Handling Long and Richly Constrained Tasks through Constrained Hierarchical Reinforcement Learning

Yuxiao Lu[1], Arunesh Sinha[2], Pradeep Varakantham[1]
[1]Singapore Management University
[2]Rutgers University
yxlu.2021@phdcs.smu.edu.sg, arunesh.sinha@rutgers.edu, pradeepv@smu.edu.sg

Abstract

Safety in goal directed Reinforcement Learning (RL) settings has typically been handled through constraints over trajectories and have demonstrated good performance in primarily short horizon tasks. In this paper, we are specifically interested in the problem of solving temporally extended decision making problems such as robots cleaning different areas in a house while avoiding slippery and unsafe areas (e.g., stairs) and retaining enough charge to move to a charging dock; in the presence of complex safety constraints. Our key contribution is a (safety) Constrained Search with Hierarchical Reinforcement Learning (CoSHRL) mechanism that combines an upper level constrained search agent (which computes a reward maximizing policy from a given start to a far away goal state while satisfying cost constraints) with a low-level goal conditioned RL agent (which estimates cost and reward values to move between nearby states). A major advantage of CoSHRL is that it can handle constraints on the cost value distribution (e.g., on Conditional Value at Risk, CVaR) and can adjust to flexible constraint thresholds without retraining. We perform extensive experiments with different types of safety constraints to demonstrate the utility of our approach over leading approaches in constrained and hierarchical RL.

1 Introduction

Reinforcement Learning (RL) is a framework to solve decision learning problems in environments that have an underlying (Partially Observable) Markov Decision Problem, (PO-)MDP. Deep Reinforcement Learning (François-Lavet et al. 2018; Hernandez-Leal, Kartal, and Taylor 2019) approaches have been shown to solve large and complex decision making problems. For RL agents to be relevant in the day-to-day activities of humans, they need to handle a wide variety of temporally extended tasks while being safe. A few examples of such multi-level tasks are: (a) planning and searching for valuable targets by robots in challenging terrains (e.g., disaster areas) while navigating safely and preserving battery to reach a safe spot; (b) for autonomous electric vehicles to travel long distances in minimum time, they need to optimize the position of recharge locations along the way to ensure the vehicle is not left stranded; (c) cleaning robots to clean a house while avoiding slippery and unsafe areas (e.g., stairs) and retaining enough charge to move to a charging dock. The following key challenge needs to be addressed in the above mentioned problems of interest:

- Computing an execution policy that satisfies safety constraints (in expectation or in a confidence bounded way) for temporally extended decision making problems in the presence of uncertainty.

Existing research in temporally extended decision making problem has focused on hierarchical RL methods (Nachum et al. 2018; Zhang et al. 2020; Kim, Seo, and Shin 2021; Levy et al. 2017). These approaches successfully solve long horizon tasks mainly in the widely applicable setting of goal conditioned RL (Liu, Zhu, and Zhang 2022), but they are unable to deal with safety constraints. On the other hand, most existing research in handling *trajectory based safety constraints* has focused on constrained RL approaches (Simão, Jansen, and Spaan 2021; Gattami, Bai, and Aggarwal 2021), where constraints are enforced on expected cost. A recent method that has considered percentile/confidence based constraints is WCSAC (Yang et al. 2021). Unfortunately, these constrained RL approaches are typically only able to solve short horizon problems where the goal is not too far away. We address the need to bring together these two threads of research on hierarchical RL and constrained RL, which have mostly progressed independently of each other (Roza, Roscher, and Günnemann 2023). To that end, we propose a new Constrained Search with Hierarchical Reinforcement Learning (CoSHRL) approach, where there is a hierarchy of decision models: (a) The lower level employs goal conditioned distributional RL to learn reward and cost distributions to move between two local states that are near to each other. (b) The upper level is a constrained search mechanism that builds on Informed RRT* (Gammell, Srinivasa, and Barfoot 2014) to identify the best waypoints to get from a given start state to a "far" away goal state. This is achieved while ensuring overall expected or percentile cost constraints (representative of robust safety measures) are enforced.

Contributions: Our key contributions are: (1) we provide a *scalable constrained* search approach suited for *long horizon tasks* within a hierarchical RL set-up, (2) we are able to handle rich *percentile* constraints on cost distribution, (3) the design of enforcing the constraints at the upper-level search allows *fast recomputation* of policies in case the constraint threshold or start/goal states change, and (4) *mathematical guarantee* for our constrained search method. Finally, we provide an extensive empirical comparison of CoSHRL to leading approaches in hierarchical and constrained RL.

Related Work: *Constrained RL* uses the Constrained MDP (CMDP) to maximize a reward function subject to *expected* cost constraints (Satija, Amortila, and Pineau 2020; Pankayaraj and Varakantham 2023; Achiam et al. 2017; Gattami, Bai, and Aggarwal 2021; Tessler, Mankowitz, and Mannor 2018; Liang, Que, and Modiano 2018; Chow et al. 2018; Simão, Jansen, and Spaan 2021; Stooke, Achiam, and Abbeel 2020; Liu et al. 2022; Yu, Xu, and Zhang 2022; Zhang, Vuong, and Ross 2020). WCSAC (Yang et al. 2021) extends Soft Actor-Critic and considers a certain level of CVaR of the cost distribution as a safety measure; (Chow et al. 2017) use Lagrangian approach for the same. (Sootla et al. 2022) prevent only worst case cost (no CVaR or expected) violation by tracking the cost budget in the state, which further does not allow for multiple constraints. As far as we know and from benchmarking work (Ray, Achiam, and Amodei 2019a), there is no constrained RL designed for long-horizon tasks, and even for short-horizon all current approaches need retraining if the constraint threshold changes.

Hierarchical Reinforcement Learning (HRL) addresses the problem of sequential decision making at multiple levels of abstraction (Kulkarni et al. 2016; Dietterich 2000). The problem could be formulated with the framework of MDP and semi-MDP (SMDP) (Sutton, Precup, and Singh 1999). Utilizing off-policy RL algorithms, a number of recent methods such as HIRO (Nachum et al. 2018), HRAC (Zhang et al. 2020), and HIGL (Kim, Seo, and Shin 2021) propose a hierarchy where both lower and upper level are RL learners and the higher level specifies sub-goals (Kaelbling 1993) for the lower level. However, it is hard to add safety constraints to such HRL with RL at both levels because to enforce constraints the higher level policy must generate constraint thresholds for the lower-level agent while ensuring the budget used by multiple invocations of the lower-level agent does not exceed the total cost budget. Also, the lower-level policy should be able to maximize reward for *any* given cost threshold in the different invocations by the upper level. However, both these tasks are not realizable with the existing results in constrained RL. Options or skills learning coupled with a higher level policy of choosing options is another approach (Eysenbach et al. 2018; Kim, Ahn, and Bengio 2019) in HRL. CoSHRL can be viewed as learning primitive skills of reaching local goals, and the simplicity of this task as well as of the search makes our approach scalable and flexible.

Closer to our method, SORB (Eysenbach, Salakhutdinov, and Levine 2019) employs a graph-based path-planning (Dijkstra's algorithm) at the higher level and distributional RL at low level, where the continuous state is discretized to yield a massive graph. SORB achieves better success rate in complex maze environments compared to other HRL techniques but cannot enforce constraint and has high computational cost due to a large graph. We present a thorough comparison of our ConstrainedRRT* to SORB's planner in Section 2. PALMER (Beker, Mohammadi, and Zamir 2022) employs RRT* for the high level, but instead of distributional RL at the low-level it uses an offline RL like approach, requiring a large pre-collected dataset fully covering the environment; importantly, PALMER also cannot enforce constraints.

Logic based compositional RL (Jothimurugan et al. 2021; Neary et al. 2022) shares similarities with our approach in terms of combining a high-level planner with a low-level RL agent. However, works in compositional RL have a binary logical specification of success, whereas we are in a quantitative setting of constrained MDP with rewards and cost constraints (and novel CVaR constraints). Also, our utilization of the RRT* planner is quite different from the reachability planner used in these works.

Problem Formulation

We have an agent interacting with an environment in a Markov Decision Process (MDP) setting. The agent observes its current state $s \in S$, where $S \subset \mathbb{R}^d$ is a continuous state space. The initial state s_O for each episode is sampled according to a specified distribution and the agent seeks to reach goal state s_G. The agent's action space can be continuous ($a \subset \mathbb{R}^n$) or discrete. The episode terminates when the agent reaches the goal, or after T steps, whichever occurs first. The agent earns immediate reward $r^t(s^t, a^t)$ and separately also incurs immediate cost $c^t(s^t, a^t)$ when acting in time step t. $V^\pi(s_O, s_G)$ and $V_c^\pi(s_O, s_G)$ are the cumulative undiscounted expected reward and cost respectively for reaching goal state s_G from origin state s_O following policy π. The typical optimization in constrained RL (Achiam et al. 2017) is:

$$\max_\pi V^\pi(s_O, s_G) \quad s.t. \quad V_c^\pi(s_O, s_G) \le K \tag{1}$$

where the value functions are given as $V^\pi(s_O, s_G) = \mathbb{E}\big[\sum_{t=0}^T r^t(s^t, a^t)|s^T = s_G, s^0 = s_O\big]$ and $V_c^\pi(s_o, s_G) = \mathbb{E}\big[\sum_{t=0}^T c^t(s^t, a^t)|s^T = s_G, s^0 = s_O\big]$ with the expectation taken over policy and environment.

However, in the above, the constraint on the expected cost value is not always suitable to represent constraints on safety. E.g., to ensure that an autonomous electric vehicle is not stranded on a highway, we need a robust constraint that ensures the chance of that happening is low, which cannot be enforced by expected cost constraint. Therefore, we consider a cost constraint where we require that the CVaR (Rockafellar, Uryasev et al. 2000) of the cost distribution (given by the bold font random variable $\mathbf{V}_c^\pi(s_O, s_G)$ is less than a threshold. We skip writing s_O, s_G when implied. Intuitively, Value at Risk, VaR_α represents the minimum value for which the chance of violating the constraint (i.e., $\mathbf{V}_c^\pi > k$) is less than α specified as

$$VaR_\alpha(\mathbf{V}_c^\pi) = \inf\{k \mid Pr(\mathbf{V}_c^\pi > k) \le \alpha\}$$

Conditional VaR, $CVaR_\alpha$ intuitively refers to the expectation of values that are more than the VaR_α, i.e., $CVaR_\alpha(\mathbf{V}_c^\pi) = \mathbb{E}[\mathbf{V}_c^\pi \mid \mathbf{V}_c^\pi \ge VaR_\alpha(\mathbf{V}_c^\pi)]$. With this robust variant of the cost constraint (also known as percentile constraint), the problem that we solve for any given α is

$$\max_\pi V^\pi(s_O, s_G) \quad s.t. \quad CVaR_\alpha(\mathbf{V}_c^\pi) \le K \tag{2}$$

