Hierarchical Imitation Learning with Vector Quantized Models

Kalle Kujanpää [1 2] **Joni Pajarinen** [3 2] **Alexander Ilin** [1 2]

Abstract

The ability to plan actions on multiple levels of abstraction enables intelligent agents to solve complex tasks effectively. However, learning the models for both low and high-level planning from demonstrations has proven challenging, especially with higher-dimensional inputs. To address this issue, we propose to use reinforcement learning to identify subgoals in expert trajectories by associating the magnitude of the rewards with the predictability of low-level actions given the state and the chosen subgoal. We build a vector-quantized generative model for the identified subgoals to perform subgoal-level planning. In experiments, the algorithm excels at solving complex, long-horizon decision-making problems outperforming state-of-the-art. Because of its ability to plan, our algorithm can find better trajectories than the ones in the training set.

1. Introduction

Learning from expert demonstrations has proven successful in many sequential decision-making settings that can be modeled with Markov decision processes (Abbeel & Ng, 2004). Imitation learning (IL) is a technique for learning to imitate the behavior of an expert by discovering the mapping between states and actions without access to information such as rewards (Osa et al., 2018). IL has proven useful in aviation (Sammut et al., 1992), autonomous driving (Chen & Krähenbühl, 2022), robotics (Kober & Peters, 2008), video games (Vinyals et al., 2019) and even healthcare (Mayer et al., 2008).

Recent advances in planning with learned dynamics models have improved our ability to solve complex long-horizon problems when interacting with the environment is possible (Hafner et al., 2022; Ye et al., 2021; Schrittwieser et al., 2020). Learning models for planning in the offline setting is possible as well (Argenson & Dulac-Arnold, 2021), but these model-based reinforcement learning (RL) methods assume access to environment rewards and are not directly applicable to the IL setting. Hierarchical RL with decision-making at multiple time scales has succeeded in tasks where flat RL struggles (Hafner et al., 2022; Nachum et al., 2019). A hierarchy should also be useful in the IL setting as many real-world problems have a natural hierarchical structure (Sharma et al., 2019b; Jing et al., 2021). Moreover, planning with a hierarchy may shorten the effective planning horizon and avoid compounding model errors also in the IL setting (Nair & Finn, 2020).

We propose a method for hierarchical planning in the IL setting that relies on segmenting expert trajectories into subtasks without any high-level supervision. Unlike prior work that assumes a fixed number of subgoals (Pertsch et al., 2020b), fixed-length subtasks (Czechowski et al., 2021), or trains multiple models to deal with subtasks of different lengths (Zawalski et al., 2022), our algorithm segments the trajectories into a variable number of variable-length subtasks. We use the segmentation to learn a generative model over the subgoals and a subgoal-conditioned low-level policy to execute the subtasks. To perform high-level planning, we use standard search methods such as Policy-Guided Heuristic Search (Orseau & Lelis, 2021), Monte Carlo Tree Search (Coulom, 2006), or A* (Hart et al., 1968) in which our generative model is used for node expansion. Our method outperforms strong search, hierarchical IL, and offline RL algorithms at complex long-horizon decision-making problems. Our experiments also show that our algorithm can handle suboptimal expert trajectories and self-improve. In summary, the main contributions of this work are:

1. A novel yet conceptually simple RL approach for identifying subgoals from trajectories based on the prediction performance of a low-level policy.
2. A VQVAE generative model for proposing subgoals for planning with temporal abstraction.
3. In experiments, our generative model combined with the learned low-level policy and a suitable high-level search algorithm solves complex problems with sparse rewards better and with fewer node expansions than

[1]Department of Computer Science, Aalto University, Finland [2]Finnish Center for Artificial Intelligence FCAI [3]Department of Electrical Engineering and Automation, Aalto University, Finland. Correspondence to: Kalle Kujanpää <kalle.kujanpaa@aalto.fi>.

Proceedings of the 40th International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

state-of-the-art subgoal search and outperforms offline RL algorithms.

2. Related Work

Our method combines hierarchical discrete planning with imitation learning to solve complex planning problems. Hierarchical planning allows shortening the effective planning horizon, which is beneficial in long-horizon tasks (Pertsch et al., 2020a; Nair & Finn, 2020). We use offline data to learn a VQVAE (Van Den Oord et al., 2017) that generates adaptive horizon subgoals and a low-level policy to reach the subgoals for high-level planning. Previous work proposes a wide variety of different approaches for learning hierarchical behavior from data, also in combination with planning. However, according to our knowledge, our method is the first method that segments a trajectory into a varying number of adaptive-length subtrajectories and uses the subgoals to learn a generative model that can solve difficult long-term decision-making problems in a discrete setting.

Hierarchical RL. Sutton et al. (1999) proposed using options, a set of low-level policies with termination, for decision-making at a higher level of temporal abstraction than standard RL. Options can be learned end-to-end without supervision (Bacon et al., 2016; Bagaria & Konidaris, 2020; Sharma et al., 2020) and applied to planning (Silver & Ciosek, 2012). Representing the hierarchy as subgoals is an alternative to options (Dayan & Hinton, 1992), which is the approach we adopt in our work. Our work differs from these methods in that we assume that we cannot interact with the environment and must learn the representation from a given dataset.

Hierarchical Continuous Planning. In hierarchical planning with continuous control, the Cross-Entropy Method (CEM) is typically used. With continuous controls, search methods with convergence guarantees such as MCTS (Coulom, 2006), PHS* (Orseau & Lelis, 2021), or A* (Hart et al., 1968), utilized with our method in the experiments, cannot be directly applied. HiGoC (Li et al., 2022) uses CEM for hierarchical planning in offline RL assuming access to rewards. HVF (Nair & Finn, 2020) adds a hierarchical structure with predicted subgoal images to Visual Model Predictive Control (Ebert et al., 2018) that plans in image space using CEM. Visual hierarchical methods that rely on identifying keyframes from trajectories using variational inference and planning with CEM include TAP (Jayaraman et al., 2019) and KeyIn (Pertsch et al., 2020b). However, these methods assume a fixed number of keyframes in the trajectories and have not been successfully applied to highly complex reasoning tasks with sparse rewards. KeyIn also uses an environment simulator for planning.

Goal-Conditioned Hierarchical Planning (Pertsch et al., 2020a) produces plans, executed by a learned inverse dynamics model, in a high-dimensional state space in an offline setting. SGT-PT does goal-based RL in a low-dimensional setting by planning with subgoal trees (Jurgenson et al., 2020). However, these models require an explicit goal state and need to generate subgoals between the initial state and the goal state, which can be a difficult learning problem in complex long-horizon environments.

Hierarchical IL without Planning. Unlike our method, many model-free hierarchical imitation learning methods assume some degree of high-level supervision (Le et al., 2018; Fox et al., 2019; Zhang et al., 2021). As an alternative Daniel et al. (2016) infer compositional structure in data discovering options with expectation-maximization. CompILE (Kipf et al., 2019) uses VAEs to segment trajectories into subtasks and the subtask encodings as subpolicies in hierarchical RL. The model-free method Option-GAIL (Jing et al., 2021) infers expert options from trajectories with an EM-like approach. Zhang & Paschalidis (2021) directly optimize a hierarchical policy with options by maximizing the probability of expert trajectories with a hidden Markov model. Directed-Info GAIL (Sharma et al., 2019a) is a variant of hierarchical inverse RL that learns latent policies by modeling problems as directed graphs. The method segments expert trajectories into sub-tasks and learns structural policies to solve different sub-tasks. OptionGAN (Henderson et al., 2018) learns to recover reward and policy options simultaneously. Learning from Guided Play (LfGP) (Ablett et al., 2021) uses scheduled auxiliary tasks to address lacking exploration in adversarial online IL. Paul et al. (2019) learn a generative model over subgoals from demonstrations and use it to augment the reward function for RL fine-tuning. However, these methods do not incorporate high-level planning mechanisms, which may make them unsuitable for solving complex reasoning problems.

Offline RL. In offline RL, the agent's objective is to learn an optimal policy without interacting with the environment. Instead, the agent has access to a dataset of transitions that have been collected by a behavior policy π_β that can be suboptimal. A significant benefit that offline RL has over imitation learning is the ability to extract strong policies even when the expert trajectories are suboptimal (Kumar et al., 2022). Conservative Q-learning (Kumar et al., 2020) is a model-free offline RL method that learns a lower bound on the policy value, which helps avoid overestimating state values. Decision Transformer (Chen et al., 2021) treats the offline reinforcement learning task as a sequence modeling problem, where the goal is to predict the action conditioned on a desired reward. We use offline RL methods as baselines in our experiments.

Hierarchical IL with Discrete Search. The closest works to ours in search and planning are kSubS (Czechowski et al.,

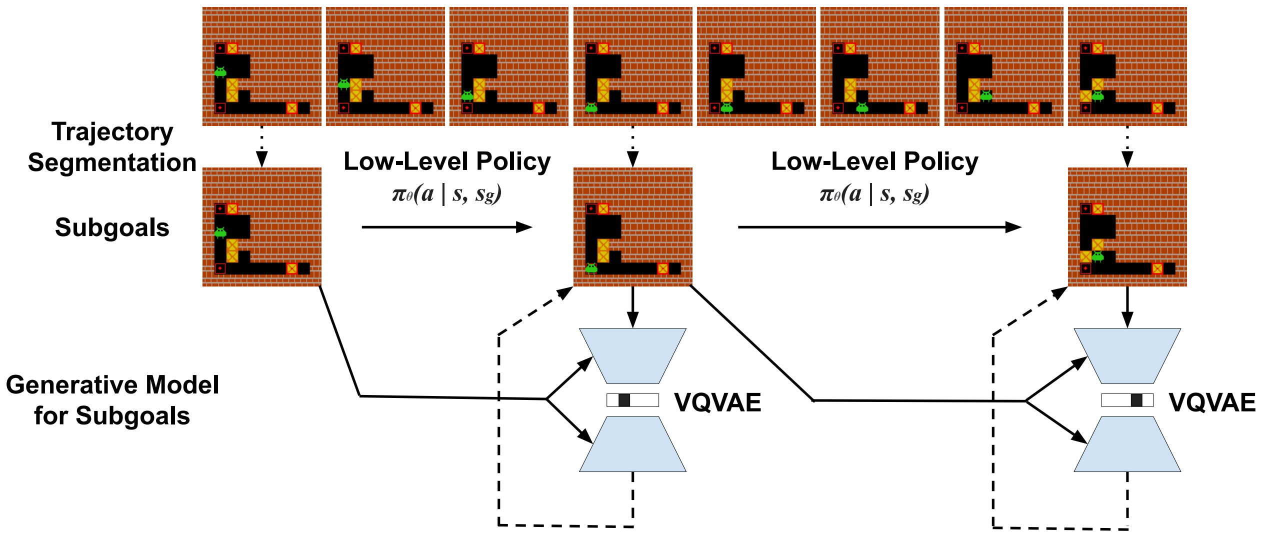

Trajectory Segmentation
Subgoals
Low-Level Policy
$\pi_\theta(a \mid s, s_g)$
Low-Level Policy
$\pi_\theta(a \mid s, s_g)$
Generative Model for Subgoals
VQVAE
VQVAE

Figure 1: A visualization of our *Hierarchical Imitation Planning with Search* (HIPS) when learning to solve Sokoban. The main components of our method are a detector $d_\xi(s_{g_k}|s_i)$ that segments the trajectory into subgoals, a subgoal-conditioned low-level policy $\pi_\theta(a_i|s_i, s_{g_k})$, and the VQVAE, a generative model over the subgoals. The low-level policy and VQVAE are used during evaluation for planning, whereas the detector is training-only.

