. Correspondence to: Mahsa Ghasemi <mahsa.ghasemi@utexas.edu>.

Proceedings of the 37th International Conference on Machine Learning, Online, PMLR 119, 2020. Copyright 2020 by the author(s).

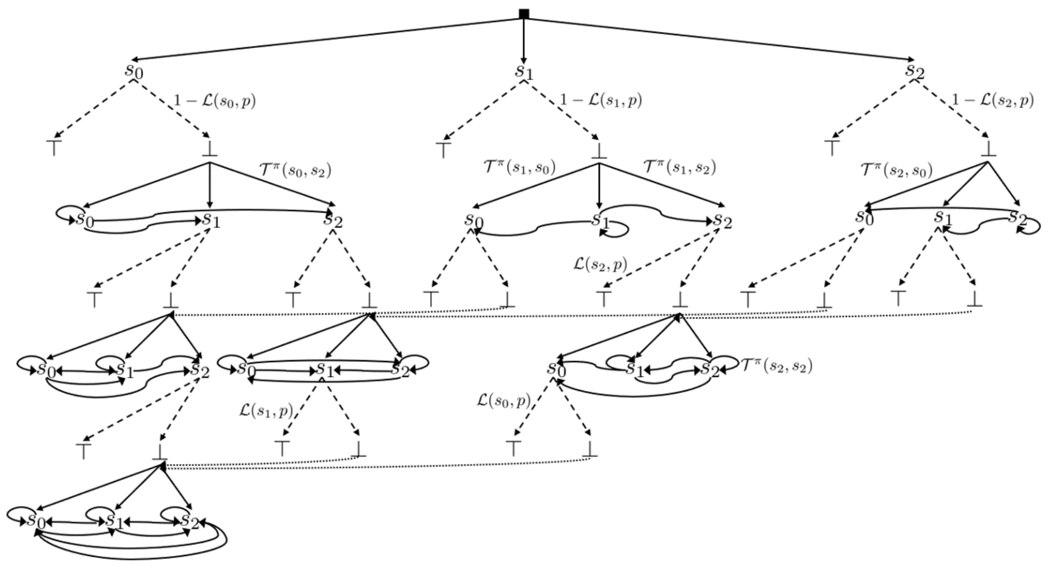


Figure 6. A computation graph for exact verification of a reachability specification $\varphi = \Diamond p$ over the Markov chain in Figure 5 that has a probabilistic labeling function. The branching of the nodes is either due to transition between the states (solid edges) or due to assigning a truth value to a property at a state (dashed edges). The truth values $True$ and $False$ are denoted by $\top$ and $\perp$, respectively. Each edge has a label (some of them shown on the graph) that represents the branching probability. The dotted edges connect a node to a subgraph that is identical to its future branchings.

the computation graph is that there are nodes for which the history of truth assignments up to them are equivalent. For instance, in one trajectory over the graph, first $s_0 \models \neg p$ and then $s_2 \models \neg p$ while on another trajectory, the assignments happen in reverse order. Such equivalent assignments happen for any possible permutation in the ordering of the states. Since the subgraph originating from these nodes are the same (except the initial labels for the branchings due to state transitions), we can avoid identical computation. In Figure 6, such similar nodes are connected with dotted edges.

Once the graph is generated, we start to compute reachability probabilities in a bottom-up manner. At the lowest level, all states have truth assignments and hence, their verification is reduced to solving a system of linear equations (Baier & Katoen, 2008). Once the reachability probability of children of a node are computed, the reachability probability of the node will be computed using the branching probabilities. Essentially, the parent node linearly combines the reachability probabilities of its children, weighted by the branching probabilities. Formally, for a node corresponding to a state n_s with a set of children $C(n_s) = \{n_{b'} = \top, n_{b'} = \perp\}$, the reachability probability is

$$Pr\left(n_s \models \Diamond p\right) = \mathcal{L}(s,p) + (1-\mathcal{L}(s,p))Pr\left((n_{b'} = \perp) \models \Diamond p\right),$$

and for a node corresponding to a truth value n_b of a state s, the reachability probability is

$$Pr(n_b \models \Diamond p) = \begin{cases} 1 & \text{if } n_b = \top, \\ \displaystyle\sum_{n_{s'} \in C(n_b)} \mathcal{T}^\pi(s,s') Pr\left(n_{s'} \models \Diamond p\right) & \text{if } n_b = \perp. \end{cases}$$

Once all levels of the graph are traversed, the reachability probability of the root node determines the expected probability of realizing the specification.