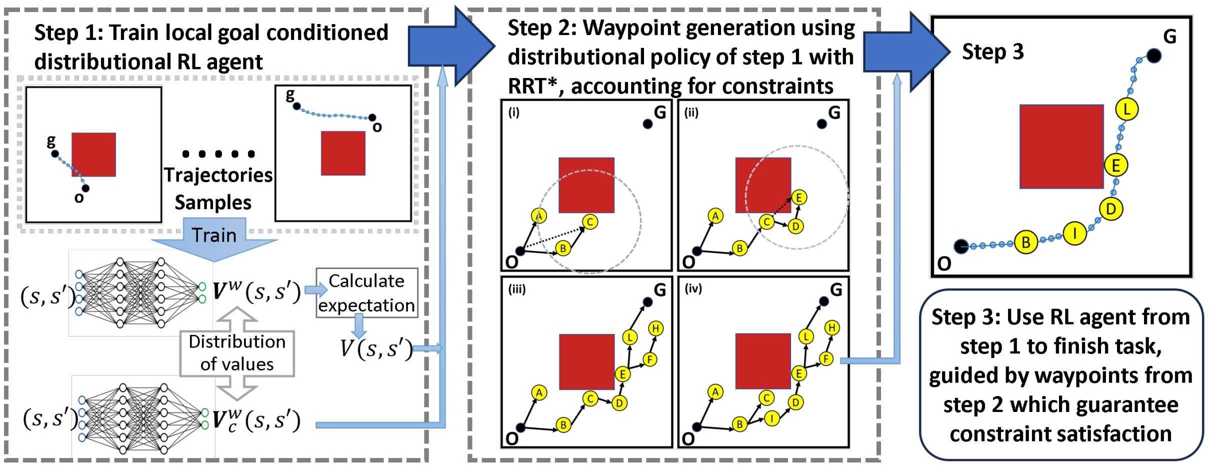
Step 1: Train local goal conditioned distributional RL agent
Trajectories Samples
Train
(s, s')
$V^w(s, s')$
Calculate expectation
$V(s, s')$
Distribution of values
$V_c^w(s, s')$
Step 2: Waypoint generation using distributional policy of step 1 with RRT*, accounting for constraints
Step 3
Step 3: Use RL agent from step 1 to finish task, guided by waypoints from step 2 which guarantee constraint satisfaction

Figure 1: Overview of CoSHRL. **Step 1**: Train a local goal-conditioned RL agent using multiple randomly selected (o, g) (o is start, g is goal) pairs in a constrained environment (top part). The red square indicates a high-cost region. The learning is local and hence the goal will be unreachable if it's not "near" to the start. In this step, the local value function V and the cost function V_c are learned. **Step 2**: Generate waypoints guided by V and V_c using the proposed ConstrainedRRT* algorithm (i) The search samples state C, and O is not within the dashed circle of "near" states. Although both A and B are within the circle, the path from O to C via B is better as V(O,B) $+V$(B,C) $<$ V(O,A) $+V$(A,C) using low-level agent's V function. So, edge (B, C) is added to the tree. (ii) For new sample E, E is "near" from C and D, but the edge (C, E) is not valid because of cost constraint $CVaR_\alpha(\mathbf{V}_c(\text{O, B}) + \mathbf{V}_c(\text{B, C}) + \mathbf{V}_c(\text{C, E})) > K$. (iii) A path (O, B, C, D, E, L, G) within the cost constraint is found. (iv) As the number of sampled states increases, a better path (O, B, I, D, E, L, G) is found. **Step 3**: Leveraging the waypoints from step 2, the pre-trained goal-conditioned RL agent completes the task.

Note that $\alpha = 1$ is risk neutral, i.e., $CVaR_1(\mathbf{V}_c^\pi) = \mathbb{E}[\mathbf{V}_c^\pi] = V_c^\pi$, and α close to 0 is completely risk averse.

2 Approach

Our approach, referred to as CoSHRL, solves the problem in Equation 2. As shown in Figure 1, CoSHRL employs a lower-level distributional RL agent and an upper-level search agent. First, the *goal-conditioned* (Kaelbling 1993) off-policy distributional RL agent learns local distribution of reward and cost between states that are "near" to each other. Then, the upper-level agent is constructed using a constrained search algorithm by utilizing the reward and cost distributions. Finally, through its interactions with the environment, the lower-level agent reaches the far away goal guided by the waypoints produced by the constrained search.

Lower Level Agent: Distributional RL (Bellemare, Dabney, and Rowland 2023) is a popular technique that enables learning distribution of value functions instead of just expected values. Distributional RL learns a policy $\hat{\pi}$ and maintains a network representing the distribution of $\mathbf{Q}$; we show how to derive $\mathbf{V}$, $\mathbf{V}_c$ from the learned policy $\hat{\pi}$.

Why distributional RL? For rewards, we need to estimate just the expected $V^\pi(s, s')$, but it is known from the literature that learning the distribution of $\mathbf{V}^\pi$ and then calculating expected value leads to better estimates (Eysenbach, Salakhutdinov, and Levine 2019; Beker, Mohammadi, and Zamir 2022). For completeness, we provide experimental evidence of this phenomenon in Appendix. For enforcing percentile based cost constraint, we need to estimate the distribution of cost $\mathbf{V}_c$ for the $\hat{\pi}$ learned by lower-level agent. This is only possible with the use of distributional RL.

Representation: In distributional RL for discrete actions, the distribution of $\mathbf{Q}$ is assumed to be over N discrete values. The distribution of a goal conditioned $\mathbf{Q}$ is represented by Q^θ (neural network parameterized by θ), which takes as input s, s', a (s' is local goal) and outputs a vector $[p_1, \ldots, p_N]$ where p_i is the probability of expected reward value taking the i^{th} discrete value. For completeness, the standard training of distributional RL is described in the Appendix, yielding a trained policy $\hat{\pi}$. For training, we choose nearby start and end states at random throughout the state space, relying on the generalizability of neural networks to obtain good estimates for nearby start and goal in the whole state space.

Next, for discrete actions, we represent the distribution of value $\mathbf{V}^{\hat{\pi}}$ as a neural network V^w, which again outputs a probability vector. For simplicity, we do not include the learned policy $\hat{\pi}$ (which will not change) in the notation for V^w. The fixed learned $\hat{\pi}$ allows us to estimate V^w directly by minimizing the KL divergence between a target $V^t(s, s') = Q^\theta(s, s', a), a \sim \hat{\pi}(\cdot|s, s')$ and the current V^w, i.e., $\min_w D_{KL}(V^t || V^w)$. We optimize the above by storing experiences sampled according to $\hat{\pi}$ in a replay buffer and sampling mini-batches to minimize the loss above, analogous to supervised learning. Once the vector of probabilities V^w is obtained, we can obtain the expected V by calculating the expectation.

For continuous actions, we can directly learn the distribution of $\mathbf{V}$, represented by a network V^w using the same vector of probability representation of the distribution of value as used above for Q^θ.

For problems in path search with no movement uncertainty, reward r is set to -1 for each step such that the learned expected negated reward value function $-V(s, s')$ reflects the estimated length of the shortest path (avoiding impenetrable obstacles) from s to s' as done in (Kaelbling 1993; Eysenbach, Salakhutdinov, and Levine 2019). In particular, we assume that $-V$ is learned accurately and prove the following result:

Lemma 1. *Given $S \subset \mathbb{R}^d$, assuming $-V$ gives the obstacle avoiding shortest path length, $-V$ is a distance metric.*

Next, for costs, we note that we performed the reward estimation without considering costs since in our approach the lower-level agent does not enforce constraints. However, the lower level agent does estimate the local costs as distributional $\mathbf{Q}$ values as a Q_c^θ network in the discrete action case or distributional $\mathbf{V}$ values as a V_c^w network in the continuous action case. Then, in the discrete action case, similar to above learning of V^w, the fixed learned policy $\hat{\pi}$ allows us to estimate the vector of probability V_c^w function directly by minimizing the KL divergence between a target $V_c^t = Q_c^\theta(s, s', a), a \sim \hat{\pi}(\cdot|s, s')$ and the current V_c^w: $\min_w D_{KL}(V_c^t || V_c^w)$. In the continuous action case, the network V_c^w is already learned directly (details in Appendix).

Upper Level Agent: Once the lower-level RL training is complete, we obtain a *local* goal-conditioned value function for any origin and local goal state that are near to each other. In this section, we use the learned expected value V and cost random variable $\mathbf{V}_c$ (*removing superscripts for notation ease*). First, we formulate the upper-level optimal constrained search problem. The RRT* search works in a continuous space $S \subset \mathbb{R}^d$. A path is a continuous function $\sigma : [0, 1] \rightarrow \mathbb{R}^d$ with the start point as $\sigma(0)$ and end as $\sigma(1)$. In practice, a path is represented by a discrete number of states $\{\sigma(x_i)\}_{i \in [n]}$ for $0 = x_0 < x_1 < ... < x_{n-1} < x_n = 1$ and some positive integer n (n can be different for different paths). A collision-free path is one that has no overlap with fixed obstacles. The set of all paths is Σ, and the set of obstacle free paths is Σ_{free}. A length of path is defined as $\sup_{n:0=t_0<\ldots t_n=1} \sum_{i=1}^n d(x_{t_{i-1}}, x_{t_i})$ for given underlying distance d. The RRT* search (or the Informed version) finds the shortest path from the given start and end point.

Given the discrete representation, for our CoSHRL the path traversed between $\sigma(x_i)$ and $\sigma(x_{i+1})$ is determined by the lower-level agent's policy. Every path $\sigma \in \Sigma$ provides a reward R_σ and incurs a cost C_σ. We define the reward for segment $(\sigma(x_i), \sigma(x_{i+1}))$ of a path as $V(\sigma(x_i), \sigma(x_{i+1}))$, where V is the local goal-conditioned value function learned by the lower-level agent. Similarly, the cost incurred for segment $(\sigma(x_i), \sigma(x_{i+1}))$ is $\mathbf{V}_c(\sigma(x_i), \sigma(x_{i+1}))$. Thus,

$$R_\sigma = \sum_{i=0}^{n-1} V(\sigma(x_i), \sigma(x_{i+1})) \quad (3)$$

$$\mathbf{C}_\sigma = \sum_{i=0}^{n-1} \mathbf{V}_c(\sigma(x_i), \sigma(x_{i+1})) \quad (4)$$

In CoSHRL, the constrained search problem is to find the optimal path, σ^* ($\in \text{argmax}_{\sigma \in \Sigma} R_\sigma$) from s_O to s_G subject to a cost threshold, i.e., $CVaR_\alpha(\mathbf{C}_{\sigma^*}) \leq K$. As $-V$ is the shortest distance considering obstacles (see the text before Lemma 1), the above optimization essentially finds the shortest path measured in distance $-V$ from s_O to s_G avoiding all obstacles and within the cost constraint K.

Algorithm 1: ConstrainedRRT* $(s_o, s_G, V, \mathbf{V}_c, K)$

```
1  $\mathcal{V} \leftarrow \{s_o\}, \mathcal{E} \leftarrow \emptyset, S_{soln} \leftarrow \emptyset, \mathcal{T} = (\mathcal{V}, \mathcal{E})$
2  for iteration = 1 ... N do
3      $\mathcal{T}$ = Extend_node$(s_o, s_G, V, \mathbf{V}_c, K, \mathcal{T}, S_{soln})$
4  return best solution in $S_{soln}$
   def Extend_node $(s_o, s_G, V, \mathbf{V}_c, K, \mathcal{T}, S_{soln})$
5      Sample $s_{new}$ within $\min(r_{RRT*}, \eta)$ from its
         nearest node in $\mathcal{T}$ as in Informed RRT*
6      $S_{near} \leftarrow$ Find all nodes in $\mathcal{T}$ within
         $\min(r_{RRT*}, \eta)$ from $s_{new}$
7      Find $s_{min} \in \text{argmin}_s\{-R(s) - V(s, s_{new}) \mid s \in$
         $S_{near}$, Valid_edge$(\mathcal{T}, s, s_{new}, \mathbf{V}_c, K)\}$
8      $\mathcal{V} \leftarrow \mathcal{V} \cup \{s_{new}\}, \mathcal{E} \leftarrow \mathcal{E} \cup \{(s_{min}, s_{new})\}$
9      $S_{cand} = \{s \in S_{near} \mid -R(s_{new}) - V(s_{new}, s) <$
         $-R(s)$, Valid_edge$(\mathcal{T}, s_{new}, s, \mathbf{V}_c, K)\}$
10     for $\forall s \in S_{cand}$ do
11         $s_{parent} \leftarrow$Parent$(s)$, $\mathcal{E} \leftarrow \mathcal{E} \backslash \{(s_{parent}, s)\}$,
             $\mathcal{V} \leftarrow \mathcal{V} \cup \{s_{new}\}, \mathcal{E} \leftarrow \mathcal{E} \cup \{(s_{new}, s)\}$
12     If $s_{new}$ is near the goal $s_G$, then form $\sigma$ by
         tracking parents of $s_{new}$ and $S_{soln} \leftarrow S_{soln} \cup \{\sigma\}$
```

Our approach has immediate advantages over the state-of-the-art SORB (Eysenbach, Salakhutdinov, and Levine 2019), which also employs an upper level planner and lower level RL. SORB constructs a complete graph and then computes the shortest path using Dijkstra's algorithm. However, SORB has fundamental limitations: (1) The graph is built from the replay buffer of explored nodes. This can result in bad distribution of nodes in the state space (without considering start, goal, or obstacles). (2) The coarse discretization can result in a non-optimal path between the start and goal state (Karaman and Frazzoli 2011). (3) Construction of complete graph yields $O(N^2)$ complexity for Dijkstra's algorithm with N nodes (compared to $N \log N$ for our search).