2021) and AdaSubS (Zawalski et al., 2022). kSubS learns an autoregressive model for generating subgoals given a set of trajectories and plans with subgoal search. Unlike our method, kSubS relies on the true environment dynamics for low-level search to find fixed-length subtrajectories between subgoals in some environments when combined with an autoregressive CNN. AdaSubS replaces the low-level search of kSubS with a learned policy and supports subtrajectories of multiple hard-coded lengths by training an autoregressive generative model for each length. Our approach relies on a learned low-level policy and dynamics model and supports varying-length segments. We show that it is possible to train a single non-autoregressive adaptive generative model, which makes generating subgoals more efficient.

3. Method

We consider goal-oriented, complex reasoning problems, in which the agent's objective is to act in the environment to reach a terminal state. This corresponds to a Markov decision process with a reward of one upon solving the task and zero otherwise. We consider environments with fully Markovian, discrete-valued states with full observability. We work in the imitation learning (offline) setting: the agent needs to learn to solve tasks only from available demonstrations without the possibility to interact with the environment before evaluation. We assume that there is a dataset $\mathcal{D}$ of trajectories $\tau = \{s_0, a_0, s_1, \ldots, a_{T-1}, s_T\}$ collected by experts who know how to solve tasks, with only a reward of one at the terminal state s_T. The experts may not reach the terminal states in the fastest possible way. Our method should be able to combine parts of the dataset trajectories, that is, perform stitching (Singh et al., 2020), to discover efficient solutions.

We propose to solve the imitation learning task using an agent which has a hierarchical structure with two levels. Our agent learns a hierarchical representation of the available trajectories by identifying likely experts' subgoals in the existing trajectories in an unsupervised manner. A low-level policy for reaching these subgoals is learned simultaneously. The identified subgoals are then used to train a discrete-code generative model which can generate reasonable subgoals to perform subgoal-level planning with standard search algorithms such as PHS, MCTS, or A*. The low-level policy executes the plan generated by the planner by sequentially reaching the determined subgoals. Planning with search allows our method to improve on suboptimal demonstrations. A graphical representation of our method is shown in Fig. 1. We call our method *Hierarchical Imitation Planning with Search* (HIPS). Below we describe the main components of the approach: the subgoal detector $d_\xi(s_{g_{k+1}}|s_{g_k})$, the subgoal conditioned low-level policy $\pi_\theta(a_i|s_i, s_{g_k})$, and the subgoal generative model $p(s_g|s)$.

3.1. Subgoal Identification

The goal of the subgoal identification phase is to learn a high-level representation $\tau_* = (s_{g_1}, s_{g_2}, \ldots, s_{g_M})$ of each trajectory such that the trajectory is represented as a sequence of subgoals s_{g_k}. Each subgoal is a state from the trajectory, that is $\forall_k s_{g_k} \in \tau$, and in particular, $s_{g_M} = s_T$. This is a time series segmentation problem where the solution has three desirable qualities: 1) we want the identified subgoals s_{g_k} to be easy to reach by a trainable low-level policy $\pi_\theta(a_i|s_i, s_{g_k})$ which takes the subgoal s_{g_k} as input, 2) we want the subgoals to be easy to sample from a generative model, and 3) we want to segment the trajectories into as few subgoals as possible to make planning over them more efficient.

Successfully segmenting the trajectories into a variable number of variable-length segments turns out to be a highly nontrivial task, as most prior work has focused on fixed-length segments or a fixed number of segments. Our solution is to formulate this segmentation task as an RL problem in which we treat each trajectory from $\mathcal{D}$ as one episode for training, and try to maximize the subgoal-conditioned log-probability of the actions in the trajectory, $\sum_{i=0}^{T-1} \log \pi_\theta(a_i|s_i, s_{g_k})$, and minimize the number of segments.

In each episode, the segmentation agent starts at the first state s_0 of the trajectory and selects the next subgoal state s_{g_1} according to the probabilities produced by its policy $d_\xi(s_{g_1}|s_0)$, which we call *the detector*. We limit the detector to consider only the following H states as candidate subgoals, that is s_{g_1} is selected from $s_1, ..., s_H$. The agent samples the next subgoal according to the computed probabilities and gets the reward

$$R_1 = r_1 - \alpha, \tag{1}$$

where r_1 is the log-probability that a low-level policy $\pi_\theta(a_i|s_i, s_{g_1})$ selects the sequence of actions $a_0, ..., a_{g_1-1}$ in the first segment:

$$r_1 = \sum_{i=0}^{g_1-1} \log \pi_\theta(a_i|s_i, s_{g_1})$$

and α is a penalty to prevent segmentation into too many subtrajectories. After that, the segmentation agent changes its state to s_{g_1}, selects the next subgoal according to $d_\xi(s_{g_2}|s_{g_1})$ and gets reward R_2 computed using action log-probabilities from the second segment, similarly to (1). The episode continues like this until the end of the trajectory is reached.

The low-level policy $\pi_\theta(a_i|s_i, s_{g_k})$ is considered as part of the environment of the segmentation agent. It is updated during training using goal-conditioned behavioral cloning to minimize

$$\mathcal{L}_\theta = -\mathbb{E}_{\tau\sim\mathcal{D}} \sum_{k=1}^{M(\tau)} \sum_{i=g_{k-1}}^{-1+g_k} \log \pi_\theta(a_i|s_i, s_{g_k}), \tag{2}$$

with subgoals s_{g_k} produced by the segmentation agent. Note that the number of identified subgoals $M(\tau)$ may vary across trajectories.

Algorithm 1 Segmenting Trajectories for Hierarchical IL

Input: A dataset of trajectories $\mathcal{D}$, untrained low-level policy network π_θ, detector d_ξ
Parameters: The parameters of the low-level policy network, θ, detector network, ξ
Output: Trained low-level policy π, dataset $\mathcal{D}'$ of subgoal pairs $\{(s_{g_{k+1}}, s_{g_k})\}$

```
1: while $\pi_\theta$, $d_\xi$ not converged do
2:     Sample a trajectory $\tau$.
3:     Segment $\tau$ with $d_\xi$
4:     Predict low-level actions with $\pi_\theta$ conditioned on produced subgoals
5:     Compute returns (Equation 1), update $\xi$ with REINFORCE
6:     Compute the losses for $\pi_\theta$ (Equation 2), update $\theta$
7: end while
8: Create a dataset $\mathcal{D}'$ of subgoal pairs $\{(s_{g_{k+1}}, s_{g_k})\}$ by sampling trajectories $\tau$ from $\mathcal{D}$ and segmenting them with $d_\xi$.
9: return $\pi_\theta, \mathcal{D}'$
```

Thus, the segmentation agent is trained by giving it a higher reward when selected subgoals lead to more accurate action predictions by the low-level policy. The low-level policy is trained concurrently with the subgoals (high-level commands) produced by the segmentation agent as input. We train the segmentation agent using the policy gradient algorithm REINFORCE with a learnable value function as baseline (Williams, 1992).

We train the low-level policy and detector simultaneously, and the associated non-stationarity might cause issues. In practice, we observe that $\pi_\theta(a_i|s_i, s_{g_k})$ converges rapidly during the simultaneous training and its effect on the non-stationarity is limited. Note that using a trainable detector $d_\xi(s_{g_{k+1}}|s_{g_k})$ may encourage subgoals that are easy to recognize among the states in the training trajectories. If that is true, we hypothesize that such subgoals may be easy to produce by a learned generative model. Our approach for segmenting trajectories is summarized in Algorithm 1.

3.2. Generative Model for Subgoals

To plan in terms of the high-level subgoals, the agent needs the ability to generate reasonable subgoals s_g for each environment state s. We do this by learning a generative model $p(s_g|s)$ over the subgoals using the ones identified in the trajectory segmentation step as training data. We implement the model as a VQVAE with discrete latent codes, which is inspired by the vector quantized models proposed by Ozair et al. (2021). The VQVAE encoder takes a pair of states (s_g, s) as input and outputs a continuous latent code z_e. Then, the code is quantized by finding the nearest code e_k

from the codebook such that $k = \arg\min_m \|z_e - e_m\|_2$. The decoder uses the code e_k to reconstruct the subgoal state $\hat{s}_g = g_\psi(e_k, s)$. The loss minimized during training is

$$\mathcal{L} = \mathcal{L}_{\text{rec}}(\hat{s}_g, s_g) + \|[z_e] - e_k\|_2^2 + \beta\|z_e - [e_k]\|_2^2 , \quad (3)$$

where $\mathcal{L}_{\text{rec}}$ is the reconstruction loss and $[\cdot]$ denotes the stop gradient operation.

We train the VQVAE in two stages. In the pre-training stage, we use random pairs of states $(s_j, s_i), i < j \leq i + H$ from the trajectories as inputs (s_g, s). We skip the discretization layer and only use the reconstruction loss $\mathcal{L}_{\text{rec}}(\hat{s}_g, s_g)$ for training the encoder and the decoder. After the pre-training has converged, the complete VQVAE is trained by using pairs of consecutive subgoals $(s_{g_{k+1}}, s_{g_k})$ as input. We initialize the codebook by running KMeans++ clustering (Arthur & Vassilvitskii, 2007) on the first batches of encoder outputs and using the cluster centers as the initial codes. This training strategy was inspired by the strategy proposed by Łańcucki et al. (2020). Finally, we learn a subgoal-conditioned prior $p(e_{k+1}|s_{g_k})$ over the latent codes.

Once the model has been trained, one can generate subgoals conditioned on the current state s by sampling a code e from the learned codebook and running it through the decoder $g_\psi(e, s)$. Note that the number of possible codes e is finite, which means that the number of generated subgoals s_g is finite as well. Note also that distinct codes e may result in the same generated subgoal s_g, which is a desired behavior when the size of the codebook is larger than the number of reasonable subgoals for the considered state. The pseudocode for our VQVAE training is given in Algorithm 2 in the Appendix I.

3.3. High-Level Planning with Search

We perform planning in the subgoal space, as it can be more efficient and suitable for long-horizon planning than planning in the state space (Nair & Finn, 2020). We demonstrate that our method is compatible with many different search algorithms. Each search node represents a subgoal. When a search node is expanded, possible next subgoals (child nodes) are generated with the VQVAE. Given a codebook of size K, there are K possible child nodes for each subgoal. This can be interpreted as the latent codes encoding the different branches of the search tree rather than the actual subgoals, which helps us significantly constrain the codebook sizes. To limit the size of the search tree, we remove duplicates and unreachable subgoals. The reachability of a proposed subgoal s_g from state s_i is evaluated by using the low-level policy $\pi_\theta(a_i|s_i, s_g)$ trained in the segmentation phase. We run the policy iteratively from state s_i and simulate state transitions by using the true dynamics or a learned model $f_{\text{dyn}}(s_{i+1}|a_i, s_i)$. If the subgoal s_g is reached within a specific number of steps, the subgoal is considered reachable, and a search node is created. In this work, we work with discrete states and require an exact match between the reached state and the subgoal. Using a suitable threshold to evaluate the match is an alternative in the continuous setting. We also learn a value function $V(s)$ that predicts the number of low-level steps necessary to reach the goal (terminal state) and use it as a heuristic in planning.