As opposed to naively generating all possible $2^{|\mathcal{S}||\mathcal{AP}|}$ environment configurations, the proposed algorithm is significantly advantageous. Since, for a deterministic labeling function, verification may lead to solving a linear system of $|\mathcal{S}|$ equations while in the computation graph, the number of equations may be $|\mathcal{S}|$ only at the last layer. More precisely, the i^{th} (state-based) layer of the graph may yield a linear equation system with up to i equations. The number of nodes in the computation graph depends on the Markov chain as well as the reachability specification. The highest number of nodes happen when the Markov chain is fully-connected with $|S| = n$ and the reachability specification is in the conjunctive form $\varphi = \Diamond \bigwedge_{j=1}^{m} p_j$, leading to $O(n2^{nm})$ nodes.

Remark 1. *The proposed statistical verification approach only samples a subset of environment configurations as opposed to the exact method that exhaustively accounts for all possible configurations. If verification of a specification* *over a finite time horizon is desired, instead of applying an exact verification on each sampled environment, one may also sample trajectories from the Markov chain. This approach leads to a Markov chain Monte Carlo strategy over the computation graph. More specifically, each sample will be a trajectory with length equal to the time horizon over the computation graph.*

Details of Active Perception Strategy

In settings where an agent has a single shot at performing a task, the safety of an exploration strategy is crucial. Safe exploration not only depends on the characteristics of the model (here, MDP) but also the properties of the given task. A typical strong assumption for ensuring safety in exploration is ergodicity of the MDP (Moldovan & Abbeel, 2012). If an MDP is ergodic, then the agent will always be able to find a policy to return to a previously visited state after an exploratory action. For a temporal logic task, this ergodicity assumption should hold on the product of the MDP and the DFA corresponding to the task. Nevertheless, the ergodicity assumption does not usually hold in a realistic setting.

The proposed active perception strategy (detailed in Algorithm 1 of the main manuscript) balances the exploration safety with the information quality by a hyperparameter β. In particular, the algorithm locally searches for a state on the product of the MDP and the DFA to which it can go and come back with high probability. Similar to an ergodicity assumption, this requirement ensures that the agent can return to its current state (of the MDP and the task). On the other hand, the exploratory actions must lead to gathering information that reduces the uncertainty in the environment semantics.

To generate the active perception strategy, Algorithm 1 generates a tree of possible sequences of actions with a bounded length. Each node of the tree captures the exploration safety by a reachability metric and the information quality by an entropy reduction metric over the belief. The reachability metric depends on the probability of being in an MDP state b_k^s, the probability of being in a task state b_k^q, as well as the reachability probability r_k back to the state from which the active perception strategy is initiated. The belief over the MDP states originates from the transition probabilities of the MDP, i.e., for an action $a \in \mathcal{A}$,

$$b_k^s(s') = \sum_{s \in \mathcal{S}} b_k^s(s) \mathcal{T}(s, a, s').$$

The belief over the DFA states depends on the belief over the MDP states and the belief over the atomic propositions, i.e., for an action $a \in \mathcal{A}$,

$$\begin{aligned} b_k^q(q') = &\sum_{s \in \mathcal{S}} b_k^s(s) \sum_{s' \in \mathcal{S}} \mathcal{T}(s, a, s') \\ &\sum_{P \subseteq \mathcal{AP}} \mathcal{L}_t(s', P) \sum_{q \in \mathcal{Q}} \mathbb{1}[\delta(q, P) = q'], \end{aligned}$$

where $\mathbb{1}[.]$ is the indicator function. It is worth noting that instead of explicitly enumerating all $P \subseteq \mathcal{AP}$, it suffices to only find the activation probabilities of transitions in the DFA. Each transition has a propositional logic formula whose truth assignment can be evaluated using the following rules recursively:

$$\begin{aligned} &Pr(s \models p) = \mathcal{L}(s, p), \\ &Pr(s \models \neg p) = 1 - \mathcal{L}(s, p), \\ &Pr(s \models p_1 \wedge p_2) = \mathcal{L}(s, p_1)\mathcal{L}(s, p_2), \\ &Pr(s \models p_1 \vee p_2) = \mathcal{L}(s, p_1) + \mathcal{L}(s, p_2) \\ &\qquad\qquad\qquad\qquad - \mathcal{L}(s, p_1)\mathcal{L}(s, p_2). \end{aligned}$$