Thus, an online search method that samples and grows a tree from the given start to the goal state while avoiding extending into obstacles is more suited as the upper-level search. Hence we provide Constrained-RRT*, which builds on Informed-RRT* (Gammell, Srinivasa, and Barfoot 2014) to handle constraints. Informed-RRT* builds upon RRT* (Karaman and Frazzoli 2011), which works by constructing a tree whose root is the start state and iteratively growing the tree by randomly sampling new points as nodes till the tree reaches the goal. In Informed RRT*, as an informed heuristic, the sampling is restricted to a *specially constructed ellipsoid.* However, both Informed-RRT* and RRT* do not take constraints into account.

Algorithm Description: We propose Constrained RRT* (Algorithm 1), which builds on Informed RRT* to handle the cost constraint. The pseudocode is provided in Algorithm 1. We search for the optimal path σ^* by incrementally building a tree $\mathcal{T}$ in the state space S. The tree, $\mathcal{T}$ consists of a

Algorithm 2: Valid_edge ($\mathcal{T}, s, s', \mathbf{V}_c, K$)

```
1 $result \leftarrow \mathbf{V}_c(s, s')$
2 while $s.parent$ do
3     $result \leftarrow result + \mathbf{V}_c(s.parent, s)$
4     $s \leftarrow s.parent$
5 if $CVaR_\alpha(result) \leq K$ then
6     return True
7 return False
```

set of nodes, $\mathcal{V}$ ($\subset S$), and edges $\mathcal{E}$ ($\subset S \times S$). In the subroutine Extend_node, a candidate state s_{new} is chosen (line 5) to be added to the tree $\mathcal{T}$ by a *sampling process that is the same as in Informed RRT** (see Appendix for details of sampling). The hyper-parameter η accounts for the fact that our distance estimates are precise only locally (see Appendix for hyperparameter settings).

The rewiring radius, $r_{RRT*} = \gamma_{RRT*}(\log n/n)^{1/d}$, where n is the current number of nodes sampled, is described in (Karaman and Frazzoli 2011). The node s_{min} (line 7) that results in the shortest path (highest reward) to s_{new} among the nearby nodes S_{near} (line 6) is connected to s_{new} in line 8, if the edge is valid.

Here, we take a detour to explain how we determine the validity of edges. An edge is valid if and only if adding it does not result in a (partial) path that violates the cost constraint. The key insight is that this validity can be determined by computing the convolution of the distributions associated with the (partial) path and the current $\mathbf{V}_c$. By providing the definition of Valid_edge ($\mathcal{T}, s, s', \mathbf{V}_c, K$) in Algorithm 2 and doing the Valid_edge checks in the Extend_node subroutine, we ensure that any path output by the overall algorithm will satisfy the cost constraints. In the pseudocode of Valid_edge, $\mathbf{V}_c$ represents a random variable (and so does $result$). Then, the addition in line 3 of Valid_edge is a convolution operation (recall that the distribution of a sum $\mathbf{X} + \mathbf{Y}$ of two random variables $\mathbf{X}, \mathbf{Y}$ is found by a convolution (Ross 2014)).

Coming back to Extend_node, we explore further the possible edges to be added to the tree. In particular, in line 9 (1) the edge is created only if it is valid and (2) new edges are created from s_{new} to vertices in S_{near}, if the path through s_{new} has lower distance (higher reward) than the path through the current parent; in this case, the edge linking the vertex to its current parent is deleted, to maintain the tree structure. An example search run is shown in Figure 1.

Theoretical Results: The RRT* algorithm (Karaman and Frazzoli 2011) satisfies two properties: *probabilistic completeness* and *asymptotic optimality*. Intuitively, probabilistic completeness says that as number of samples $n \to \infty$, RRT* finds a feasible path if it exists and asymptotic optimality says that as $n \to \infty$, RRT* finds the optimal path with the highest reward. Unsurprisingly, asymptotic optimality implies probabilistic completeness. Our key contribution is proving asymptotic optimality of ConstrainedRRT*, which requires complicated analysis because of constraints.

Background: We summarize many definitions from Karaman and Frazzoli (2011). For detailed definition statements, we request the reader to peruse the referred paper. Karaman and Frazzoli (2011) define addition and multiplication operations that make the set of paths Σ a vector space. Further, they define a norm $||\sigma||_{BV}$ on this vector space (please refer to page 22 of (Karaman and Frazzoli 2011)). The distance induced by the BV norm allows for defining limits of a sequence of path, i.e., $\lim_{n\to\infty} \sigma_n$. A solution path σ^* is called *robustly optimal* if under the metric induced by the BV norm for any sequence of collision-free paths σ_n, if $\lim_{n\to\infty} \sigma_n = \sigma^*$ then $\lim_{n\to\infty} R_{\sigma_n} = R_{\sigma^*}$. A path is said to have *strong δ clearance* if it is not within δ distance of any obstacle. A path σ has weak δ clearance if there exists a sequence of paths with strong clearance converging to σ. For any path finding algorithm ALG, let Y_n^{ALG} be the random variable corresponding to the reward of the max-reward solution returned at the end of iteration n.

Definition 1 (Asymptotic optimality (Karaman and Frazzoli 2011))**.** *An algorithm ALG is asymptotically optimal if, for any path search problem that admits a robustly optimal solution with finite reward R^*, $\mathbb{P}(\{\limsup_n Y_n^{ALG} = R^*\}) = 1$.*

Theoretical Results for Constraints: In this paper, due to the presence of constraints, we have to modify definitions. For instance, robustly optimal definition has to account for costs, i.e., the solution path σ^* is called *robustly optimal with constraints* if under the metric induced by the BV norm for any sequence of collision-free paths σ_n if $\lim_{n\to\infty} \sigma_n = \sigma^*$ then $\lim_{n\to\infty} R_{\sigma_n} = R_{\sigma^*}$ and if $\lim_{n\to\infty} \sigma_n = \sigma^*$ then $\lim_{n\to\infty} C_{\sigma_n} = C_{\sigma^*}$. Next, the definition of weak δ clearance of optimal path σ^* is extended to assume that there exists a sequence of strong δ clearance paths with total cost $\leq K + \epsilon$ when the path σ^* has cost $\leq K$ for any small $\epsilon > 0$. Intuitively, this means that if the optimal path has cost at most K then nearby strong δ clearance paths converging to the optimal path are also cost bounded closely by K while allowing ϵ extra cost for possibly slightly longer paths. We redefine Definition 1 with the cost constraint. Let Z_n^{ALG} be the random variable corresponding to the cost of the max-reward solution included in the graph returned by ALG at the end of iteration n (n samples). Then, we define:

Definition 2 (Asymptotic optimality with constraints)**.** *An algorithm ALG is asymptotically optimal with constraints if, for any path search problem that admits a robustly optimal solution with finite cost constraints K and with finite reward R^*, $\mathbb{P}(\{\limsup_n Y_n^{ALG} = R^*\}) = 1$ and $Z_n^{ALG} \leq K$.*

We justify this definition as follows: since ALG will stop in finite n, we require that the output of ALG is always within the cost threshold K for any n at which the algorithm stops. We prove that our change (Valid_Edge check) preserves asymptotic optimality with constraints.

Theorem 1. *Let d be the dimension of the space S, $\mu(S_{free})$ denotes the Lebesgue measure (i.e., volume) of the obstacle-free space, and τ_d be the volume of the unit Euclidean norm ball in the d-dimensional space. The Constrained RRT* in Algorithm 1 preserves asymptotic optimality with constraints for $\gamma_{RRT*} \geq (2(1 + 1/d))^{1/d} \big(\frac{\mu(S_{free})}{\tau_d}\big)^{1/d}$.*

The proof of RRT* involves constructing a random graph via a marked point process that is shown as equivalent to

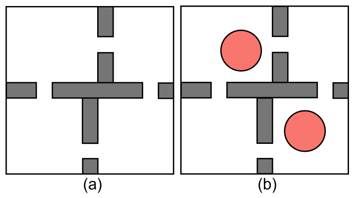

Figure 2(a): The complex point maze environment. Wall obstacles are in black. The environment on the right has hazardous red circles.

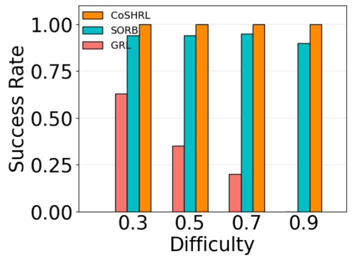

Figure 2(b): Success rate vs Difficulty

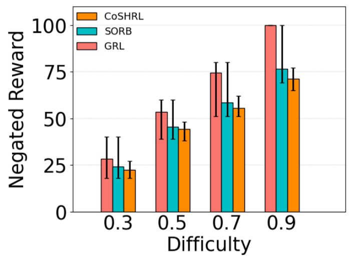

Figure 2(c): Neg. reward vs Difficulty

the RRT* algorithm. In the full proof in Appendix. we incorporate cost constraints in the construction of the random graph and show its equivalence to ConstrainedRRT*. Then, the analysis is done for this constructed random graph. The analysis involves (1) constructing a sequence of paths σ_n with strong δ_n clearance converging to the optimal path σ^* within cost constraint, (2) constructing a covering of the path σ_n with a sequence of norm balls with radius $\delta_n/4$; we use a special value for δ_n to account for cost constraints. It is shown that with large enough n and our special choice of δ_n, the tree in ConstrainedRRT* will have a path satisfying cost constraints through these balls and will converge to σ^*.

3 Experiments

We evaluate our method on two complex point maze environments and a novel image-based ViZDoom environment which have been used as a benchmark in RL navigation tasks (Zhang et al. 2020; Nachum et al. 2018; Beker, Mohammadi, and Zamir 2022). These maps include *obstacles* (impenetrable) and *hazards* (high cost but penetrable). We compare against SAC-Lagrangian (SAC-lag) (Yang et al. 2021; Stooke, Achiam, and Abbeel 2020), WCSAC (Yang et al. 2021), SORB (Eysenbach, Salakhutdinov, and Levine 2019), and Goal-conditioned RL (GRL) (Kaelbling 1993). SORB and GRL are not designed to enforce constraints, so they can get higher rewards but suffer from constraint violations. *Hyperparameter settings and additional results on other environments are in Appendix.*

2D Navigation with Obstacles: The first environment is point maze environment of Figure 2a (left), which has wall obstacles, but no hazards (thus, no cost constraints). The start point is randomly set in the environment while the goal is set 69ν away from the start where ν is the difficulty level. As the immediate reward $r(s, a) = -1$, the agent needs to reach the goal using the shortest path that avoids the walls.