The search methods we use are Greedy Best-First Search (GBFS), Policy-Guided Heuristic Search (PHS*, Orseau & Lelis, 2021), A* (Hart et al., 1968) and Monte-Carlo Tree Search (MCTS, Coulom, 2006; Kocsis & Szepesvári, 2006). PHS* is dependent on a good policy, but when the dataset contains suboptimal trajectories, learning a good VQVAE prior to act as the policy might be impossible. Then, policy-independent algorithms like A* or GBFS can be superior. PHS* is also aimed at minimizing the search loss, not finding a particularly high-quality solution. When that is important, A* or MCTS can be superior to PHS*.

4. Experiments

In our experimental phase, we evaluate our method on complex, sparse reward problems that require reasoning. We compare our method to existing search, hierarchical imitation learning, and offline RL methods. We also analyze whether our RL-based approach for identifying subgoals is superior to subgoals sampled at fixed intervals.

4.1. Environments

We evaluate our method in four environments that are all complex reasoning domains (see Fig. 2). The first environment is Sokoban, which is a PSPACE-complete puzzle where the agent must push boxes onto goal locations (Culberson, 1997). The moves are irreversible and one wrong push can make the puzzle unsolvable. We use a 10×10 problem size with four boxes, the default configuration in the earlier literature (Orseau & Lelis, 2021; Guez et al., 2019; Racanière et al., 2017). We use a one-hot encoded tensor with shape $10 \times 10 \times 4$ as the observation space (Orseau & Lelis, 2021).

The second environment is the sliding tile puzzle (STP) which is a classic benchmark in the search literature (Korf, 1985). We use a puzzle size of 5×5, and the objective is to sort the number tiles in a specific descending order.

The third environment is Box-World (BW), where the agent must collect colored keys and open color-matching locks to recover more keys until it finally reaches a goal target (Zambaldi et al., 2018). Keys can only be used once. If the agent uses its key to open the wrong box, the game will become unsolvable. Hence, careful planning and reasoning about entities and their relations are required.

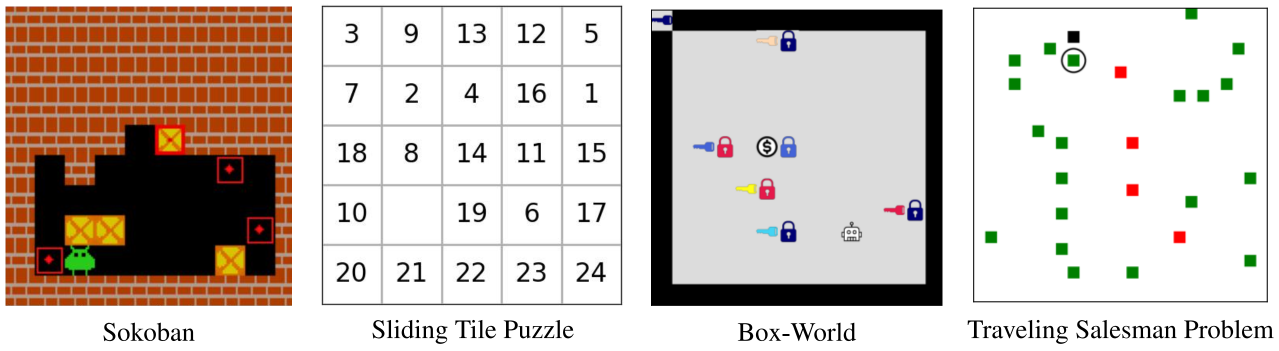


Figure 2: The environments we consider in the experiments. Sokoban: the task is to push yellow boxes onto the target locations (marked with red squares). Sliding Tile Puzzle (STP): The task is to order the tiles from 1 to 24 by moving them. Box-World: The agent must collect colored keys and open color-matching locks to recover more keys until it finally reaches a goal target (marked with the $ sign). Traveling salesman problem (TSP): The agent (marked with the circle) has to visit all cities (marked with squares) before returning to the start (marked with the black square). Visited cities are marked with green squares and unvisited ones with red squares.

The fourth environment is a grid-based Traveling Salesman Problem (TSP). The agent moves through the 2D grid to visit all the cities before returning to the starting point (see Fig. 2). The TSP is an NP-hard combinatorial optimization problem. However, finding any solution to TSP is relatively easy. Hence, grid-based TSP serves as an environment for evaluating the solution quality.

In Sokoban, we collect a training set of 10340 trajectories using gym-sokoban (Schrader, 2018). 10240 of the problem instances have been solved using Curry (Shoham, 2021) and 100 trajectories were collected by performing random actions. The training set consists of 5100 trajectories in STP and 22100 in Box-World, of which 5000 and 22000 were collected by solving the problem instances with a subgoal-based A* algorithm (Hart et al., 1968) and 100 by performing random actions. In subgoal-based A*, we generated subgoals progressively closer to the terminal state procedurally and executed A* to reach these subgoals, as solving complete problem instances with A* would have been computationally very expensive due to the complexity of the environments. In TSP, we do not limit the number of demonstrations available to the agent but the demonstration trajectories are highly suboptimal and generated by running an agent that visits 25 cities in a 25×25 grid in random order.

4.2. Agents

Our HIPS agent consists of the following neural networks: the detector $d_\xi(s_{g_{k+1}}|s_{g_k})$, the low-level policy $\pi_\theta(a_i|s_i, s_{g_k})$, the VQVAE encoder $f_\phi(z_{e_{k+1}}|s_{g_{k+1}}, s_{g_k})$, the VQVAE decoder $g_\psi(s_{g_{k+1}}|e_{k+1}, s_{g_k})$, the VQVAE prior $p(e_{k+1}|s_{g_k})$, the low-level dynamics model $f_{\text{dyn}}(s_{i+1}|a_i, s_i)$, and the distance function $V(s_i)$. The encoder f_ϕ and detector d_ξ are not used during evaluation. All networks are ResNet-based CNNs (He et al., 2016). The decoder and low-level dynamics CNNs also contain FiLM layers (Perez et al., 2018). We only use the 100 random trajectories for training the low-level dynamics model. In Box-World, the neural networks additionally use Deep Recurrent Convolutions (Guez et al., 2019). All neural networks are implemented with PyTorch (Paszke et al., 2019) and trained with an Adam optimizer (Kingma & Ba, 2015). We evaluate Sokoban and Box-World with PHS* as the high-level search algorithm. Because of the suboptimality of the TSP trajectories, learning a good VQVAE prior is difficult, and we use A* in TSP. We also observed that GBFS is superior to PHS* with very small search budgets in STP (see Appendix G).

We compare our agents to two main classes of baselines. The first class of baselines is strong IL and offline RL algo-

Table 1: The overall success rates (%) of different algorithms. The algorithms in the bottom part have access to the true environment dynamics, and those in the upper part do not.

Method	Sokoban	STP	BW	TSP
HIPS (ours)	97.5	**94.7**	**55.7**	**100.0**
HIPS-k (ours)	**99.0**	**94.7**	20.1	**100.0**
BC	18.7	82.5	41.1	28.8
CQL	3.3	11.7	6.0	33.6
DT	36.7	0.0	10.0	0.0
RIS	0.0	0.0	7.3	0.0
Option-GAIL	0.3	0.0	0.0	0.0
IQ-Learn	0.0	0.0	0.0	0.0
HIPS-env (ours)	98.1	94.6	99.6	100.0
AdaSubS	91.4	0.0		22.4
kSubS	90.5	93.3		87.9

Table 2: The success rates (%, higher is better) of different search algorithms after performing N node expansions. In Sokoban and Box-World, HIPS was evaluated with PHS*, with GBFS in STP, and with A* in TSP.

	Sokoban			**Sliding Tile Puzzle**			**Box-World**			**Travelling Salesman**		
N	100	500	1000	100	500	1000	100	500	1000	100	500	1000
HIPS (ours)	**88.5**	94.7	95.9	89.8	91.8	92.2	55.7	55.7	55.7	**100.0**	**100.0**	**100.0**
HIPS-k (ours)	77.7	90.9	94.3	68.0	79.1	84.4	20.1	20.1	20.1	77.3	**100.0**	**100.0**
HIPS-env (ours)	**88.5**	**94.9**	**96.4**	**90.2**	**92.1**	92.6	**99.6**	**99.6**	**99.6**	**100.0**	**100.0**	**100.0**
HIPS-env-k (ours)	76.9	91.3	94.0	69.7	80.9	87.2	99.1	99.1	99.1	77.2	**100.0**	**100.0**
AdaSubS	82.2	88.8	90.2	0.0	0.0	0.0				1.2	12.2	20.8
kSubS	73.1	79.9	82.8	79.9	91.7	**92.7**				40.4	80.9	85.3
PHS*	1.8	76.1	91.1	0.0	0.0	0.0	25.8	93.8	94.0	0.0	0.0	0.0

rithms. We compare our agent to standard flat behavioral cloning (BC), a powerful offline RL algorithm, Conservative Q-Learning (CQL, Kumar et al., 2020), and a strong IL algorithm, Inverse Soft-Q Learning (IQ-Learn, Garg et al., 2021). We ran IQ-Learn in the online mode, where it could collect additional data during training. Furthermore, we include the Decision Transformer (DT, Chen et al., 2021), the hierarchical IL algorithm Option-GAIL (Jing et al., 2021), and the online goal-conditioned RL algorithm RIS (Chane-Sane et al., 2021) in our experiments. For CQL, we use the implementation of d3rlpy (Seno & Imai, 2021), and for other algorithms, we use the open-sourced implementations of the authors. For all methods that need rewards, we give the agent a reward of one upon completing the task and 0 otherwise. We do not include the 100 random trajectories in the dataset when evaluating imitation learning algorithms.

The second class of baselines is search methods that use the true environment dynamics, a state-of-the-art low-level search, Policy Guided Heuristic Search (PHS*, Orseau & Lelis, 2021), and two subgoal-level search methods: kSubS (Czechowski et al., 2021) and AdaSubS (Zawalski et al., 2022). In PHS*, we observed that using a policy trained with behavioral cloning works better than a policy trained using the loss function proposed by Orseau & Lelis (2021) and therefore, we use the BC policy in the experiments. We evaluate kSubS and AdaSubS with the autoregressive CNNs on all environments except Box-World because it would have required significant changes to the existing implementation. Our method, HIPS, relies on a learned dynamics model instead of the true dynamics. Hence, it solves a more complex problem. We also train a more comparable variant of our method, HIPS-env, that uses an environment simulator for planning.

4.3. Results

We use the overall success rates reported in Table 1 as the main evaluation metric in all environments. We evaluate the performance of each seed and take the mean over the random seeds. We use 10 random seeds to evaluate our method, HIPS, at least five seeds per ablation, and at least three seeds per baseline method. When evaluating the overall success rate, the search algorithms may perform as many expansions as necessary to find a solution to the problems. The critical success factor for our model is the capacity of the generative model to cover the complete search space with the proposed subgoals.

Our method, HIPS, outperforms the baseline methods in all four environments (see Table 1). The table also contains an ablation of our method, HIPS-k, where we train the VQ-VAE with subgoals sampled at fixed intervals as done in AdaSubS and kSubS instead of using the detector network (see Appendices G and J for more details). Eliminating the detector leads to a clear drop in performance in one of the four environments. HIPS-k is superior to kSubS in all environments despite solving a more difficult problem than kSubS. HIPS-k must learn the environment dynamics and a low-level policy, whereas kSubS uses a low-level search with the environment dynamics. HIPS-k also is superior to AdaSubS which also uses a low-level policy instead of search. The sparse reward structure and the required reasoning capabilities prove to be very difficult for the model-free baseline IL and offline RL methods that do not rely on planning, which is why they struggle with all four tasks.