The information quality of a node is measured by the amount of entropy reduction Δe_k of the belief distribution on the atomic propositions. Given an observation model and the current belief, it is straightforward to compute the expected entropy reduction. The proposed active perception strategy will keep reducing the uncertainty over the environment semantics. Hence, if the ergodicity assumption holds on the product MDP, then the agent will be able to find the optimal policy.

Simulation Details

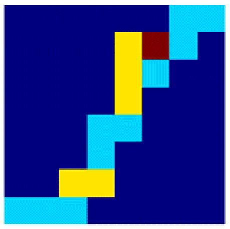
We provide the details of the simulation settings as well as further results next.

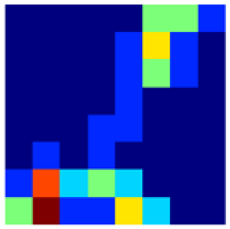
Planar Navigation with Finite-Horizon Tasks

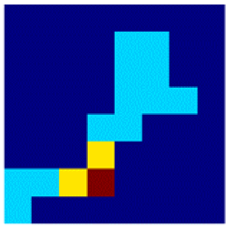
We simulated a planar navigation in a discretized 2D environment of size 8×8. We implemented the proposed task-oriented perception and planning algorithm and its different variations without certain blocks such as belief update, divergence test, and active perception. The simulations for planar navigation were run on a laptop with 2.0 GHz Intel Core i7-4510U CPU and 8 GB RAM. The Python codes are provided in the following repository: https://github.com/MahsaGhasemi/task-oriented-perception-and-planning

The results of 50 runs of each algorithm is averaged and represented in Table 1 of the main manuscript. The success rate is the average of how many runs the agent successfully completed the task. The number of steps indicates how many actions the agent has taken to either complete the task or fail it. The failure of the task could either be due to exceeding the maximum time step of 50 or due to hitting an obstacle. The number of (re)planning is reported in the last column. If the divergence test is included in the algorithm, the agent performs fewer policy synthesis, however, it also takes longer to complete the task and has smaller success rate. The results show that adding an active perception strategy increases the success rate whether replanning happens at every state or only when required by the divergence test.

The proposed task-oriented perception and planning algorithm has two hyperparameters that affect its performance. The hyperparameter γ_d determines how frequent the replanning must occur. Lower values of γ_d impose a higher amount of computation as the agent should synthesize a policy more often. On the other hand, higher values of γ_d may increase the risk of following a potentially poor plan obtained from poor perception. A relatively conservative choice for γ_d is the minimum value that the divergence measure will change if the truth value of a state's property is flipped. The hyperparameter γ_r determines the willingness of the agent to risk, i.e., follow a policy whose success is considerably changed when perception uncertainties are taken into account. The choice of γ_r depends on the application, more specifically how much cost a wrong or unsafe action may incur. Figure 7 demonstrates the effect of risk threshold on an agent's behavior. The agent starts from the bottom-left cell and aims to safely navigate through the environment to reach the top-right cell. The paths depicted in (a) and (b) are both successful, however, in (b) the risk threshold is lower. As a result of that, the agent performs more information-gathering strategies. As shown in (c) and (d), an improper choice of the the risk threshold may lead to task failure. If γ_r is too high, the agent may take a bad action and arrive at a state with an obstacle. On the other hand, if γ_r is too low, the agent keeps gathering information until the maximum number of time steps is exceeded.


(a) A successful path with high risk threshold.


(b) A successful path with low risk threshold.


(c) An unsuccessful path due to very high risk threshold.