We compare CoSHRL with goal-conditioned RL and SORB at different difficulty levels. For a fair comparison, both the number of nodes for SORB and the number of iterations for our method are set as 1000. For each experiment, we ran 100 trials with different seeds. We compare (a) the percentage of times the agent reaches the goal; and (b) the negated reward (i.e., the path length). In Figure 2b, we observe that the success rate of CoSHRL is 100% and it outperforms SORB with a larger margin as the difficulty level increases. In Figure 2c, we show all trials' negated reward (lower is better) for GRL, SORB, and CoSHRL. The difficulty level ν decides the optimal distance between start and goal, e.g., when $\nu = 0.3$, the optimal distance is set at $69 \times 0.3 \approx 21$; we observed that the baseline approaches frequently provided very circuitous paths much longer than the optimal path, e.g., SORB and GRL often provide circuitous paths with length exceeding 40 for $\nu = 0.3$, so we cut them off at 40 for $\nu = 0.3$. We cut all trajectories off for baselines (thereby providing advantage to baselines) at $40, 60, 80, 100$ for difficulty levels $0.3, 0.5, 0.7, 0.9$ respectively.

Yet, we observe that not only the average negated reward (path length) but also the upper bound and lower bound outperform SORB and GRL at different difficulty levels.

2D Navigation with Obstacles and Hazards: In this part, we evaluate our method in the point maze environment of Figure 2a (right), where there are two hazards set in the top left room and bottom left room. The agent starts randomly in the bottom left room and the goal is randomly set in the top right room. The trajectory length will be longer if the agent tries to avoid the hazardous area. We show results for static costs as well as for stochastic costs at different risk levels. It is worth noting that we don't need to retrain our lower-level RL policy for different cost thresholds K.

Static Cost: In this environment, the agent incurs a cost $c = 1$ for each step in the hazard, otherwise $c = 0$. We evaluate our method with different cost limits K shown with CoSHRL-4, CoSHRL-7, and CoSHRL-10 in Figure 3b and Figure 3a. In Figure 3b, the bars provide the path length (negated reward) to reach the goal (plotted on the primary Y-axis) and the purple dots indicate the success rate (plotted on the secondary Y-axis). For average negated reward (path length), we only consider the successful trials for all algorithms. We have the following key observations from Figure 3b: (1) Our method reaches the goal with a high success rate under different cost limits with nearly 100% success. (2) Even though our method considers cost constraints, it is able to outperform SORB (which does not consider the cost constraint) not only in success rate but also in the length of the trajectory (average negated reward). (3) The success rate of GRL (goal-conditioned RL) is less than 20% but for the average negated reward we only count the successful trials, hence the negated reward for goal-conditioned RL is better (lower) than our method. (4) WCSAC and SAC-Lagrangian, both non-hierarchical RL techniques that consider cost constraints, have $\approx 0\%$ success rate in this long-horizon task and we don't consider them as baselines in further experiments.

The min., max. and mean cost for the different algorithms

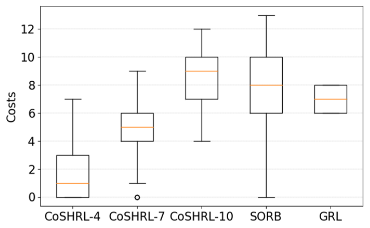

Figure 3(a): Boxplot of cost in evaluation for static risk after training in maze environment.

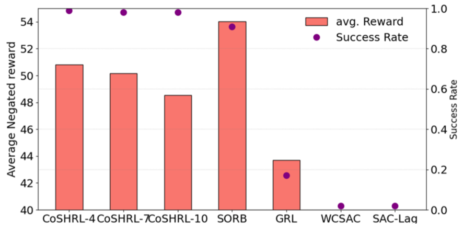

Figure 3(b): Success rate and avg. neg. reward of our method and baselines in maze environment. Only successful trials are counted for reward.

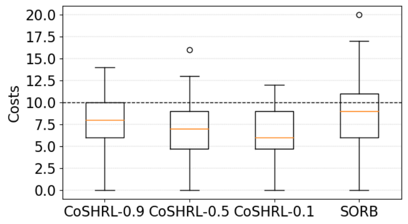

Figure 3(c): Boxplot of cost evaluation for stochastic risk after training

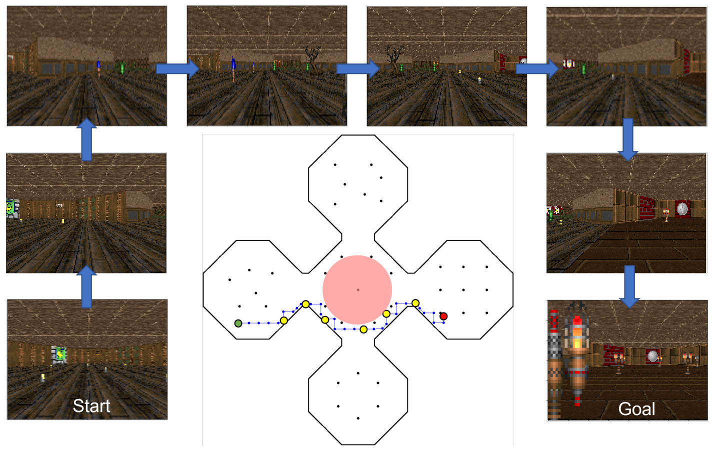

Figure 4(a): An example safe trajectory in the safe ViZDoom environment. Fixed obstacles are shown in black points and hazardous area is shown in red circle. Given a start state (green point) and goal state (red point), our method could find a sequence of waypoints (yellow points) conditioning on flexible constraints threshold K ($K = 0$ in this figure). Using the low level RL between the waypoints our method could reach the goal constraints (shown in the blue line).

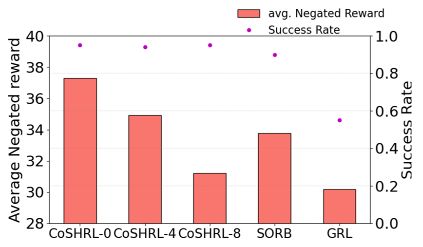

Figure 4(b): Success rate and avg. negated reward of our method, SORB, and GRL in Safe-ViZDoom. Only successful trials are counted for reward

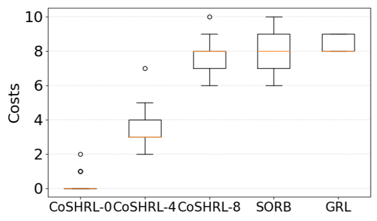

Figure 4(c): Boxplot of cost in evaluation after training in Safe-ViZDoom.

are shown in Figure 3a. With increasing cost limit, the upper bound, lower bound, and median of the total cost increase for CoSHRL. This is expected as the path in the hazardous area increases and therefore potentially the error in the computation of $\mathbf{V}_c$ can increase. The proportion of trajectories that exceed the cost limit $K = 4, 7, 10$ are $4\%, 6\%, 6\%$ respectively. Examples of paths produced by different approaches are shown in Appendix.

Stochastic Cost: In this environment, the agent incurs a cost c uniformly sampled from $\{0, 1, 2\}$ for each step in the hazard, otherwise $c = 0$, i.e., the total cost of n steps inside the hazard follows a multinomial distribution. In safety-critical domains, a worst-case cost guarantee is preferred over the average cost bound (Yang et al. 2021). To achieve this, we use CVaR (Rockafellar, Uryasev et al. 2000) instead of the expected value of cost to threshold the safety of a policy.

We set cost limit $K = 10$ for all α, that is, the expectation of the cost of the worst $\alpha * 100\%$ cases should be lower than K. We evaluate our method with different α shown with CoSHRL-0.9, CoSHRL-0.5, and CoSHRL-0.1. All experiments are averaged over 100 runs.

In Table 1, the results show that our method CoSHRL with $\alpha = 0.9, 0.5$ satisfy the corresponding CVaR bound

	EC	C0.9	C0.5	C0.1	ENR	%
CoSHRL-0.9	7.90	**8.47**	10.20	14.67	47.83	16%
CoSHRL-0.5	6.68	7.31	**9.20**	13.22	48.58	11%
CoSHRL-0.1	6.47	7.06	8.86	**12.11**	48.60	7%
SORB	8.06	8.97	11.54	15.14	52.98	30.5%

Table 1: Different metrics of performance in the environment with stochastic cost: expected cost (EC), cost-CVaR-0.9 (C0.9), cost-CVaR-0.5 (C0.5), cost-CVaR-0.1 (C0.1), and expected negated reward (ENR)

(columns Cα shows the estimated average costs of the worst $\alpha * 100\%$ trajectories) while CoSHRL violates the CVaR bound ($K = 10$) slightly with the tight level $\alpha = 0.1$ because of the inherent approximation in distributional RL, namely that of discretization and truncation of long-tailed multinomial distribution. As α decreases, our method is more risk-averse so the percentage of trajectories that exceed the cost limit K decreases (% column), and cost and reward both improve. The statistical properties of the total cost incurred by CoSHRL under different risk level α are shown in Figure 3c.

Image-based Navigation with Obstacles and Hazards:

Due to the lack of a constrained image-based environment, we design the Safe-ViZDoom environment in Figure 4a based on ViZDoom (Wydmuch, Kempka, and Jaśkowski 2019). The Safe-VizDoom environment is a labyrinth in the shape of a clover with a hazardous area in the middle, making it challenging due to the very narrow safe area in the middle. The agent can move North/South/East/West by a fixed distance, whereas states only consist of first-person visual perspective (3x160x120 dimension). The agent incurs a cost $c = 1$ for each step in the hazard, otherwise $c = 0$. The start is randomly placed in one of the four rooms, while the goal is randomly set in the opposite room.

We evaluate CoSHRL with different cost limits K shown as CoSHRL-0, CoSHRL-4, and CoSHRL-8 in Figure 4b and Figure 4c without retraining the low level RL agent. Each result is the average over 100 random runs. We obtain similar results to other domains. Figure 4b shows that CoSHRL achieves a high success rate ($> 95\%$) in reaching the goal with varying cost limits. As the cost limit increases, CoSHRL obtains shorter paths (avg. negated reward), indicating that the agent ventures deeper into hazards. For avg. negated reward, CoSHRL outperforms the *unconstrained* SORB and GRL for cost limit $K = 8$, which is roughly the cost incurred by SORB and GRL in Figure 4c. Figure 4c shows the proportions of trajectories exceeding the cost limits of $K = 0, 4, 8$ are 2%, 4%, 5% respectively. In comparison, unconstrained SORB and GRL achieve shorter path lengths (average negated reward) but incur cost over 8 in over half of their trajectories. The non-hierarchical GRL has a low success rate of 55%, resulting in the agent getting stuck in corners.

4 Discussion

We introduced a *constrained search* within the hierarchical RL approach. The RL agent is utilized to find paths between any two "nearby" states. Then, the constrained search utilizes the RL agent to reach far away goal states from starting states, while satisfying various types of constraints. We were able to demonstrate the better scalability, theoretical soundness, and empirical utility of our approach, CoSHRL, over existing approaches for Constrained RL and Hierarchical RL. Next, we discuss some limitations and future work.

Our work is based on the assumption that the low level RL agent has a high success rate in reaching each waypoint, even though there might be events such as action execution failure. RL in general can handle action execution uncertainty (process noise) by observing the current (unexpected) state after an action failure and appropriately executing contingency actions from such observations. Thus, the low level RL will ultimately reach the local goal even though it might occasionally (with some probability) take more steps due to action execution failure. In extreme cases, due to poor generalization the low level RL can declare a state unreachable, even though the state might be reachable. This can sometimes result in no path being found to the final goal. However, this happens very rarely, which is the main reason why the success rate of our method in the test environments (Figures 3b, 4b) are not exactly 100%. A possible direction to improve this is for constrained RRT* to actively ask for retraining the low level agent; an active retraining paradigm could be an interesting future research direction.

5 Acknowledgments

This research/project is supported by the National Research Foundation Singapore and DSO National Laboratories under the AI Singapore Programme (AISG Award No: AISG2-RP-2020-017)

References

Achiam, J.; Held, D.; Tamar, A.; and Abbeel, P. 2017. Constrained policy optimization. In *International conference on machine learning*, 22–31. PMLR.

Beker, O.; Mohammadi, M.; and Zamir, A. 2022. PALMER: Perception-Action Loop with Memory for Long-Horizon Planning. *arXiv preprint arXiv:2212.04581*.