HIPS-env performs equally to HIPS, except in Box-World, where the search exploits the inaccuracies of the dynamics model. However, HIPS was evaluated using open-loop planning, where one plan was generated, executed, and evaluated. If the agent is allowed to re-plan when the dynamics model deviates from the environment and fine-tune the model with the new transition, the performance would most likely increase. The issues with the dynamics model do not prevent HIPS from outperforming the baselines.

Letting a search algorithm perform unlimited expansions is unrealistic in most real-world applications. Following Czechowski et al. (2021), we evaluate the percentage of test problems solved after N node expansions. The results are

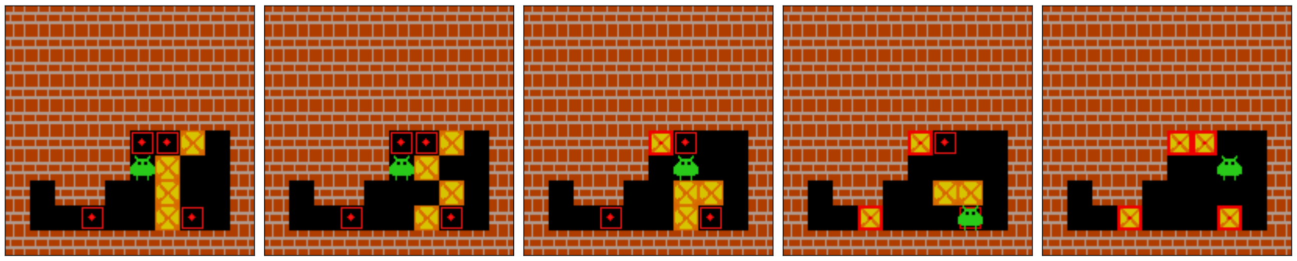


Figure 3: An example of a high-level plan (a sequence of subgoals) found by HIPS in Sokoban.

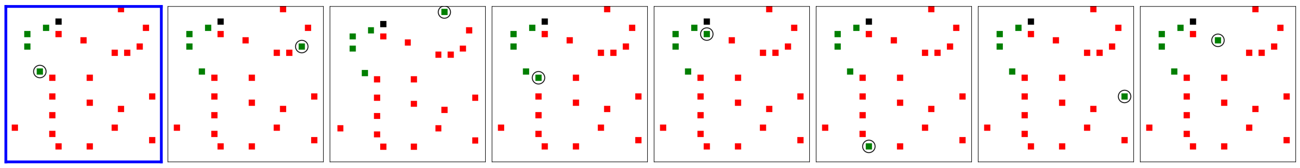


Figure 4: Examples of subgoals proposed for the current state (marked with blue boundaries) in TSP.

Table 3: The success rates of different algorithms (higher is better) and the average number of steps (lower is better) needed to solve TSP.

Method	Success rate (%)	Avg. steps
HIPS-PHS* (ours)	**100.0**	305.9
HIPS-MCTS (ours)	**100.0**	213.0
HIPS-A* (ours)	**100.0**	**168.2**
kSubS	87.9	268.2
CQL	33.6	336.9
BC	28.8	339.3
AdaSubS	22.4	338.9
Teacher	100.0	336.5
Oracle MCTS	100.0	199.9
Christofides	100.0	139.0

given in Table 2. The benefits of using RL to detect subframes become clear, as HIPS outperforms the fixed-length ablation HIPS-k in Sokoban, Sliding Tile Puzzle (STP), and Travelling Salesman Problem (TSP). HIPS-env outperforms the baseline methods in Sokoban and TSP and is superior to kSubS on STP when the search budget is small. PHS* can solve STP and TSP, but cannot make enough progress in 1000 node expansions. AdaSubS cannot solve STP because it struggles to reliably reach the generated subgoals with its low-level policy. Therefore, kSubS outperforms AdaSubS.

TSP is a problem where generating a successful trajectory is easy, but finding an efficient solution is much more difficult. Hence, we evaluate the solution lengths of the methods. We also compare the search algorithms PHS*, MCTS, and A* when used with HIPS. Given the amount of data, we perform VQVAE pre-training using pairs of subgoals instead of random pairs. In addition to the baselines capable of solving TSP, the performance of our method is compared to a known approximation algorithm Christofides (Christofides, 1976), to the training dataset (Teacher), and to an Oracle variant of subgoal-level MCTS where we replace the VQVAE generator with procedurally generated subgoals, where the agent is visiting one of the remaining unvisited cities.

The solution lengths found in TSP are reported in Table 3. HIPS finds better solutions than the baselines in TSP. HIPS with PHS* can only slightly improve the training data, whereas HIPS-MCTS with subgoal-level rollouts can find significantly better solutions than the ones in the training dataset. It is inferior to the approximation algorithm, but the gap to the Oracle MCTS is small (around 6 %), which shows that the subgoals generated by the VQVAE are competitive with the procedurally generated subgoals. Finally, HIPS-A* is the best-performing agent, and the gap to Christofides is around 20 %. Note that our learned heuristic is non-admissible, and we trade off optimality for speed. kSubS can also improve on the training dataset, but it is uncompetitive against HIPS-MCTS and HIPS-A*. CQL, BC, and AdaSubS can make some progress on the task and solve some instances, but they cannot improve the solution lengths. A problem of model-free baseline methods is the inability to commit to going to a specific city, which highlights the benefits of goal-conditioned learning. Complete results including the standard errors can be found in Appendix F.

4.4. Visualizations

We visualize the subgoals and plans generated by our agent to gain further understanding into the agent. An example of a high-level plan found by HIPS for Sokoban is visualized in Fig. 3. Fig. 4 illustrates the subgoals proposed by the

HIPS generative model for an intermediate state in TSP. The model suggests visiting one of the unvisited cities as the next subgoal, which is a very reasonable planning strategy. More visualizations of the high-level trajectories discovered by our agent and the subgoals proposed by the generative model can be found in Appendix J.

5. Conclusion

We present a novel method for hierarchical IL that can address difficult reasoning problems that require long-term decision-making. Our approach relies on identifying subgoals from trajectories and generating new subgoals for search-based planning on new problem instances. Our method outperforms powerful search, IL, and offline RL baselines on the benchmark tasks. Our experiments demonstrate that our VQVAE is a suitable generative model for subgoal-level search and using a detector to discover subgoals has benefits over subtrajectories of fixed length.

We see many promising directions for addressing the limitations of our method and improving it. Quantifying the model uncertainty could help prevent the search from exploiting the learned models. Combining discrete high-level planning with continuous low-level execution could make it possible to solve real-world tasks with robotics. Learning more abstract goals not formulated in the observation space to improve the efficiency of high-level planning and allowing the agent to ignore task-irrelevant sensory inputs to handle real-world vision tasks are also left for future work. Combining low-level and high-level searches would improve the solution rate. Applying the learned high-level models in an RL setting to improve exploration or in a curriculum learning to solve progressively harder problem instances are also promising directions for future research.

Acknowledgements

We acknowledge the computational resources provided by the Aalto Science-IT project and CSC, Finnish IT Center for Science. The work was funded by Academy of Finland within the Flagship Programme, Finnish Center for Artificial Intelligence (FCAI). J. Pajarinen was partly supported by Academy of Finland (345521).

References

Abbeel, P. and Ng, A. Y. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the Twenty-First International Conference on Machine Learning*, pp. 1, 2004.

Ablett, T., Chan, B., and Kelly, J. Learning from guided play: A scheduled hierarchical approach for improving exploration in adversarial imitation learning. In *Proceedings of the Neural Information Processing Systems Deep Reinforcement Learning Workshop*, 2021.

Argenson, A. and Dulac-Arnold, G. Model-based offline planning. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*, 2021.

Arthur, D. and Vassilvitskii, S. K-Means++: The advantages of careful seeding. In *ACM-SIAM Symposium on Discrete Algorithms*, 2007.

Bacon, P.-L., Harb, J., and Precup, D. The option-critic architecture. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2016.

Bagaria, A. and Konidaris, G. Option discovery using deep skill chaining. In *International Conference on Learning Representations*, 2020.

Chane-Sane, E., Schmid, C., and Laptev, I. Goal-conditioned reinforcement learning with imagined subgoals. In *International Conference on Machine Learning*, pp. 1430–1440. PMLR, 2021.

Chen, D. and Krähenbühl, P. Learning from all vehicles. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 17222–17231, 2022.

Chen, L., Lu, K., Rajeswaran, A., Lee, K., Grover, A., Laskin, M., Abbeel, P., Srinivas, A., and Mordatch, I. Decision transformer: Reinforcement learning via sequence modeling. *Advances in Neural Information Processing Systems*, 34:15084–15097, 2021.

Christofides, N. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical report, Carnegie-Mellon Univ Pittsburgh Pa Management Sciences Research Group, 1976.

Coulom, R. Efficient selectivity and backup operators in Monte-Carlo tree search. In *International Conference on Computers and Games*, pp. 72–83. Springer, 2006.

Culberson, J. Sokoban is PSPACE-complete. *IEICE Technical Report*, 1997.

Czechowski, K., Odrzygóźdź, T., Zbysiński, M., Zawalski, M., Olejnik, K., Wu, Y., Kuciński, Ł., and Miłoś, P. Subgoal search for complex reasoning tasks. *Advances in Neural Information Processing Systems*, 34:624–638, 2021.

Daniel, C., Van Hoof, H., Peters, J., and Neumann, G. Probabilistic inference for determining options in reinforcement learning. *Machine Learning*, 104(2):337–357, 2016.

Dayan, P. and Hinton, G. E. Feudal reinforcement learning. *Advances in Neural Information Processing Systems*, 5, 1992.

Ebert, F., Finn, C., Dasari, S., Xie, A., Lee, A., and Levine, S. Visual foresight: Model-based deep reinforcement learning for vision-based robotic control. *arXiv preprint arXiv:1812.00568*, 2018.

Fox, R., Berenstein, R., Stoica, I., and Goldberg, K. Multi-task hierarchical imitation learning for home automation. In *2019 IEEE 15th International Conference on Automation Science and Engineering (CASE)*, pp. 1–8. IEEE, 2019.

Garg, D., Chakraborty, S., Cundy, C., Song, J., and Ermon, S. IQ-learn: Inverse soft-Q learning for imitation. *Advances in Neural Information Processing Systems*, 34: 4028–4039, 2021.

Guez, A., Mirza, M., Gregor, K., Kabra, R., Racanière, S., Weber, T., Raposo, D., Santoro, A., Orseau, L., Eccles, T., et al. An investigation of model-free planning. In *International Conference on Machine Learning*, pp. 2464–2473. PMLR, 2019.

Hafner, D., Lee, K.-H., Fischer, I., and Abbeel, P. Deep hierarchical planning from pixels. *arXiv preprint arXiv:2206.04114*, 2022.

Hart, P. E., Nilsson, N. J., and Raphael, B. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.

Henderson, P., Chang, W.-D., Bacon, P.-L., Meger, D., Pineau, J., and Precup, D. Optiongan: Learning joint reward-policy options using generative adversarial inverse reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Jayaraman, D., Ebert, F., Efros, A. A., and Levine, S. Time-agnostic prediction: Predicting predictable video frames. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019.

Jing, M., Huang, W., Sun, F., Ma, X., Kong, T., Gan, C., and Li, L. Adversarial option-aware hierarchical imitation learning. In *International Conference on Machine Learning*, pp. 5097–5106. PMLR, 2021.

Jurgenson, T., Avner, O., Groshev, E., and Tamar, A. Sub-goal trees a framework for goal-based reinforcement learning. In *International Conference on Machine Learning*, pp. 5020–5030. PMLR, 2020.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In Bengio, Y. and LeCun, Y. (eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

Kipf, T., Li, Y., Dai, H., Zambaldi, V., Sanchez-Gonzalez, A., Grefenstette, E., Kohli, P., and Battaglia, P. Compile: Compositional imitation learning and execution. In *International Conference on Machine Learning*, pp. 3418–3428. PMLR, 2019.