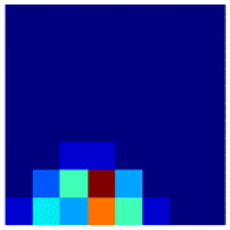

(d) An unsuccessful path due to very low risk threshold.

Figure 7. Samples of different paths of an agent in an 8×8 2D environment under different risk thresholds. The agent's goal is to reach the top-right cell from the initial bottom-left cell. The colors indicate the normalized frequency of visiting each state where dark blue means no-visit and dark red means highest number of visits.

Drone Navigation in Simulated Urban Environment

We used the open-source platform AirSim (Shah et al., 2017) to evaluate the proposed task-oriented perception and planning algorithm on a drone flying through a simulated urban environment. AirSim can be integrated with any environment created on the Unreal Engine. We used version 4.18 of Unreal Engine. We developed an urban environment, partially depicted in Figure 8, using the commercial Unreal Engine package DownTown created by PolyPixel. All AirSim simulations were run on a machine with 3.50 GHz Intel Core i9-9920X CPU, GeForce RTX 2080 Super GPU, and 128 GB RAM. The Python codes and the setting file (JSON format) for AirSim are provided in in the following repository: https://github.com/MahsaGhasemi/task-oriented-perception-and-planning

We modeled a drone flying in this environment. The drone's task is to reach to a target building with a flag on top of it while avoiding the other buildings in its surroundings. We control the drone through a PX4 flight controller. The drone flies in a point-to-point manner with constant speed between centers of the cells of a discretized map of the environment. The map is 120×185 m^2 and consists of 24×37 cells of size 5×5 m^2. The drone's perception relies on two types of data gathered by 4 RGB cameras (with the resolution of 128×128 pixels) and 4 depth cameras (type DepthPlanner in AirSim). Each camera has a $90°$ field of view so that the 4 of them cover the whole $360°$. Whenever the drone arrives at a cell, the perception modules processes the current data through the procedure shown in Figure 9. In particular, using a projection matrix for the depth data, we create a point cloud, labeling each pixel with its corresponding coordinates

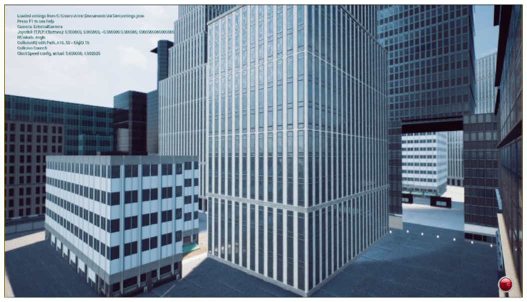


Figure 8. A scene from the created urban environment.