Bellemare, M. G.; Dabney, W.; and Munos, R. 2017. A distributional perspective on reinforcement learning. In *International Conference on Machine Learning*, 449–458. PMLR.

Bellemare, M. G.; Dabney, W.; and Rowland, M. 2023. *Distributional Reinforcement Learning*. MIT Press. http://www.distributional-rl.org.

Chow, Y.; Ghavamzadeh, M.; Janson, L.; and Pavone, M. 2017. Risk-constrained reinforcement learning with percentile risk criteria. *The Journal of Machine Learning Research*, 18(1): 6070–6120.

Chow, Y.; Nachum, O.; Duenez-Guzman, E.; and Ghavamzadeh, M. 2018. A lyapunov-based approach to safe reinforcement learning. *Advances in neural information processing systems*, 31.

Dietterich, T. G. 2000. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of artificial intelligence research*, 13: 227–303.

Eysenbach, B.; Gupta, A.; Ibarz, J.; and Levine, S. 2018. Diversity is all you need: Learning skills without a reward function. *arXiv preprint arXiv:1802.06070*.

Eysenbach, B.; Salakhutdinov, R. R.; and Levine, S. 2019. Search on the replay buffer: Bridging planning and reinforcement learning. *Advances in Neural Information Processing Systems*, 32.

François-Lavet, V.; Henderson, P.; Islam, R.; Bellemare, M. G.; Pineau, J.; et al. 2018. An introduction to deep reinforcement learning. *Foundations and Trends® in Machine Learning*, 11(3-4): 219–354.

Gammell, J. D.; Srinivasa, S. S.; and Barfoot, T. D. 2014. Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2997–3004. IEEE.

Gattami, A.; Bai, Q.; and Aggarwal, V. 2021. Reinforcement learning for constrained markov decision processes. In *International Conference on Artificial Intelligence and Statistics*, 2656–2664. PMLR.

Han, D.-H.; Kim, Y.-D.; and Lee, J.-Y. 2014. Multiple-criterion shortest path algorithms for global path planning of unmanned combat vehicles. *Computers & Industrial Engineering*, 71: 57–69.

Hernandez-Leal, P.; Kartal, B.; and Taylor, M. E. 2019. A survey and critique of multiagent deep reinforcement learning. *Autonomous Agents and Multi-Agent Systems*, 33(6): 750–797.

Jothimurugan, K.; Bansal, S.; Bastani, O.; and Alur, R. 2021. Compositional reinforcement learning from logical specifications. *Advances in Neural Information Processing Systems*, 34: 10026–10039.

Kaelbling, L. P. 1993. Learning to achieve goals. In *IJCAI*, volume 2, 1094–8. Citeseer.

Karaman, S.; and Frazzoli, E. 2011. Sampling-based algorithms for optimal motion planning. *The international journal of robotics research*, 30(7): 846–894.

Kim, J.; Seo, Y.; and Shin, J. 2021. Landmark-guided subgoal generation in hierarchical reinforcement learning. *Advances in Neural Information Processing Systems*, 34: 28336–28349.

Kim, T.; Ahn, S.; and Bengio, Y. 2019. Variational temporal abstraction. *Advances in Neural Information Processing Systems*, 32.

Kulkarni, T. D.; Narasimhan, K.; Saeedi, A.; and Tenenbaum, J. 2016. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. *Advances in neural information processing systems*, 29.

Lenz, I.; Knepper, R. A.; and Saxena, A. 2015. DeepMPC: Learning deep latent features for model predictive control. In *Robotics: Science and Systems*, volume 10. Rome, Italy.

Levy, A.; Konidaris, G.; Platt, R.; and Saenko, K. 2017. Learning multi-level hierarchies with hindsight. *arXiv preprint arXiv:1712.00948*.

Liang, Q.; Que, F.; and Modiano, E. 2018. Accelerated primal-dual policy optimization for safe reinforcement learning. *arXiv preprint arXiv:1802.06480*.

Liu, M.; Zhu, M.; and Zhang, W. 2022. Goal-Conditioned Reinforcement Learning: Problems and Solutions. In Raedt, L. D., ed., *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, 5502–5511. International Joint Conferences on Artificial Intelligence Organization. Survey Track.

Liu, Z.; Cen, Z.; Isenbaev, V.; Liu, W.; Wu, S.; Li, B.; and Zhao, D. 2022. Constrained variational policy optimization for safe reinforcement learning. In *International Conference on Machine Learning*, 13644–13668. PMLR.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; and Riedmiller, M. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Nachum, O.; Gu, S. S.; Lee, H.; and Levine, S. 2018. Data-efficient hierarchical reinforcement learning. *Advances in neural information processing systems*, 31.

Neary, C.; Verginis, C.; Cubuktepe, M.; and Topcu, U. 2022. Verifiable and compositional reinforcement learning systems. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 32, 615–623.

Pankayaraj, P.; and Varakantham, P. 2023. Constrained reinforcement learning in hard exploration problems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, 15055–15063.

Ray, A.; Achiam, J.; and Amodei, D. 2019a. Benchmarking safe exploration in deep reinforcement learning. *arXiv preprint arXiv:1910.01708*, 7: 1.

Ray, A.; Achiam, J.; and Amodei, D. 2019b. Benchmarking Safe Exploration in Deep Reinforcement Learning.

Rockafellar, R. T.; Uryasev, S.; et al. 2000. Optimization of conditional value-at-risk. *Journal of risk*, 2: 21–42.

Ross, S. M. 2014. *Introduction to probability models*. Academic press.

Roza, F. S.; Roscher, K.; and Günnemann, S. 2023. Safe and Efficient Operation with Constrained Hierarchical Reinforcement Learning. In *Sixteenth European Workshop on Reinforcement Learning*.

Rudin, W. 1987. *Real and Complex Analysis*. Mathematics series. McGraw-Hill. ISBN 9780071002769.

Satija, H.; Amortila, P.; and Pineau, J. 2020. Constrained markov decision processes via backward value functions. In *International Conference on Machine Learning*, 8502–8511. PMLR.

Schaul, T.; Horgan, D.; Gregor, K.; and Silver, D. 2015. Universal value function approximators. In *International conference on machine learning*, 1312–1320. PMLR.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. 2016. Mastering the game of Go with deep neural networks and tree search. *nature*, 529(7587): 484–489.

Simão, T. D.; Jansen, N.; and Spaan, M. T. 2021. AlwaysSafe: Reinforcement learning without safety constraint violations during training. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*. International Foundation for Autonomous Agents and Multiagent Systems.

Sootla, A.; Cowen-Rivers, A. I.; Jafferjee, T.; Wang, Z.; Mguni, D. H.; Wang, J.; and Ammar, H. 2022. Sauté rl: Almost surely safe reinforcement learning using state augmentation. In *International Conference on Machine Learning*, 20423–20443. PMLR.

Stooke, A.; Achiam, J.; and Abbeel, P. 2020. Responsive safety in reinforcement learning by pid lagrangian methods. In *International Conference on Machine Learning*, 9133–9143. PMLR.

Sutton, R. S.; Precup, D.; and Singh, S. 1999. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2): 181–211.

Tessler, C.; Mankowitz, D. J.; and Mannor, S. 2018. Reward constrained policy optimization. *arXiv preprint arXiv:1805.11074*.

Watter, M.; Springenberg, J.; Boedecker, J.; and Riedmiller, M. 2015. Embed to control: A locally linear latent dynamics model for control from raw images. *Advances in neural information processing systems*, 28.

Wydmuch, M.; Kempka, M.; and Jaśkowski, W. 2019. ViZDoom Competitions: Playing Doom from Pixels. *IEEE Transactions on Games*, 11(3): 248–259. The 2022 IEEE Transactions on Games Outstanding Paper Award.

Yang, Q.; Simão, T. D.; Tindemans, S. H.; and Spaan, M. T. 2021. WCSAC: Worst-Case Soft Actor Critic for Safety-Constrained Reinforcement Learning. In *AAAI*, 10639–10646.

Yu, H.; Xu, W.; and Zhang, H. 2022. Towards safe reinforcement learning with a safety editor policy. *Advances in Neural Information Processing Systems*, 35: 2608–2621.

Zhang, T.; Guo, S.; Tan, T.; Hu, X.; and Chen, F. 2020. Generating adjacency-constrained subgoals in hierarchical reinforcement learning. *Advances in Neural Information Processing Systems*, 33: 21579–21590.

Zhang, Y.; Vuong, Q.; and Ross, K. 2020. First order constrained optimization in policy space. *Advances in Neural Information Processing Systems*, 33: 15338–15349.

A Additional Related Work

Constrained Planning commonly uses classic Dijkstra or Bellman-Ford algorithms with label setting (Han, Kim, and Lee 2014), which is used to label each node with its accumulated resource consumption to ensure the shortest path till now does not violate the constraints. However, as the environment grows bigger, the scale of the problem becomes intractable quickly. Moreover, the graph-based method discretize the continuous state space which may be infeasible in practice, especially in high-dimension environments (Gammell, Srinivasa, and Barfoot 2014). Our use of sampling-based planning algorithm (Karaman and Frazzoli 2011; Gammell, Srinivasa, and Barfoot 2014) avoids discretization of the whole state space which allows them to scale with larger problems and directly consider constraints.

Model based RL techniques (Lenz, Knepper, and Saxena 2015; Mnih et al. 2013; Silver et al. 2016; Watter et al. 2015) typically use a planner after learning aspects of the underlying environent. We note that our learning of local reward and cost provides a high-level local inverse model of the environment. The model learned is coarse as it does not learn detailed prediction of actions or observations. As such, the learning of rewards and costs is also simpler than detailed prediction of actions and transition probabilities (Eysenbach, Salakhutdinov, and Levine 2019). The sampling-based planning algorithm RRT*, without constraints, has been used an alternative choice for the planner (Beker, Mohammadi, and Zamir 2022). But, instead of utilizing distributional RL for accessing reachable points, PALMER (Beker, Mohammadi, and Zamir 2022) employs contrastive learning to extract perceptual representations of states, capturing local reachability. PALMER achieves the shortest path by retrieving neighboring states and trajectories from a pre-collected dataset, and subsequently employs sampled-based algorithms (e.g., PRM and RRT) to perform long-horizon planning by connecting trajectory segments within the dataset. It should be noted that PALMER is more like an offline approach, as it requires a pre-collected dataset fully covering the environment. In contrast, our method does not necessitate a pre-collected dataset and exploration within the environment is facilitated by a low-level reinforcement learning algorithm. Note that none of these methods enforce any constraint.

B Experimental Details and Additional Experiments

The reader might want to read the training process for distributional RL in Appendix D before reading the implementation details.

Implementation Details

In this section, we describe how we performed policy optimization for, the cost estimation of lower level agent and planning for upper level agent. To learn the optimal policy in all maze environment with continuous action space, we use the same RL algorithm DDPG and basic network structure as SORB (Eysenbach, Salakhutdinov, and Levine 2019). To learn the optimal policy in Safe ViDoom environment with discrete action space, we use distributional DQN due to the discrete action space. Due to the highly-dimensional state space of Safe ViZDoom (3x160x120), we employ a pretrained ResNet16 as the image encoder. The ResNet16 is pre-trained using a pre-collected dataset containing 30k (state, position) pairs and fixed when training the policy. the cost estimation for (s, s_g) pairs is performed by the low level RL and hence is reliable for smaller distances.