Kober, J. and Peters, J. Policy search for motor primitives in robotics. *Advances in Neural Information Processing Systems*, 21, 2008.

Kocsis, L. and Szepesvári, C. Bandit based Monte-Carlo planning. In *European Conference on Machine Learning*, pp. 282–293. Springer, 2006.

Korf, R. E. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence*, 27(1):97–109, 1985.

Kumar, A., Zhou, A., Tucker, G., and Levine, S. Conservative Q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 33: 1179–1191, 2020.

Kumar, A., Hong, J., Singh, A., and Levine, S. When should we prefer offline reinforcement learning over behavioral cloning? *arXiv preprint arXiv:2204.05618*, 2022.

Łańcucki, A., Chorowski, J., Sanchez, G., Marxer, R., Chen, N., Dolfing, H. J., Khurana, S., Alumäe, T., and Laurent, A. Robust training of vector quantized bottleneck models. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–7. IEEE, 2020.

Le, H., Jiang, N., Agarwal, A., Dudik, M., Yue, Y., and Daumé III, H. Hierarchical imitation and reinforcement learning. In *International Conference on Machine Learning*, pp. 2917–2926. PMLR, 2018.

Li, J., Tang, C., Tomizuka, M., and Zhan, W. Hierarchical planning through goal-conditioned offline reinforcement learning. *IEEE Robotics and Automation Letters*, 2022.

Mayer, H., Gomez, F., Wierstra, D., Nagy, I., Knoll, A., and Schmidhuber, J. A system for robotic heart surgery that learns to tie knots using recurrent neural networks. *Advanced Robotics*, 22(13-14):1521–1537, 2008.

Nachum, O., Tang, H., Lu, X., Gu, S., Lee, H., and Levine, S. Why does hierarchy (sometimes) work so well in reinforcement learning? *arXiv preprint arXiv:1909.10618*, 2019.

Nair, S. and Finn, C. Hierarchical foresight: Self-supervised learning of long-horizon tasks via visual subgoal generation. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, 2020.

Niepert, M., Minervini, P., and Franceschi, L. Implicit MLE: Backpropagating through discrete exponential family distributions. *Advances in Neural Information Processing Systems*, 34:14567–14579, 2021.

Orseau, L. and Lelis, L. H. Policy-guided heuristic search with guarantees. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 12382–12390, 2021.

Orseau, L., Lelis, L., Lattimore, T., and Weber, T. Single-agent policy tree search with guarantees. *Advances in Neural Information Processing Systems*, 31, 2018.

Osa, T., Pajarinen, J., Neumann, G., Bagnell, J. A., Abbeel, P., and Peters, J. An algorithmic perspective on imitation learning. *Foundations and Trends® in Robotics*, 7(1-2): 1–179, 2018.

Ozair, S., Li, Y., Razavi, A., Antonoglou, I., Van Den Oord, A., and Vinyals, O. Vector quantized models for planning. In *International Conference on Machine Learning*, pp. 8302–8313. PMLR, 2021.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. PyTorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, 32, 2019.

Paul, S., Vanbaar, J., and Roy-Chowdhury, A. Learning from trajectories via subgoal discovery. *Advances in Neural Information Processing Systems*, 32, 2019.

Perez, E., Strub, F., De Vries, H., Dumoulin, V., and Courville, A. Film: Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Pertsch, K., Rybkin, O., Ebert, F., Zhou, S., Jayaraman, D., Finn, C., and Levine, S. Long-horizon visual planning with goal-conditioned hierarchical predictors. *Advances in Neural Information Processing Systems*, 33:17321–17333, 2020a.

Pertsch, K., Rybkin, O., Yang, J., Zhou, S., Derpanis, K., Lim, J., Daniilidis, K., and Jaegle, A. Keyin: Keyframing for visual planning. *Conference on Learning for Dynamics and Control*, 2020b.

Racanière, S., Weber, T., Reichert, D., Buesing, L., Guez, A., Jimenez Rezende, D., Puigdomènech Badia, A., Vinyals, O., Heess, N., Li, Y., et al. Imagination-augmented agents for deep reinforcement learning. *Advances in Neural Information Processing Systems*, 30, 2017.

Sammut, C., Hurst, S., Kedzier, D., and Michie, D. Learning to fly. In *Machine Learning Proceedings 1992*, pp. 385–393. Elsevier, 1992.

Schrader, M.-P. B. gym-sokoban. https://github.com/mpSchrader/gym-sokoban, 2018.

Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839): 604–609, 2020.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Seno, T. and Imai, M. d3rlpy: An offline deep reinforcement learning library. *arXiv preprint arXiv:2111.03788*, 2021.

Sharma, A., Gu, S. S., Levine, S., Kumar, V., and Hausman, K. Dynamics-aware unsupervised skill discovery. In *International Conference on Learning Representations*, 2020.

Sharma, M., Sharma, A., Rhinehart, N., and Kitani, K. M. Directed-info GAIL: Learning hierarchical policies from unsegmented demonstrations using directed information. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019a.

Sharma, P., Pathak, D., and Gupta, A. Third-person visual imitation learning via decoupled hierarchical controller. *Advances in Neural Information Processing Systems*, 32, 2019b.

Shoham, Y. Curry. https://festival-solver.site/curry/, 2021.

Silver, D. and Ciosek, K. Compositional planning using optimal option models. In *International Conference on Machine Learning*, 2012.

Singh, A., Yu, A., Yang, J., Zhang, J., Kumar, A., and Levine, S. COG: Connecting new skills to past experience with offline reinforcement learning. *arXiv preprint arXiv:2010.14500*, 2020.

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

Van Den Oord, A., Vinyals, O., and Kavukcuoglu, K. Neural discrete representation learning. *Advances in Neural Information Processing Systems*, 30, 2017.

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., et al. Alphastar: Mastering the real-time strategy game Starcraft II. *DeepMind Blog*, 24, 2019.

Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256, 1992.

Ye, W., Liu, S., Kurutach, T., Abbeel, P., and Gao, Y. Mastering atari games with limited data. *Advances in Neural Information Processing Systems*, 34:25476–25488, 2021.

Zambaldi, V., Raposo, D., Santoro, A., Bapst, V., Li, Y., Babuschkin, I., Tuyls, K., Reichert, D., Lillicrap, T., Lockhart, E., et al. Relational deep reinforcement learning. *arXiv preprint arXiv:1806.01830*, 2018.

Zawalski, M., Tyrolski, M., Czechowski, K., Stachura, D., Piekos, P., Odrzygóźdź, T., Wu, Y., Kuciński, Ł., and Miłoś, P. Fast and precise: Adjusting planning horizon with adaptive subgoal search. *arXiv preprint arXiv:2206.00702*, 2022.

Zhang, D., Li, Q., Zheng, Y., Wei, L., Zhang, D., and Zhang, Z. Explainable hierarchical imitation learning for robotic drink pouring. *IEEE Transactions on Automation Science and Engineering*, 2021.

Zhang, Z. and Paschalidis, I. Provable hierarchical imitation learning via EM. In *International Conference on Artificial Intelligence and Statistics*, pp. 883–891. PMLR, 2021.

A. Ethical Considerations and Societal Impact

We do not see any immediate negative societal impacts associated with our work. We do not train our models with any sensitive or private data, and our model is not directly applicable to, for instance, real-world decision-making concerning humans. However, we cannot exclude the method being applied to something harmful that is difficult to foresee, for instance, military purposes.

B. Infrastructure

We trained our models on an HPC cluster using one NVIDIA GPU and multiple CPU workers per run. Most runs were performed on V100 GPUs with 32 GB of GDDR SDRAM. For some of the runs, the GPU was an A100, a K100, or a P80. We used 6 CPU workers per GPU with 10 GB of RAM per worker and each worker running on one core. By reducing the number of workers, it is possible to train and evaluate the agent on a workstation with Intel i7-8086K, 16 GB of RAM, and an NVIDIA GeForce GTX 1080 Ti GPU with 10 GB of video memory.

C. Tried Ideas

During the development of the algorithm, we evaluated the different components, and we found that many of the elements in the presented solution are extremely important for achieving good performance. We tried replacing the detector with a straight-through estimator but found it very difficult to train, and it did not achieve good results. We also tried Implicit Maximum Likelihood Estimation (I-MLE, Niepert et al., 2021) for trajectory segmentation, but it failed to learn meaningful representations. We found that training the detector with PPO (Schulman et al., 2017) instead of REINFORCE did not have a significant effect on the performance in our initial tests, so we chose REINFORCE for the conceptual simplicity. Having fixed-length subgoals performed inferior to the detector, as the ablation (see Appendix G) study shows, and training a non-goal conditioned low-level policy did not work with our algorithm either. We also tried replacing the VQVAE with an autoregressive model, but it was not computationally feasible. A GAN-based generative model was difficult to train and could not easily generate subgoals that were dissimilar enough.

We attempted not evaluating reachability during planning. However, even though the generative model has been successfully trained, the fact that we use the same number of codes regardless of the state causes the generator to create some unreachable subgoals in states where the number of realistic subgoals is smaller than the number of latent codes or some latent code corresponds to, for instance, movement in an impossible direction. Given the limited number of trajectories, we do not have enough data to train the VQVAE prior to accurately reflect this. Hence, reachability must also be addressed during planning. We tried training a reachability network to achieve this, but the performance was not good enough in practice to rely solely on it. However, using a verifier to support the reachability evaluation as in (Zawalski et al., 2022) could improve the performance of our method.

D. Impact of Value Function and VQVAE Prior

We ran an ablation study in Sokoban to analyze the value function and the VQVAE prior. When the algorithm uses both the value function and the prior, the natural choice for the search algorithm is PHS*. When there is no value function, PHS* cannot be used, and the most similar suitable alternative to PHS* is LevinTS (Orseau et al., 2018). With a value function but without a prior, neither LevinTS nor PHS* cannot be used, so we chose GBFS as a representative search algorithm. The results for the number of puzzles solved after N expansions are plotted in Table 4. The results of the ablation show that HIPS-PHS* is superior to HIPS-GBFS, indicating that learning a VQVAE prior is beneficial for solving Sokoban. The comparison between HIPS-PHS* and HIPS-LevinTS shows that value functions are important for quickly solving the puzzles, as the inferior performance after 100 node expansions shows, but the impact at 500 expansions and after that is smaller. Overall having both a value function and the VQVAE prior increases performance in Sokoban, but using one suffices for achieving a practically functional algorithm.

Table 4: The success rates (%, higher is better) of HIPS (with PHS*), HIPS-GBFS and HIPS-LevinTS algorithms after performing N node expansions in Sokoban, including the standard errors of the means of the runs.

	Sokoban				
N	50	100	200	500	1000
HIPS-PHS*	**82.9** ± 0.9	**88.5** ± 0.7	**92.1** ± 0.7	**94.7** ± 0.7	**95.9** ± 0.5
HIPS-GBFS	74.4 ± 1.6	78.4 ± 1.5	82.9 ± 1.4	86.5 ± 1.4	89.7 ± 1.2
HIPS-LevinTS	63.8 ± 3.6	78.6 ± 2.3	87.0 ± 1.4	93.7 ± 0.6	95.6 ± 0.3

E. Generalization Ability

To analyze the generalizability of our method, we took the HIPS and HIPS-env agents trained on Sokoban levels with 4 boxes and evaluated them on Sokoban levels with 5 boxes without any fine-tuning, and the results in Table 5 show that the performance stays good and our method has some generalization capability:

Table 5: The success rates (%, higher is better) of HIPS and HIPS-env after performing N node expansions in Sokoban and the final success rate (∞), including the standard errors of the means of the runs.