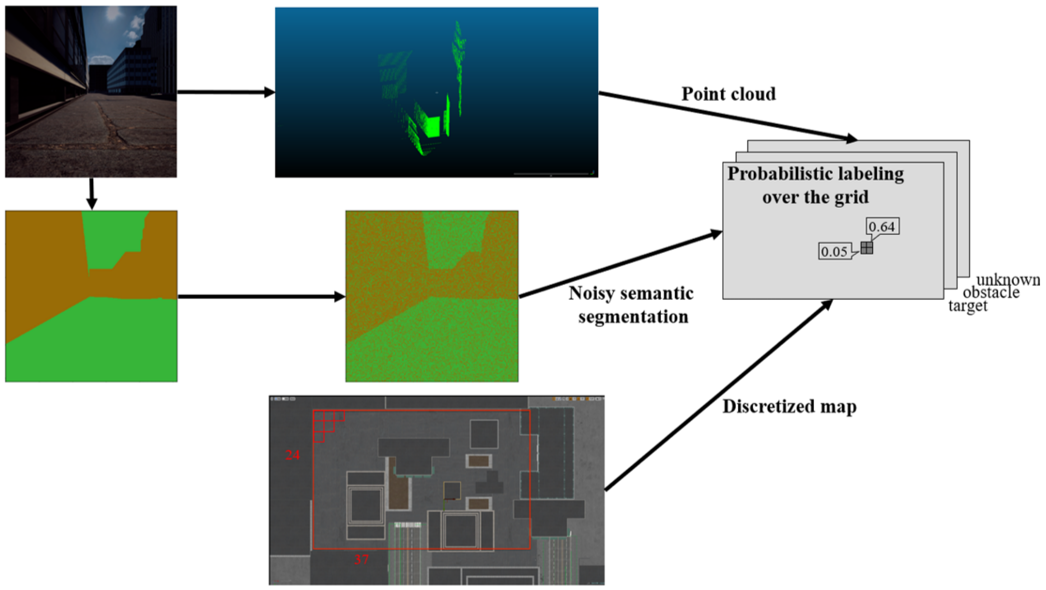

Point cloud
Probabilistic labeling over the grid
0.64
0.05
unknown
obstacle
target
Noisy semantic segmentation
Discretized map
24
37

Figure 9. The data processing of the drone's perception module in the AirSim environment. The depth data is used to create a point cloud and the RGB data is used to create a semantic segmentation. The information from the point cloud and the semantic segmentation are integrated to generate a probabilistic labeling over the discretized map. Probabilistic labels consist of the target building, the obstacles (including other buildings), and unknown entities that include any entities not related to the task.

in the environment. We semantically segment the RGB data based on three classes of target building, obstacles (e.g., other buildings), and unknown entities (e.g., sky). We add a noise of up to 0.25 to the output of semantic segmentation to model perception uncertainty. By mapping the coordinates from the point cloud to the cells of the discretized map, we match each pixel to a cell. Then, for each cell we average the labeling output of semantic segmentation from the pixels matched to that cell. The average labeling probability of each cell determines the probabilistic semantics.

Since in this setting, an exact observation model does not exist, we resort to a frequentist way of updating the agent's belief over the states' properties. To realize that, at each time step, we compute the set of states that are not visible given the current obstacles in the view. Therefore, we can keep track of how many times each cell has been visited and use that frequency to update the belief. At time step t, $\mathcal{L}_t(s,p) = \mathcal{L}_{t-1}(s,p)$ if state s is not visible and updates according to

$$\mathcal{L}_t(s,p) = \frac{\mathcal{L}_{t-1}(s,p) * \text{freq}(s) + Pr_{avg}(s \models p)}{\text{freq}(s) + 1}$$

if state s is visible, where $\text{freq}(s)$ is the frequency of visiting state s up to time $t-1$ and $Pr_{avg}(s \models p)$ is the output of the perception module at the current time step.

We simulate three different scenarios. In the first scenario, the drone has exact knowledge of the semantic labeling and hence, synthesizes a policy only at the beginning and continues with that policy. In the second scenario, the drone starts with a noisy belief of the semantic labeling, however, it receives perception data from a constructed observation function. The observation function provides noisy measurements of the labels over the entire map. In the third scenario, the drone is equipped with the more realistic perception module depicted in Figure 9 that gathers local RGB and depth measurements. In both the second and the third scenarios, the drone's semantic knowledge evolves over time and hence, it has to replan as necessary. To reduce the size and shorten the videos, we have sped up the recordings of the drone's flight for each of these scenarios by $16\times$ and made them available in the supplementary material.

As seen in the videos, the drone successfully reaches to the flagged building in all three scenarios. In the first scenario, the drone reaches to the target sooner than the other scenarios due to it having access to the perfect labeling as well as the fact that it does not need to synthesize a policy except at the initial point. In the second scenario, the drone receives measurements of the labels through a constructed observation function which it can use to update its belief in a Bayesian way. Since these measurements are from the entire environment, the drone's knowledge improves more quickly than the third scenario. As a result of that, it is able to complete the task earlier than the third scenario. However, in the third scenario where the agent has a realistic perception module with local observations, it is able to successfully complete the task.

References

Baier, C. and Katoen, J.-P. *Principles of model checking*. MIT press, 2008.

Moldovan, T. M. and Abbeel, P. Safe exploration in Markov decision processes. In *Proceedings of the 29th International Coference on International Conference on Machine Learning*, pp. 1451–1458, 2012.

Shah, S., Dey, D., Lovett, C., and Kapoor, A. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and Service Robotics*, 2017. URL https://arxiv.org/abs/1705.05065.