In Appendix D, we identified two primary sources of representation error in distributional RL: accumulation and the rounding of r to discrete values. To reduce quantization errors when computing the Q function, it's crucial to appropriately determine the first discrete value, the n^{th} discrete value, and the interval between adjacent discrete values. In goal-reach tasks, where the distance is zero upon reaching the goal, the first discrete value is naturally set to 0. Although the smaller the interval between consecutive discrete values is, the smaller the quantization errors are. To simplify calculations, the interval between two consecutive discrete values can be set equal to the reward for each step. By doing this, the distributional Q value only needs a single shift, and our experimental results indicate satisfactory performance. Regarding the n^{th} discrete value: if set too high, the Q fitting process slows down; if set too low, the sampling efficiency of the higher-level agent suffers. This is because the agent can only sample points less than a certain threshold (denoted as η), which is related to the n^{th} discrete value. After adjustments, the n^{th}discrete value was determined to be -20 for 2D navigation tasks and -10 for image-based navigation tasks.

We take a curriculum-based method to further improve the local cost estimate learned by the low level RL agent. s_g is sampled with a hyper-parameter d which shows the distance between s and s_g is no bigger than d. Once the loss of estimator has converged, we slowly increase d until s_g can be freely sampled from the entire environment.

We provide a list of all hyper-parameters in Table 2 and Table 3 We use the original hyper-parameters for the baseline SORB, WCSAC and SAC-lag, respectively. All experiments were run on NVIDIA Quadro RTX 6000 GPUs, CUDA 11.7 with Python version 3.8.16 in Pytorch 1.13.

Additional Experiments

This environment shown in Figure 5a is based on the environment introduced in WCSAC (Yang et al. 2021) and is similar to the point navigation task in Safty Gym (Ray, Achiam, and Amodei 2019b). The scale of the environment is 40 by 40, and there is a circular hazard of radius 16 in the center of the environment. The goal is fixed at $(38, 38)$, while the agent starts from near the left bottom corner $(0, 0)$ of the environment randomly ($x \leq 20$ and $y \leq 20$). In each step, if the agent stays in the hazardous area, it incurs a cost $c = 1$, otherwise $c = 0$. We test SAC-Lagrangian (SAC-lag) (Yang et al. 2021; Stooke, Achiam, and Abbeel 2020), WCSAC (Yang et al. 2021), SORB (Eysenbach, Salakhutdinov, and Levine 2019), and Goal-conditioned RL (GRL) (Kaelbling 1993) with $K = 20$ which is the cost limit. Note that SORB and GRL are not

Table 2: Hyper-parameters used in all Maze tasks

Hyper-parameters	**Values**
Policy optimization	
Actor learning rate	0.0003
Critic learning rate	0.0003
Collect steps per optimization	1
Replay buffer size	1000
Batch size	128
Soft update rate	0.005
Target update frequency	5
Actor update frequency	1
discount factor	1
Exploration strategy	Gaussian($\sigma = 1.0$)
Cost estimation	
Collect steps per optimization	10
Replay buffer size	100000
Batch size	256
Critic learning rate	0.0003
Planning	
η	2
r_{RRT*}	5

Table 3: Hyper-parameters used in Safe ViZDoom tasks

Hyper-parameters	**Values**
Policy optimization	
Image encoder	ResNet16
QNetwork learning rate	0.00025
Collect steps per optimization	2
Replay buffer size	2000
Batch size	64
Soft update rate	0.005
Target update frequency	500
discount factor	1
Exploration strategy	Epsilon ($\epsilon = 0.2$)
Cost estimation	
Collect steps per optimization	10
Replay buffer size	100000
Batch size	256
Critic learning rate	0.0003
Planning	
η	2
r_{RRT*}	5

designed to enforce constraints. All of the agents are fully trained so that the success rate of each agent is 100%.

After training, we run 100 trials for each policy of these algorithms. The length of trajectories for each policy is represented by negated reward (lower is better) in Figure 5b. Our method achieves comparable results with SAC-lag while outperforming WCSAC with the different risk levels (WCSAC-0.1 is more risk-averse while WCSAC-0.9 is more risk-neutral). Since SORB and GRL do not enforce constraints, the paths output by these go directly through the hazardous area, hence the length of paths of SORB and GRL are shorter than all the other constrained policies.

In the boxplot in Figure 5c, we compare the cost distribution of 100 trials for each policy. The whiskers are set as $1.5 * (Q3 - Q1)$ where $Q3$ and $Q1$ are the upper edge and lower edge of the box separately. As expected, most trials of unconstrained policies SORB and GRL violate the cost constraints, while only a few trials of our method CoSHRL, as well as a few trials of SAC-lag and WCSAC exceed the cost limit. It is worth noting that all of the trials of WCSAC-0.1, which is highly risk-averse, avoid the hazardous area completely and its cost is much lower than the cost threshold, but the closer the cost gets to the cost threshold without exceeding the threshold, the shorter the trajectory should be. Thus, the conservative WCSAC-0.1 make sure that no trajectories violate the constraints but the worst cases of these trajectories may be caused by the exploratory policy in the training stage, and these worst cases do not occur in the testing stage with the deterministic policy, hence WCSAC-0.1 has worse rewards in Figure 5b. Our method is not affected by this gap benefiting from we estimate the cost function in a supervised manner after the low-level policy is learned.

Additionally, in Figure 6, we provide examples of paths generated by various approaches within the point maze environment mentioned in the main paper.

C Asymptotic Optimality of CoSHRL

The upper-level planning agent is derived from Informed RRT* which is asymptotically optimal (Gammell, Srinivasa, and Barfoot 2014), i.e., more samples and iterations N would result in discovery of the shortest path. To show that our method inherits this property, we evaluate our method with different numbers of iterations at difficulty level $\alpha = 0.5$ in Figure 7. We observe the gap between our method and goal-conditioned RL becomes smaller as the number of iterations (N in Algorithm 1) increases. Due to the low success rate of goal-conditioned RL at high difficulty levels, we don't show the comparison at higher difficulty levels. Based on these observations, our method is asymptotically optimal with a high success rate in the long-horizon task.

D Training of Distributional RL

Recall that is standard RL updation the Q-value function with scale factor $\gamma = 1$ given action $a \in A$ in state s that leads to a state s':

$$Q(s,a) = r(s,a) + \max_{a' \in A} Q(s', a') \tag{5}$$

Then we minimize the difference between current $Q^{\theta}(s,a) = Q(s,a)$ and the target $Q^{t}(s,a) = r(s,a) + \max_{a' \in A} Q(s', a')$.

As we care about cost distribution (due to cost constraint in Eq. (2)) , we need to estimate the distribution of $\mathbf{V}_c^{\pi}$ for the π that maximizes expected $V^{\pi}(s,s')$ for nearby s, s'. While it would suffice to use a standard RL method to maximize expected $V^{\pi}(s,s')$, it is known from literature that learning the distribution of $\mathbf{V}^{\pi}$ and then calculating expected value leads to better estimates (Eysenbach, Salakhutdinov, and Levine 2019). Thus, we use distributional RL for both

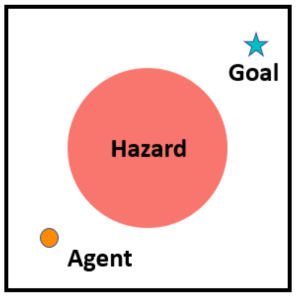

Figure 5(a): The simple point navigation environment

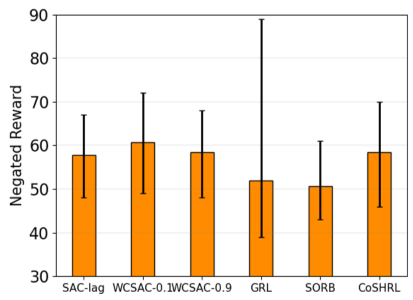

Figure 5(b): Negated reward for SAC-lag, WCSAC, GRL, SORB and our method over 100 trials in evaluation.

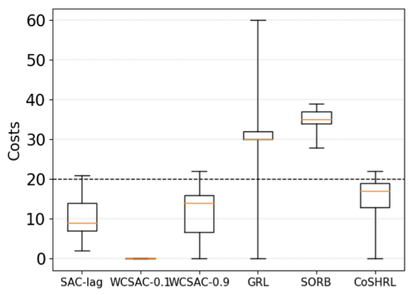

Figure 5(c): Boxplot of cost in evaluation after training. The dashed line indicates the cost limit d.

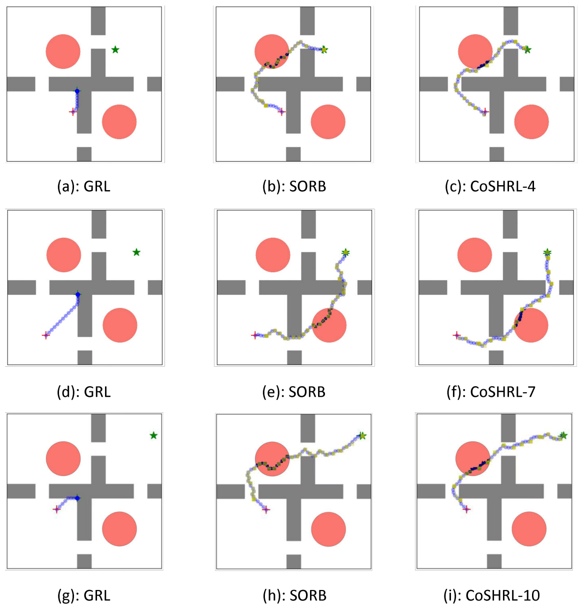

Figure 6: Typical trajectories for static cost in the complex environment. (a)-(c) are generated by GRL, SORB, and CoSHRL with cost limit $K=4$, (d)-(f) with $K=7$, and (g)-(i) with $K=10$. GRL gets stuck in corners and SORB goes directly through hazards without considering cost. CoSHRL completely avoids the hazard for $K = 1$ and goes deeper into hazard for a better solution with higher K.

reward and cost distribution estimation. Distributional RL learns a policy $\hat{\pi}$ and in the process maintains a network that represents the distribution of $\mathbf{Q}$ which we describe next; after that we show how to derive $\mathbf{V}$, $\mathbf{V_c}$ from the learned policy $\hat{\pi}$.

Distributional value functions: We use distributional RL (Bellemare, Dabney, and Munos 2017) to estimate the distribution of the goal conditoned reward random variable $\mathbf{Q}$ (Eysenbach, Salakhutdinov, and Levine 2019). In distributional RL, the possible value estimates are assumed to be one of N discrete values. In goal reaching tasks the rewards are non-positive (only 0,-1 in the code, but we write below for general negative reward) The distribution of $\mathbf{Q}$ is represented by Q^θ (neural network parameterized by θ), which

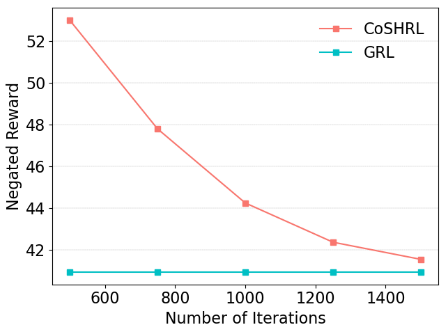

Figure 7: The performance of our method with a varying number (N) of iterations.

takes as input s, s_g, a and outputs a vector $[p_1, \ldots, p_N]$ where p_i is the probability of expected reward value taking the i^{th} discrete value, with p_1 corresponding to first bin of 0 value. We write $-r(s, a) = i$ to mean that the immediate reward takes the i^{th} discrete positive value. We also abuse notation in the distributional RL setting to use $Q(\cdot, \cdot, \cdot)$ to represent a vector of probabilities instead of a scalar number.