N	50	100	200	500	1000	∞
HIPS, 4 boxes	82.9 ± 0.9	88.5 ± 0.7	92.1 ± 0.7	94.7 ± 0.7	95.9 ± 0.5	97.5 ± 0.6
HIPS, 5 boxes	67.5 ± 2.1	79.4 ± 1.2	86.0 ± 1.3	91.5 ± 0.8	93.3 ± 0.6	97.1 ± 0.5
HIPS-env, 4 boxes	82.8 ± 0.6	88.5 ± 0.4	91.5 ± 0.4	94.9 ± 0.3	96.4 ± 0.3	98.1 ± 0.4
HIPS-env, 5 boxes	68.6 ± 1.7	77.3 ± 0.9	83.9 ± 1.0	89.9 ± 0.8	92.8 ± 0.8	97.3 ± 0.7

F. Complete Results

Table 6: The overall success rates (%) of different algorithms including the standard errors of the means of the runs. The algorithms in the bottom part have access to the true environment dynamics, and those in the upper part do not.

Method	Sokoban	STP	BW	TSP
HIPS (ours)	97.5 $\pm$ 0.6	**94.7** $\pm$ 1.0	**55.7** $\pm$ 2.4	**100.0** $\pm$ 0.0
HIPS-k (ours)	**99.0** $\pm$ 0.3	**94.7** $\pm$ 1.5	20.1 $\pm$ 1.4	**100.0** $\pm$ 0.0
BC	18.7 $\pm$ 0.7	82.5 $\pm$ 2.2	41.1 $\pm$ 1.6	28.8 $\pm$ 8.5
CQL	3.3 $\pm$ 0.4	11.7 $\pm$ 3.3	6.0 $\pm$ 1.0	33.6 $\pm$ 2.6
DT	36.7 $\pm$ 1.2	0.0 $\pm$ 0.0	10.0 $\pm$ 2.3	0.0 $\pm$ 0.0
RIS	0.0 $\pm$ 0.0	0.0 $\pm$ 0.0	7.3 $\pm$ 2.2	0.0 $\pm$ 0.0
Option-GAIL	0.3 $\pm$ 0.3	0.0 $\pm$ 0.0	0.0 $\pm$ 0.0	0.0 $\pm$ 0.0
IQ-Learn	0.0 $\pm$ 0.0	0.0 $\pm$ 0.0	0.0 $\pm$ 0.0	0.0 $\pm$ 0.0
HIPS-env (ours)	98.1 $\pm$ 0.4	94.6 $\pm$ 1.0	99.6 $\pm$ 0.2	100.0 $\pm$ 0.0
AdaSubS	91.4 $\pm$ 0.5	0.0 $\pm$ 0.0		22.4 $\pm$ 2.3
kSubS	90.5 $\pm$ 1.0	93.3 $\pm$ 0.9		87.9 $\pm$ 3.8

Table 7: The interquartile means of the success rates (%, higher is better) of HIPS after performing N node expansions, and the final success rate (∞ expansions)

N	50	100	200	500	1000	∞
Sokoban	82.9	88.9	92.8	95.3	96.2	98.1
Sliding Tile Puzzle	79.1	90.6	92.0	92.7	93.1	94.6
N	20	50	100	500	1000	∞
Box-World	56.1	56.2	56.2	56.2	56.2	56.2
Travelling Salesman	58.1	100.0	100.0	100.0	100.0	100.0

Table 8: The success rates of different algorithms (higher is better) and the average number of steps (lower is better) needed to solve TSP. The table also includes the standard errors of the means of the runs.

Method	Success rate (%)	Avg. steps
HIPS-PHS* (ours)	**100.0** $\pm$ 0.0	305.9 $\pm$ 5.4
HIPS-MCTS (ours)	**100.0** $\pm$ 0.0	213.0 $\pm$ 4.4
HIPS-A* (ours)	**100.0** $\pm$ 0.0	**168.2** $\pm$ 3.6
kSubS	87.9 $\pm$ 3.8	268.2 $\pm$ 12.0
CQL	33.6 $\pm$ 2.6	336.9 $\pm$ 4.1
BC	28.8 $\pm$ 8.5	339.3 $\pm$ 3.9
AdaSubS	22.4 $\pm$ 2.3	338.9 $\pm$ 18.6
Teacher	100.0 $\pm$ 0.0	336.5 $\pm$ 0.4
Oracle MCTS	100.0 $\pm$ 0.0	199.9 $\pm$ 0.7
Christofides	100.0 $\pm$ 0.0	139.0 $\pm$ 0.1

Table 9: The mean success rates (%, higher is better) of different search algorithms after performing N node expansions, including the standard errors of the means of the runs.

	Sokoban				
N	50	100	200	500	1000
HIPS (ours)	**82.9** $\pm$ 0.9	**88.5** $\pm$ 0.7	**92.1** $\pm$ 0.7	94.7 $\pm$ 0.7	95.9 $\pm$ 0.5
HIPS-k (ours)	63.1 $\pm$ 1.7	77.7 $\pm$ 1.0	85.8 $\pm$ 0.6	90.9 $\pm$ 0.6	94.3 $\pm$ 0.0
HIPS-env (ours)	82.8 $\pm$ 0.6	**88.5** $\pm$ 0.4	91.5 $\pm$ 0.4	**94.9** $\pm$ 0.3	**96.4** $\pm$ 0.3
HIPS-env-k (ours)	64.3 $\pm$ 1.2	76.9 $\pm$ 0.8	86.3 $\pm$ 1.1	91.3 $\pm$ 1.5	94.0 $\pm$ 1.6
AdaSubS	76.4 $\pm$ 0.5	82.2 $\pm$ 0.5	85.7 $\pm$ 0.6	88.8 $\pm$ 0.4	90.2 $\pm$ 0.5
kSubS	69.1 $\pm$ 2.2	73.1 $\pm$ 2.2	76.3 $\pm$ 1.9	79.9 $\pm$ 2.2	82.8 $\pm$ 1.3
PHS*	0.3 $\pm$ 0.0	1.8 $\pm$ 0.1	14.9 $\pm$ 0.3	76.1 $\pm$ 0.5	91.1 $\pm$ 0.3
	Sliding Tile Puzzle				
N	50	100	200	500	1000
HIPS (ours)	**78.7** $\pm$ 2.5	89.8 $\pm$ 1.9	91.1 $\pm$ 1.7	91.8 $\pm$ 1.7	92.2 $\pm$ 1.7
HIPS-k (ours)	10.3 $\pm$ 1.2	68.0 $\pm$ 3.9	76.0 $\pm$ 3.8	79.1 $\pm$ 3.5	84.4 $\pm$ 1.6
HIPS-env (ours)	78.6 $\pm$ 2.6	**90.2** $\pm$ 1.8	**91.4** $\pm$ 1.6	**92.1** $\pm$ 1.7	92.6 $\pm$ 1.6
HIPS-env-k (ours)	10.5 $\pm$ 1.8	69.7 $\pm$ 2.2	76.6 $\pm$ 3.0	80.9 $\pm$ 2.9	87.2 $\pm$ 2.3
AdaSubS	0.0 $\pm$ 0.0	0.0 $\pm$ 0.0	0.0 $\pm$ 0.0	0.0 $\pm$ 0.0	0.0 $\pm$ 0.0
kSubS	0.7 $\pm$ 0.2	79.9 $\pm$ 3.1	89.8 $\pm$ 1.5	91.7 $\pm$ 1.3	**92.7** $\pm$ 1.1
PHS*	0.0 $\pm$ 0.0	0.0 $\pm$ 0.0	0.0 $\pm$ 0.0	0.0 $\pm$ 0.0	0.0 $\pm$ 0.0

	Box-World				
N	20	50	100	500	1000
HIPS (ours)	55.6 $\pm$ 2.3	55.7 $\pm$ 2.3	55.7 $\pm$ 2.3	55.7 $\pm$ 2.3	55.7 $\pm$ 2.3
HIPS-k (ours)	20.1 $\pm$ 1.3	20.1 $\pm$ 1.3	20.1 $\pm$ 1.3	20.1 $\pm$ 1.3	20.1 $\pm$ 1.3
HIPS-env (ours)	**99.3** $\pm$ 0.2	**99.6** $\pm$ 0.2	**99.6** $\pm$ 0.2	**99.6** $\pm$ 0.2	**99.6** $\pm$ 0.2
HIPS-env-k (ours)	89.1 $\pm$ 1.7	98.9 $\pm$ 0.1	99.1 $\pm$ 0.2	99.1 $\pm$ 0.2	99.1 $\pm$ 0.2
PHS*	0.6 $\pm$ 0.2	5.4 $\pm$ 0.5	25.8 $\pm$ 1.0	93.8 $\pm$ 0.7	94.0 $\pm$ 0.7

	Travelling Salesman				
N	20	50	100	500	1000
HIPS (ours)	52.9 $\pm$ 13.3	**99.9** $\pm$ 0.1	**100.0** $\pm$ 0.0	**100.0** $\pm$ 0.0	**100.0** $\pm$ 0.0
HIPS-k (ours)	0.0 $\pm$ 0.0	12.5 $\pm$ 1.4	77.3 $\pm$ 1.8	**100.0** $\pm$ 0.0	**100.0** $\pm$ 0.0
HIPS-env (ours)	**54.3** $\pm$ 13.7	**99.9** $\pm$ 0.1	**100.0** $\pm$ 0.0	**100.0** $\pm$ 0.0	**100.0** $\pm$ 0.0
HIPS-env-k (ours)	0.0 $\pm$ 0.0	12.9 $\pm$ 1.7	77.2 $\pm$ 1.6	**100.0** $\pm$ 0.0	**100.0** $\pm$ 0.0
AdaSubS	0.0 $\pm$ 0.0	0.0 $\pm$ 0.0	1.2 $\pm$ 0.6	12.2 $\pm$ 1.6	20.8 $\pm$ 1.2
kSubS	0.0 $\pm$ 0.0	1.5 $\pm$ 0.6	40.4 $\pm$ 11.1	80.9 $\pm$ 6.8	85.3 $\pm$ 5.3
PHS*	0.0 $\pm$ 0.0	0.0 $\pm$ 0.0	0.0 $\pm$ 0.0	0.0 $\pm$ 0.0	0.0 $\pm$ 0.0

G. Ablation: HIPS-k

Table 10: The success rates (%, higher is better) of different variants of HIPS in STP after performing N node expansions

N	100	500	1000	∞
HIPS-PHS*	55.8 $\pm$ 4.5	**94.6** $\pm$ 0.8	**94.8** $\pm$ 0.9	**94.8** $\pm$ 0.9
HIPS-GBFS	**89.8** $\pm$ 1.9	91.8 $\pm$ 1.7	92.2 $\pm$ 1.7	94.7 $\pm$ 1.0
HIPS-GBFS-3	N/A	N/A	N/A	N/A
HIPS-GBFS-5	68.1 $\pm$ 3.9	79.1 $\pm$ 3.2	84.4 $\pm$ 1.4	94.7 $\pm$ 1.4
HIPS-GBFS-7	84.3 $\pm$ 0.4	85.6 $\pm$ 0.5	86.1 $\pm$ 0.6	86.4 $\pm$ 0.5
HIPS-GBFS-9	76.4 $\pm$ 0.8	76.4 $\pm$ 0.8	76.4 $\pm$ 0.8	76.4 $\pm$ 0.8

When we replace the detector d_ξ with subgoals sampled at fixed intervals, we re-train the low-level policy π_θ to achieve these subgoals. The discrete VQVAE, including the prior, are also re-trained using the new pairs of consecutive subgoals. The distance between the subgoals can, in this case, be controlled by a hyperparameter k. In Sokoban, STP, and Box-World, we used ten as the subgoal horizon and five as k (see Table 12). In TSP, selecting a segment length half of the subgoal horizon proved to be too much, so we let k be equal to four, the default segment length used in kSubS (Czechowski et al., 2021).

We performed a small ablation study in STP to analyze the impact of the value of k. The results are shown in Table 10. We see that given a large research budget, PHS* is slightly superior to GBFS, but GBFS outperforms PHS* given a small search budget. HIPS-GBFS-3 doesn't converge because the value function is noisy. Using a larger k allows the value function to "leap over" the noise, as observed by Czechowski et al. (2021). We see that using a larger k improves the percentage of puzzles solved after a smaller number of expansions, but hurts the overall solution rate, as the search space is explored less systematically. However, using a k too large is also harmful as training the generative model becomes difficult. Furthermore, no value of k was able to outperform HIPS-GBFS, which highlights the benefits of our method that can propose subgoals at different distances adaptively in all environments (see Figure 5).

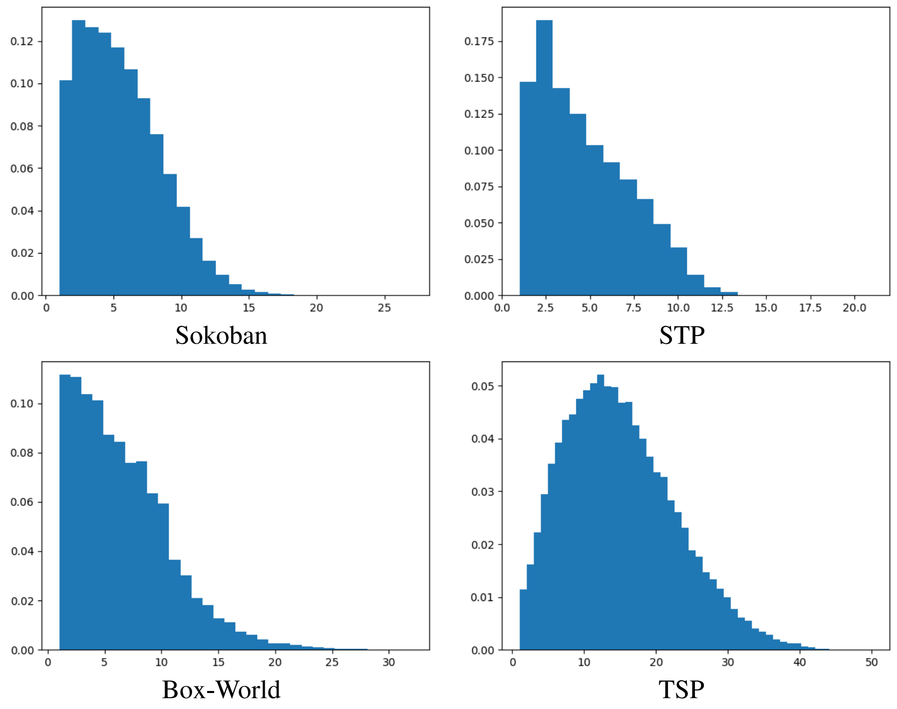


Figure 5: Lengths of the subtrajectories to reach subgoals proposed by VQVAE when it has been trained using the detector d_ξ.

H. Search Methods

Greedy Best-First Search (GBFS) is a priority queue -based search algorithm, where the evaluation function has been defined as

$$\varphi(n) = h(n),$$

where $h(n)$ is a heuristic that predicts the distance to the goal. The node that is predicted to be the closest to the goal is expanded.

Policy-Guided Heuristic Search (PHS) is a policy-guided search algorithm (Orseau & Lelis, 2021) which uses a priority queue with the evaluation function

$$\varphi(n) = \eta(n)g(n)/\pi(n),$$

where $g(n)$ is the path cost from the root to node n, $\pi(n)$ is the node policy (the probability of selecting node n) and $\eta(n)$ is a heuristic factor whose purpose is to estimate the cost to the nearest descendant solution node. We use a variant of PHS, PHS* where the heuristic factor has been defined as

$$\eta_h(n) = \frac{1 + h(n)/g(n)}{\pi(n)^{h(n)/g(n)}} \, . \tag{4}$$

A* is a heuristic-based search algorithm that tries to find the shortest path to the goal (Hart et al., 1968). It is based on a priority queue with the evaluation function

$$\varphi(n) = g(n) + h(n),$$

where $g(n)$ is the distance from the root to node n and $h(n)$ is a heuristic that predicts the distance from the node n to the goal. If the heuristic $h(n)$ never overestimates the true distance to the goal, the heuristic is said to be admissible, and A* is guaranteed to find the shortest path.

Monte Carlo Tree Search (MCTS) is a tree-based search method based on expanding a search tree by performing Monte Carlo evaluations. The selection of nodes for expansion is biased towards promising nodes to enable MCTS to focus on the relevant parts of the search tree. The most commonly used method is UCT, where nodes with higher rewards and lower visitation frequency get the highest priority (Ozair et al., 2021; Kocsis & Szepesvári, 2006).

Listing 1 contains the pseudo-code describing our high-level search with a priority queue. Subroutine `phs_cost` calculates the value of the heuristic factor $\eta(n)$ for PHS* as in (4). Subroutine `extract_plan` collects the subgoals on the path to the leaf node (terminal state) from the root (initial state). Subroutine `get_distances` takes as input the current state and the proposed children and tries to reach them using the subgoal-conditioned low-level policy π_θ. The distances between the state and the children are recovered simultaneously.

Listing 1 PyTorch pseudocode for the high-level search

```
def get_priority(node, alg):
    if alg == 'phs_star':
        return phs_cost(node.log_p, node.value, node.cum_dist)
    elif alg == 'gbfs':
        return node.value
    elif alg == 'a_star':
        return node.value + node.cum_dist


def init_node(node, alg, vqvae, policy, value_func, dynamics):
    node.value = value_func(node.state)
    node.child_states = vqvae.generate(node.state)
    node.distances_to_children = get_distances(
        node.state,
        node.child_states,
        policy,
        dynamics)
    node.filter_unreachable_children()   # Uses the distances computed
    node.children_log_probs = vqvae.prior(node.state)
    node.priority = get_priority(node, alg)


def search(state, alg, vqvae, policy, value_func, dynamics):
    n_nodes = 0
    queue = PriorityQueue()   # Create empty priority queue
    expanded = Set()   # Create empty set
    node = Node(
        state,
        parent=None,
        cum_dist=0,
        log_p=0,
    )
    init_node(node, alg, vqvae, policy, value_func, dynamics)
    queue.insert(node)

    while len(queue) > 0:
        node = queue.pop()
        expanded.add(node.state)
        n_nodes += 1
        for c_state, c_dist, c_log_p in zip(node.child_states,
                                            node.distances_to_children,
                                            node.children_log_probs):
            if c_state in expanded:
                continue
            new_node = Node(
                c_state,
                parent=node,
                cum_dist=node.cum_dist + c_dist,
                log_p=node.log_p + c_log_p,
            )
            if is_terminal(c_state):
                return extract_plan(new_node), n_nodes   # Success
            init_node(new_node, alg, vqvae, policy, value_func, dynamics)
            queue.insert(new_node)
    return None, n_nodes   # Search failed, queue empty
```

I. VQVAE Training

Algorithm 2 Training VQVAE for Subgoal Generation

Input: A dataset of trajectories $\mathcal{D}$, a dataset of subgoal pairs $\mathcal{D}'$, untrained encoder f_ϕ, decoder g_ψ, codebook $\{e_k\}$
Parameters: The codebook $\{e_k\}$ and the parameters of the encoder, ϕ, and the decoder, ψ
Output: Trained encoder f_ϕ, decoder g_ψ, codebook $\{e_k\}$.

```
1: while $f_\phi$ and $g_\psi$ not converged do
2:     Sample a trajectory $\tau$ from $\mathcal{D}$
3:     For each state $s_i \in \tau$, uniformly sample a pair $s_j$ from $(s_{i+1}, \dots, s_H)$
4:     Reconstruct $s_j$ without discretization: $\hat{s}_j = g_\psi(f_\phi(s_j, s_i), s_i)$
5:     Compute reconstruction loss $\mathcal{L}_{\text{rec}}(\hat{s}_j, s_j)$
6:     Update $\phi$, $\psi$ to minimize reconstruction loss
7: end while
8: Sample subsequent subgoal pairs $s_{g_j}$, $s_{g_{j-1}}$ from $\mathcal{D}'$ and encode them with the encoder: $z_j = f_\phi(s_{g_j}, s_{g_{j-1}})$
9: Initialize $\{e_k\}$ as the clusters centers obtained by running KMeans++ on the encodings $\{z_j\}$
10: while $\{e_k\}$, $f_\phi$ and $g_\psi$ not converged do
11:     Sample a batch of subsequent subgoal pairs $s_{g_j}$, $s_{g_{j-1}}$ from $\mathcal{D}'$
12:     Reconstruct subgoals with VQVAE
            $z_j = f_\phi(s_{g_j}, s_{g_{j-1}})$
            $k_j = \arg\min_m \|z_j - e_m\|_2$
            $\hat{s}_{g_j} = g_\psi(e_{k_j}, s_{g_{j-1}})$
13:     Compute the loss in (3)
14:     Update $\phi$, $\psi$, $\{e_k\}$ to minimize the loss computed in Step 13.
15: end while
16: return $f_\phi$, $g_\psi$, $\{e_k\}$
```

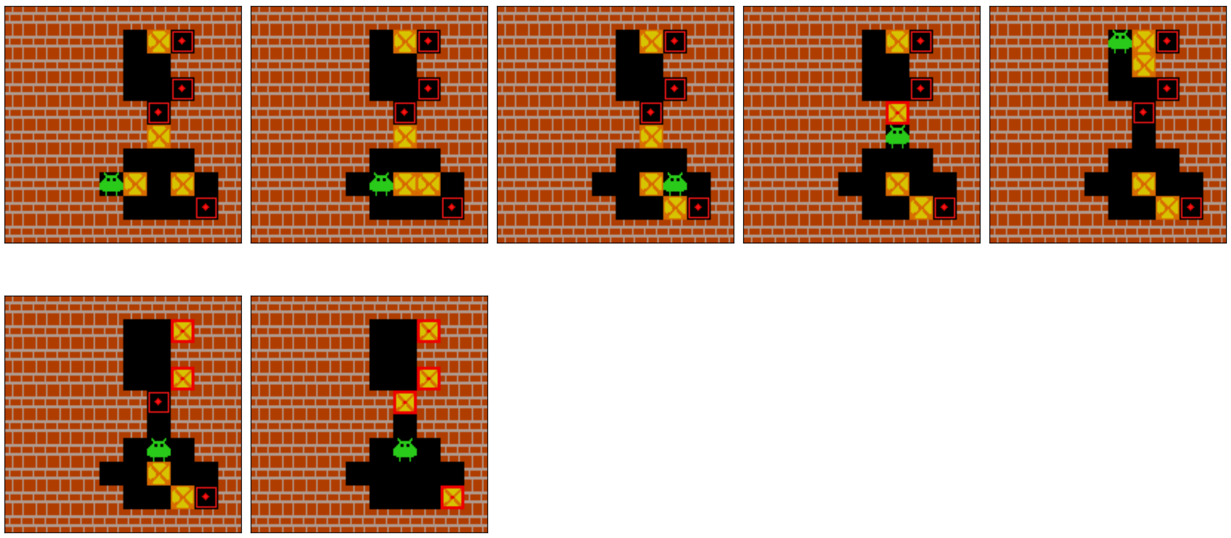
J. Hyperparameters and Visualizations

Table 11: General hyperparameters of our method.