Using distributional RL in goal-reaching tasks, the current $Q^\theta(s, s_g, a) = [p_1, \ldots, p_N]$, a target Q^t is defined as

$$Q^t(s, s_g, a) = \begin{cases} [1, 0, ..., 0] \text{ for } s = s_g \\ [f([p'_1, \ldots, p'_N], i) \text{ for } s \neq s_g \wedge -r(s, s_g, a) = i \end{cases} \quad (6)$$

$[p'_1, \ldots, p'_N]$ is probability vector of the next state s' and action $a^* \in \text{argmax}_{a' \in A} \mathbb{E}(Q^\theta(s', s_g, a'))$, which is the optimal Bellman update in distributional RL. In the above function, f denotes right shift of probability distribution based on the reward with accumulation in the rightmost bin. More formally, $f([p_1, \ldots, p_N], 1) = [0, p_1, \ldots, p_{N-1}, p_{N-1} + p_N]$ denotes right shift when reward is 1 and recursively $f([p_1, \ldots, p_N], i) = f(f([p_1, \ldots, p_N], i-1), 1)$. Adding a positive number to a distribution is same as shifting the distribution by that number, but due to finite representation, we accumulate the probability in the last bin (see Figure 8); this accumulation and the rounding of r to discrete values form the two sources of representation error in distributional RL (Bellemare, Dabney, and Munos 2017; Ey-

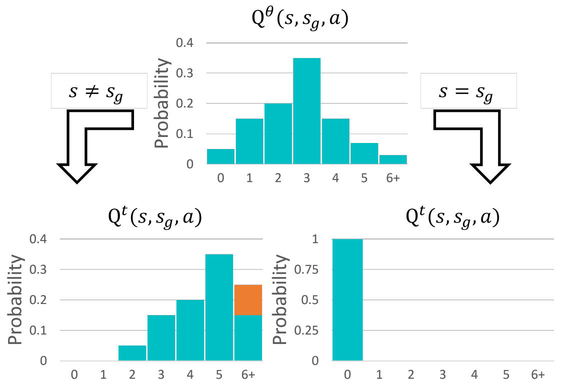

Figure 8: An illustration of updating the distributional Q value with $N = 7$ and $r(s, a) = -2$.

senbach, Salakhutdinov, and Levine 2019). Then we update Q^θ by minimizing the KL-divergence between the target and current distribution $\min_\theta D_{KL}(Q^t||Q^\theta)$. The above approach yields a trained policy $\hat{\pi}$. For training, we choose nearby start and goal states at random throughout the state space, ensuring that the estimates generalize to any pair of nearby start and goal in the whole state space.

Next, we represent the distribution of value $\mathbf{V}^{\hat{\pi}}$ as a neural network V^w, which again outputs a probability vector. For simplicity, we do not include the learned policy, $\hat{\pi}$ (which will not change) in the notation for value neural network, V^w. The fixed learned policy $\hat{\pi}$ allows us to estimate V^w directly by minimizing the KL divergence between a target $V^t(s, s_g) = Q^\theta(s, s_g, a), a \sim \hat{\pi}(\cdot|s, s_g)$ and the current V^w (Schaul et al. 2015): $\min_\omega D_{KL}(V^t||V^w)$. We optimize the above by storing experiences sampled according to $\hat{\pi}$ in a replay buffer and sampling mini-batches to minimize the loss above, analogous to supervised learning. Once the vector of probabilities V^w is obtained, we can obtain expected V by calculating the expectation.

For costs, we note that we performed the reward estimation without considering costs since in our approach the lower-level agent does not enforce constraints. However, the lower level agent does estimate the local costs and learn the cost distribution V_π^c. Again, similar to above learning of V^w, the fixed learned policy $\hat{\pi}$ allows us to estimate the vector of probability V_c^w function directly by minimizing the KL divergence between the target $V_c^t(s, s_g) = Q_c^\theta(s, s_g, a), a \sim \hat{\pi}(\cdot|s, s_g)$ and the current V_c^w, that is, $\min_\omega D_{KL}(V_c^t||V_c^w)$. where $Q_c^\theta(s, s_g, a)$ is estimated by another neural network different from the one used in estimating the distribution of Q (for simplicity, we both use θ to represent neural network's parameter). We learn $Q_c^\theta(s, s_g, a)$ similar to the learning of $Q^\theta(s, s_g, a)$ by minimizing the KL-divergence between the target and current cost distribution $\min_\theta D_{KL}(Q_c^t||Q_c^\theta)$ but this is after the policy $\hat{\pi}$ has been learned. Thus, this is learned using a policy evaluation Bellman update only rather than optimal control Bellman update.

$$Q_c^t(s, s_g, a) = \begin{cases} [1, 0, ..., 0] \text{ for } s = s_g \\ [f([p_1', \ldots, p_N'], i) \text{ for } s' \neq s_g \wedge c(s, s_g, a) = i \end{cases} \tag{7}$$

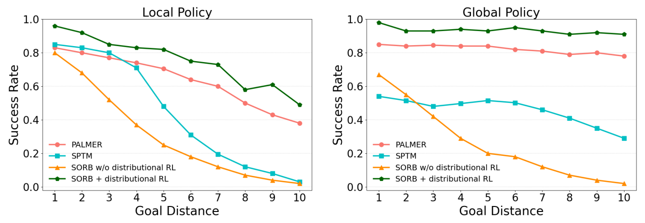

Figure 9: Comparision bewteen SORB and PALMER. Distributional RL makes the value estimation more accurate so that SORB+distibutional RL has higher success rate and outperform all the other baselines.

with $[p_1', \ldots, p_N'] = Q_c^\theta(s', s_g, a')$ where $a' \sim \hat{\pi}(s', s_g)$

E SORB v.s. PALMER

Distributional reinforcement learning (RL) plays a crucial role in the performance of SORB (Eysenbach, Salakhutdinov, and Levine 2019, Section 3.1). PALMER implemented the SORB baseline in that paper with standard RL, not the distributional version. In fact, the authors of PALMER do acknowledge this, and quoting them:

> Beker, Mohammadi, and Zamir (2022, Page 8 in supplementary material): The main differences of our SoRB implementation are: i) we use an ensemble of Q-functions, but they are trained with DDQN rather than distributional RL, ii) we train the Q-function on offline random-walk data, rather than an online episodic training setup with resets and a reward oracle as employed in the original paper. We acknowledge and emphasize that for SoRB, these differences are the most likely reason for the lower performance level we observed in our evaluations compared to the original paper, as they inevitably reduce the accuracy of Q-values.

In order to compare the distributional RL version of SORB and PALMER, we use the original version of SORB with distribution RL (Eysenbach, Salakhutdinov, and Levine 2019, Section 3.1) and use it on the same exact setting used to produce Figure 4 in the PALMER work (Beker, Mohammadi, and Zamir 2022, Figure 4). Following their approach, in Figure 9, the local policy refers to the policy without high-level planning, whereas the global policy incorporates the high-level planning algorithm. **It can be seen that distributional RL significantly enhances the success rate so that SORB outperforms all the other baselines used in PALMER, including PALMER itself, in both the local and global policies**. As discussed above, this improvement stems from the more accurate distance estimation provided by the distributional RL approach. We do wish to highlight that we directly used the reported number for PALMER and SPTM from the PALMER paper itself in Figure 9, **PALMER and SPTM were not implemented by us as the provided code is missing a pre-trained model.**

F Proofs

Proof of Lemma 1

Proof. For ease of notation, we use V instead of $-V$. By the lemma assumption, V gives the shortest length of the path with obstacles (in a path search problem with deterministic movement). We need to show the three properties of a distance metric:

- $V(s,s) = 0$: this is true as the shortest path between same point is 0.
- $V(s,s') = V(s',s)$: this is true as the shortest path is direction independent in the state space $S \subset \mathbb{R}^d$.
- $V(s,s') \leq V(s,t) + V(t,s')$: this is true because otherwise if $V(s,s') > V(s,t)+V(t,s')$ then the path from s to s' via t has shorter length than one given by $V(s,s')$, which contradicts the assumption that V gives the length of shortest path.

□

Proof of Theorem 1

This proof relies heavily on the proof of asymptotic optimality of RRT* in (Karaman and Frazzoli 2011). We ask the reader to first revisit that proof, as the proof spans many pages. In particular, Algorithm 6 (RRT*), Section 4.2 (introduces asymptotic optimality), Lemma 50, Definition 51, and Appendix G (proof of asymptotic optimality of RRT*). A background on Poisson point process is also present in the paper and is needed to understand the proof in Appendix G of (Karaman and Frazzoli 2011). As the proof in (Karaman and Frazzoli 2011) is very long and complicated, we next provide an overview of the proof in order to guide a reader through the proof in (Karaman and Frazzoli 2011).

1. By Lemma 50, the authors show that always there exists such a sequence of paths σ_n with $\delta_n > 0$ clearance converging to the optimal path σ^* (optimal path may not have non-zero δ clearance), where δ_n is defined as $\min(\delta, 4r_n)$ in Appendix G.2 ($r_n = \gamma_{RRT^*}(\frac{\log n}{n})^{1/d}$ is the rewiring radius).
2. A marked point process is defined in Appendix G.1. Let $\{X_1, X_2, ..., X_n\}$ be a independent uniformly distributed points drawn from S_{free} and let $\{Y_1, Y_2, ..., Y_n\}$ be independent uniform random variables with support [0, 1]. Each point X_i is associated with a mark Y_i that describes the order of X_i in the process. More precisely, a point X_i is assumed to be drawn after another point $X_{i'}$ 0 if $Y_{i'} < Y_i$. Later on, this marked point process is analyzed.
3. The equivalence of the marked point process to the RRT* Algorithm is also shown in Appendix G.1. Put an edge $(X_{i'}, X_i)$ whenever $Y_{i'} \leq Y_i$ and $||X_{i'} - X_i|| \leq r_n$ to get graph G_n. ($r_n = \gamma_{RRT^*}(\frac{\log n}{n})^{1/d}$ is the rewiring radius). Edges are removed from G_n to form G'_n by keeping only those edges from parent nodes that come the parent with shortest path from source. Thus, the distance of the shortest path in G'_n will be the same as any shortest paths in G_n. It is shown that G'_n is the same graph that RRT* would return after n samples.
4. Next, a sequence of balls $B_{n,1}, \ldots, B_{n,M_n}$ covering the path σ_n is chosen in Appendix G.2 of Karaman and Frazzoli (2011) (the ball covering is defined in Definition 51). Each ball is of radius r_n, situated $2r_n$ apart.
5. It is shown that marked point process results in some number of points in each of the balls such that there exists an edge between two such points in consecutive balls with probability one as $n \to \infty$ (in Lemma 71).
6. Finally, in Appendix G.4, in Lemma 72 it shown that the closest path σ'_n (in the BV norm) produced by RRT* to σ_n converges to σ^*, i.e., $\lim_{n\to\infty} \sigma'_n = \sigma^*$ and hence by robustness assumption $\lim_{n\to\infty} R_{\sigma_{n'}} = R_{\sigma^*}$.

Assumption: Finally, we impose one more mild restriction. If for a given path σ_n with cost K covered by balls $B_{n,1}, \ldots, B_{n,M}$ the radius of balls are less than a constant threshold τ (very small number), then the cost of any path constructed by joining points within consecutive balls is less that $K + f(\tau)$ where f is a continuous strictly monotonic function of τ and $f(0) = 0$.

Proof. In this proof, we will modify certain aspects of the prior RRT* proof. Thus, the reader must read the proof in Karaman and Frazzoli (2011) (for which we have provided a summary and guidance above). Like the proof in Karaman and Frazzoli (2011), we start by denoting the optimal path as σ^* but in our constrained case we additionally impose that the cost of this path $C_{\sigma^*} \leq K - \epsilon$, where ϵ is any small number. The cost for any path output by our algorithm (for any n) is always less than K due to the Valid_edge check, but we need to make sure that this constraint does not prevent the convergence in the reward to the optimal reward within constraints. This is what we show next. We run ConstrainedRRT* with cost threshold K and show asymptotic optimality with constraint K in the limit.

First, borrowing from the proof in Appendix G of (Karaman and Frazzoli 2011), we use the same marked process in Appendix G.1 where the criteria for connecting two sampled points to make an edge $(X_{i'}, X_i)$ is $Y_{i'} \leq Y_i$ and $||X_{i'} - X_i|| \leq r_n$ to get graph G_n (restated in point (3) in our summary of the same proof). By construction G_n is a directed graph with no cycles.

After G_n is constructed we impose our criteria that the shortest path to X_i must incur a cost $\leq K$. Remove all such nodes that do not satisfy this criteria to get graph G^m_n. This modified construction ensures that the graph G^m_n has no path greater than cost threshold K.

After this, perform the edge removal as stated in the proof in Karaman and Frazzoli (2011) (restated in point (3) in our summary of the same proof) to get G'_n from G^m_n. Thus, the distance of the shortest path in G'_n will be the same as any shortest paths in G^m_n. Let Y^m_n denote the reward of the best path in G^m_n and Y'_n denote the reward of the best path in G'_n, we have $\limsup_{n\to\infty} Y^m_n = \limsup_{n\to\infty} Y'_n$. Also, let Z^m_n denote the cost of the best path in G^m_n and Z'_n denote the cost of the best path in G'_n. By construction, $Z^m_n \leq K$ and $Z'_n \leq K$.

Hence G'_n is equivalent to the graph produced by ConstrainedRRT* with cost threshold K.

Next, we redefine δ_n as follows

$$\delta_n = \min(\delta, 4r_n, \eta, \tau)$$

This δ_n is the strong clearance of path σ_n, where with $n \to \infty$ σ_n converges to σ^* (the existence is guaranteed by Lemma 50 in (Karaman and Frazzoli 2011)). As $\lim_{n\to\infty} r_n = 0$, there exists a n_0 such that for all $n \geq n_0$ we have $\delta_n = 4r_n$ since δ, η, τ are constants. This ensures that the balls $B_{n,1}, \ldots, B_{n,M_n}$ that cover the paths σ_n (for $n \geq n_0$) are all obstacle free (as σ_n has δ_n clearance and the radius of the balls are chosen as $r_n = \delta_n/4$). As the balls are obstacle free, even in our distance metric with obstacles (Lemma 1) these are exactly the same as balls in the Euclidean norm. Further, as the cost of σ_n path (for some large enough n) is $\leq K\epsilon/2$ for any choice of ϵ (by our extended definition of δ clearance, since δ^* has cost of $K - \epsilon$) including the ϵ we chose at the start and the balls are radius $\leq \tau$, any path by joining two points in consecutive balls will be of cost $\leq K - \epsilon/2 + f(\tau)$. Taking τ small enough, for any ϵ this will be $\leq K - \epsilon/4$, that is, any path via points in consecutive balls has cost bounded by $K - \epsilon/4$.

Next, Lemma 71 in Appendix G.3 is shown for graph G_n in Karaman and Frazzoli (2011). The lemma basically shows that in the limit of n, there will always be a path (with probability 1) with consecutive nodes through the balls $B_{n,1}, \ldots, B_{n,M_n}$. In our case, the same lemma continues to hold for G_n^m since any path via each consecutive ball costs less than $K - \epsilon/4$ (as argued above), and hence any such path will also be present in G_n^m.

Lemma 72 also holds for us as all paths via each consecutive ball covering σ_n costs less than $K - \epsilon/4$ and hence for Lemma 72 (see Lemma 55 where the same sequence of balls in used), we do not need to bother about the cost of the path, and hence Lemma 72 holds for ConstrainedRRT* also. Thus, Lemma 72 shows that σ'_n (that path in the Constrained RRT* graph that is closest to σ_n) has cost less than $K - \epsilon/4$ and converges to σ_n. Finally, with $\lim_{n\to\infty} \sigma'_n = \sigma^*$ we get our required result due to the robustness assumption for reward, i.e., $\lim_{n\to\infty} R_{\sigma'_n} = R_{\sigma^*}$ and for cost, i.e., $\lim_{n\to\infty} C_{\sigma'_n} = C_{\sigma^*} \leq K - \epsilon/4$ for any ϵ. Also, any path output by our algorithm (for any n) is always less than K due to the Valid_edge check.

Thus, overall, as we have proved the results for any ϵ, from basic real analysis (Rudin 1987), this implies the result also holds for σ^* such that $C_{\sigma^*} \leq K$.

□

G Full Algorithm

We provide a more detailed algorithm and algorithm description than the main paper. Parts of this section is same as the main paper, the change is mainly in Algorithm 4 which is a detailed version of Extend_node. These algorithms are also pseudocode and the code provided has the complete details.

Algorithm Description: We propose Constrained RRT* (Algorithm 3), which builds on Informed RRT* to handle the cost constraint. The pseudocode is provided in Algorithm 3. We search for the optimal path σ^* to a planning problem by incrementally building a tree $\mathcal{T}$ in the state space S. The tree, $\mathcal{T}$ consists of a set of nodes, $\mathcal{V}$ ($\subset S$), and edges $\mathcal{E}$ ($\subset S \times S$).

Algorithm 3: ConstrainedRRT* $(s_o, s_G, V, \mathbf{V}_c, K)$

```
1 $\mathcal{V} \leftarrow \{s_o\}, \mathcal{E} \leftarrow \emptyset$ , $S_{soln} \leftarrow \emptyset, \mathcal{T} = (\mathcal{V}, \mathcal{E})$
2 for iteration = 1 ... N do
3     $\mathcal{T}$ = Extend_node$(s_o, s_G, V, \mathbf{V}_c, K, \mathcal{T}, S_{soln})$
4 return best solution in $S_{soln}$
```

A candidate state s_{new} is chosen (line 4) to be added to the tree $\mathcal{T}$ by sampling s_{rand} (line 2) within an ellipse with focal points as s_o and s_G and axis lengths as r_{best} and $\sqrt{r_{best}^2 - r_{min}^2}$, where r_{best} (line 1) is the length of the current best solution in S_{soln} or infinite if S_{soln} is empty . In case s_{rand} is far from the nearest node $s_{nearest}$ in $\mathcal{T}$ (line 3), then the Steer function in line 6 chooses a point s_{new} within the $\min(r_{RRT*}, \eta)$ distance ball centered at $s_{nearest}$ that is also the closest to s_{rand}. This hyper-parameter η accounts for the fact that our distance estimates are precise only locally. The rewiring radius, $r_{RRT*} = \gamma_{RRT*}(\log n/n)^{1/d}$, where n is the current number of nodes sampled, is described in (Karaman and Frazzoli 2011). Then, if the candidate edge $(s_{nearest}, s_{new})$ is valid then other nodes (set S_{near} in line 5) that are within distance $\min(r_{RRT*}, \eta)$ of s_{new} are considered as possible connections to s_{new}. The node s_{min} (line 8-12) that results in shortest path (highest reward) is connected to s_{new} in line 14, if the edge is valid.

Here, we take a detour to explain how we determine the validity of edges. An edge is valid if and only if adding it does not result in a (partial) path that violates the cost constraint. The key insight is that this validity can be determined by computing the convolution of the distributions associated with the (partial) path and the current $\mathbf{V}_c$. By providing the definition of Valid_edge $(\mathcal{T}, s, s', \mathbf{V}_c, K)$ in Algorithm 2 and doing the Valid_edge checks in lines 9, 13 and 16 of the Algorithm 4, we ensure that any path output by the overall algorithm will satisfy the cost constraints. In the pseudocode of Valid_edge $(\mathcal{T}, s, s', \mathbf{V}_c, K)$, $\mathbf{V}_c$ represents a random variable (and so does $result$). Then, the addition in line 3 of Valid_edge is a convolution operation (recall that the distribution of a sum $\mathbf{X} + \mathbf{Y}$ of two random variables $\mathbf{X}, \mathbf{Y}$ is found by a convolution (Ross 2014)). An example planner run is shown in Figure 1.

Coming back to the main Algorithm 4, we explore further the possible edges to be added to the tree. In particular, in the loop in lines 17-23 (1) the edge is created only if it is valid and (2) new edges are created from s_{new} to vertices in S_{near}, if the path through s_{new} has lower distance (higher reward) than the path through the current parent; in this case, the edge linking the vertex to its current parent is deleted, to maintain the tree structure. InGoalRegion on line 25 tracks if we are close to the goal.

Algorithm Complexity

The complexity can be divided into two parts: construction complexity and query complexity. Normally we only focus on query complexity. However, the construction complexity

Algorithm 4: Extend_node $(s_o, s_G, V, \mathbf{V}_c, K, \mathcal{T}, S_{soln})$

```
1  $r_{best} \leftarrow \min_{\sigma \in S_{soln}}\{-R_\sigma\} \cup \{\infty\}$
2  $s_{rand} \leftarrow$ Sample$(s_o, s_G, r_{best})$
3  $s_{nearest} \leftarrow$ Nearest$(\mathcal{T}, s_{rand})$
4  $s_{new} \leftarrow$ Steer$(s_{nearest}, s_{rand}, \min(r_{RRT*}, \eta))$
5  $S_{near} \leftarrow$ Near$(\mathcal{T}, s_{new}, \min(r_{RRT*}, \eta))$
6  $s_{min} \leftarrow s_{nearest}$
7  $r_{min} \leftarrow -R(s_{min}) - V(s_{nearest}, s_{new})$
8  for $\forall s_{near} \in S_{near}$ do
9      if Valid_edge$(\mathcal{T}, s_{near}, s_{new}, \mathbf{V}_c, K)$ then
10         $r_{new} \leftarrow -R(s_{near}) - V(s_{near}, s_{new})$
11         if $r_{new} < r_{min}$ then
12             $s_{min} \leftarrow s_{near}, r_{min} \leftarrow r_{new}$
13 if Valid_edge$(\mathcal{T}, s_{min}, s_{new}, \mathbf{V}_c, K)$ then
14     $\mathcal{V} \leftarrow \mathcal{V} \cup \{s_{new}\}$ , $\mathcal{E} \leftarrow \mathcal{E} \cup \{(s_{min}, s_{new})\}$
15 for $\forall s_{near} \in S_{near}$ do
16     if Valid_edge$(\mathcal{T}, s_{new}, s_{near}, \mathbf{V}_c, K)$ then
17         $r_{near} \leftarrow -R(s_{near})$
18         $r_{new} \leftarrow -R(s_{new}) - V(s_{new}, s_{near})$
19         if $r_{new} < r_{near}$ then
20             $s_{parent} \leftarrow$Parent$(s_{near})$
21             $\mathcal{E} \leftarrow \mathcal{E} \backslash \{(s_{parent}, s_{near})\}$
22             $\mathcal{V} \leftarrow \mathcal{V} \cup \{s_{new}\}$
23             $\mathcal{E} \leftarrow \mathcal{E} \cup \{(s_{new}, s_{near})\}$
24 if InGoalRegion$(s_{new})$ then
25     Form $\sigma$ by tracking parents of $s_{new}$
26     $S_{soln} \leftarrow S_{soln} \cup \{\sigma\}$
27 return $\mathcal{T}$
```

cannot be ignored in SORB's and our case as the distance of the edge in calculated by neural networks which is quite time-consuming. As described by SORB, all edges need to be calculated and those which satisfy certain condition are temporarily added into the graph to the graph (Eysenbach, Salakhutdinov, and Levine 2019). Therefore, the construction complexity of SORB is $O(N^2)$ for N nodes. Because of the usage of tree structure in RRT*, the construction complexity is $O(N \log N)$ (Karaman and Frazzoli 2011). For the query complexity, our method is only $O(N)$ because the parent of each node is recorded while constructing the tree itself. SORB uses Dijkstra's Algorithm to find the shortest path after the graph is constructed, therefore they have $O(N \log N)$ querying complexity. In conclusion, SORB and CoSHRL have $O(N^2)$ and $O(N \log N)$ complexity, respectively.

Algorithm 5: Valid_edge $(\mathcal{T}, s, s', \mathbf{V}_c, K)$

```
1 $result \leftarrow \mathbf{V}_c(s, s')$
2 while $s.parent$ do
3     $result \leftarrow result + \mathbf{V}_c(s.parent, s)$
4     $s \leftarrow s.parent$
5 if $CVaR_\alpha(result) \leq K$ then
6     return True
7 return False
```