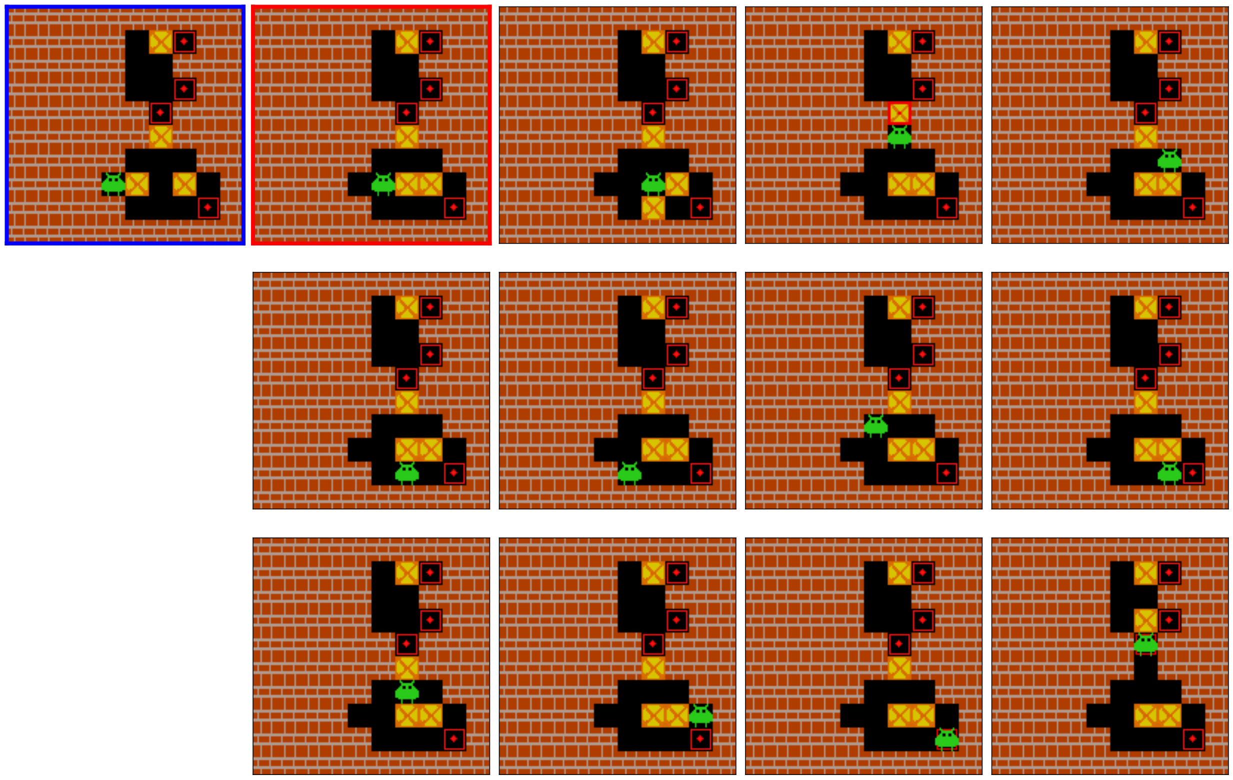
Parameter	Value
Learning rate for dynamics	$2 \cdot 10^{-4}$
Learning rate for π, d, V	$1 \cdot 10^{-3}$
Learning rate for VQVAE	$2 \cdot 10^{-4}$
Discount rate for REINFORCE	0.99

Table 12: Environment-specific hyperparameters of our method.

Parameter	Explanation	Sokoban	STP	Box-World	TSP
α	Subgoal penalty	0.1	0.1	0.1	0.05
β	Beta for VQVAE	0.1	0.1	0.1	0
c	Exploration constant for MCTS	–	–	–	0.1
D	Codebook dimensionality	128	128	128	64
H	Subgoal horizon	10	10	10	50
K	VQVAE codebook size	64	64	64	32
k	Segment length w/o REINFORCE	5	5	5	4
(N, D)	DRC size	–	–	(3, 3)	–

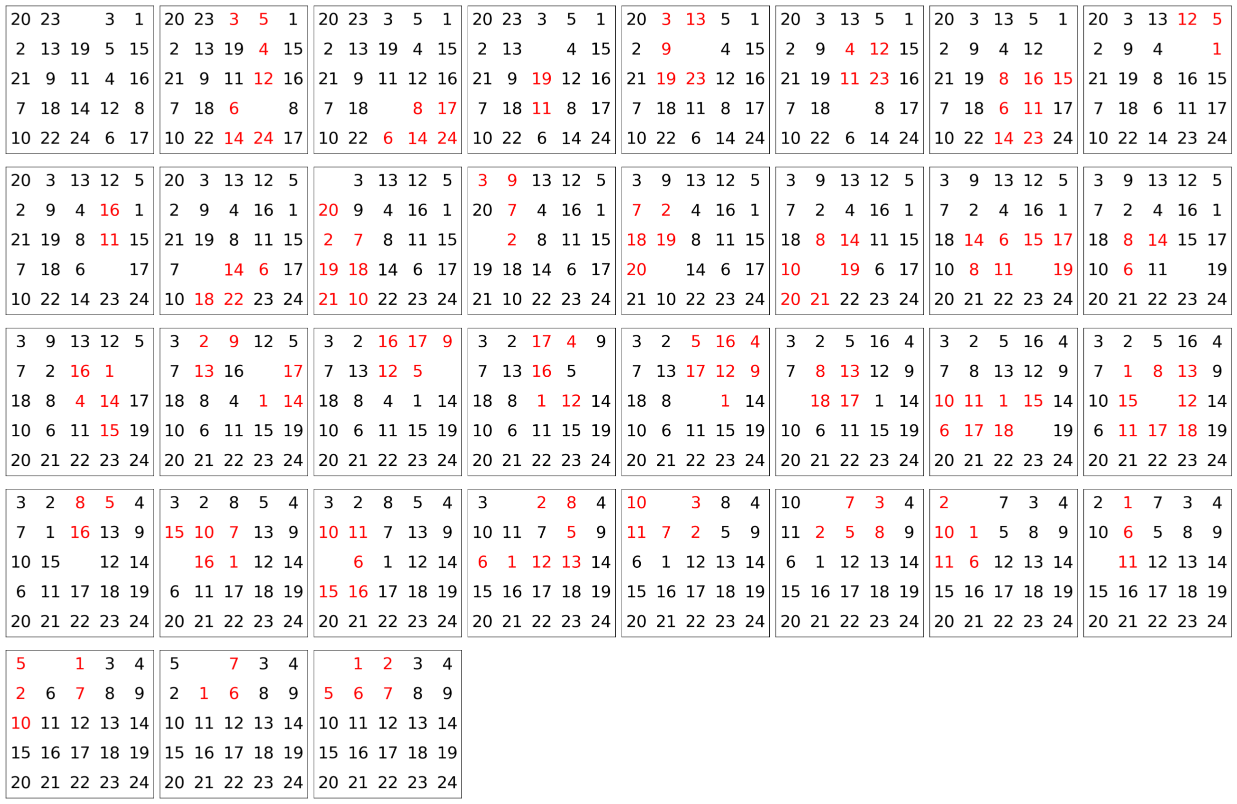


(a) A subgoal-level plan.

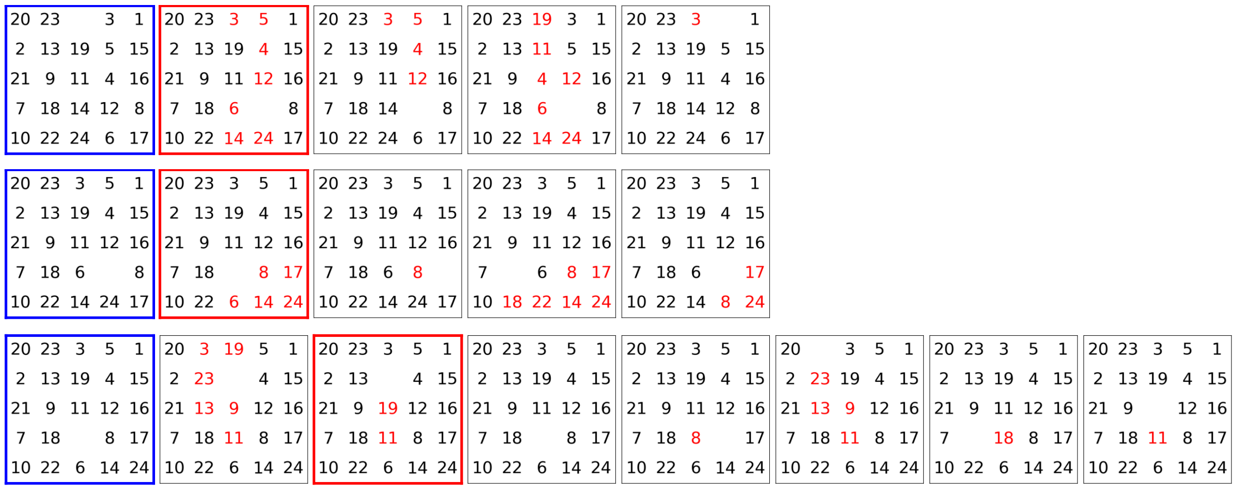


(b) Examples of generated subgoals.

Figure 6: Visualization of the solution found by HIPS for a Sokoban problem. (a): A subgoal-level plan found by HIPS. (b): Subgoals proposed for an intermediate state (marked with blue boundaries). The subgoals have been sorted according to the prior probabilities. The subgoal selected for the final plan is marked with red boundaries.

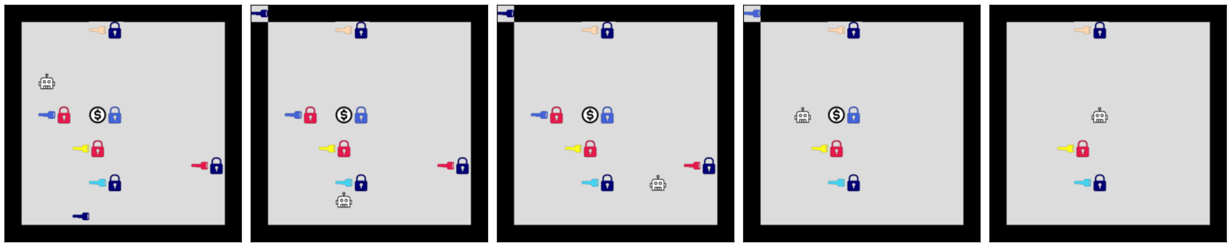


(a) A subgoal-level plan.



(b) Examples of generated subgoals.

Figure 7: Visualization of the solution found by HIPS for an STP problem. (a): A subgoal-level plan found by HIPS. (b): Subgoals proposed for intermediate states (marked with blue boundaries). The subgoals have been sorted according to the prior probabilities. The subgoal selected for the final plan is marked with red boundaries. Red color is used to highlight the tiles which are different from the reference state (the previous state in (a) and the current state in (b)).


(a) A subgoal-level plan.

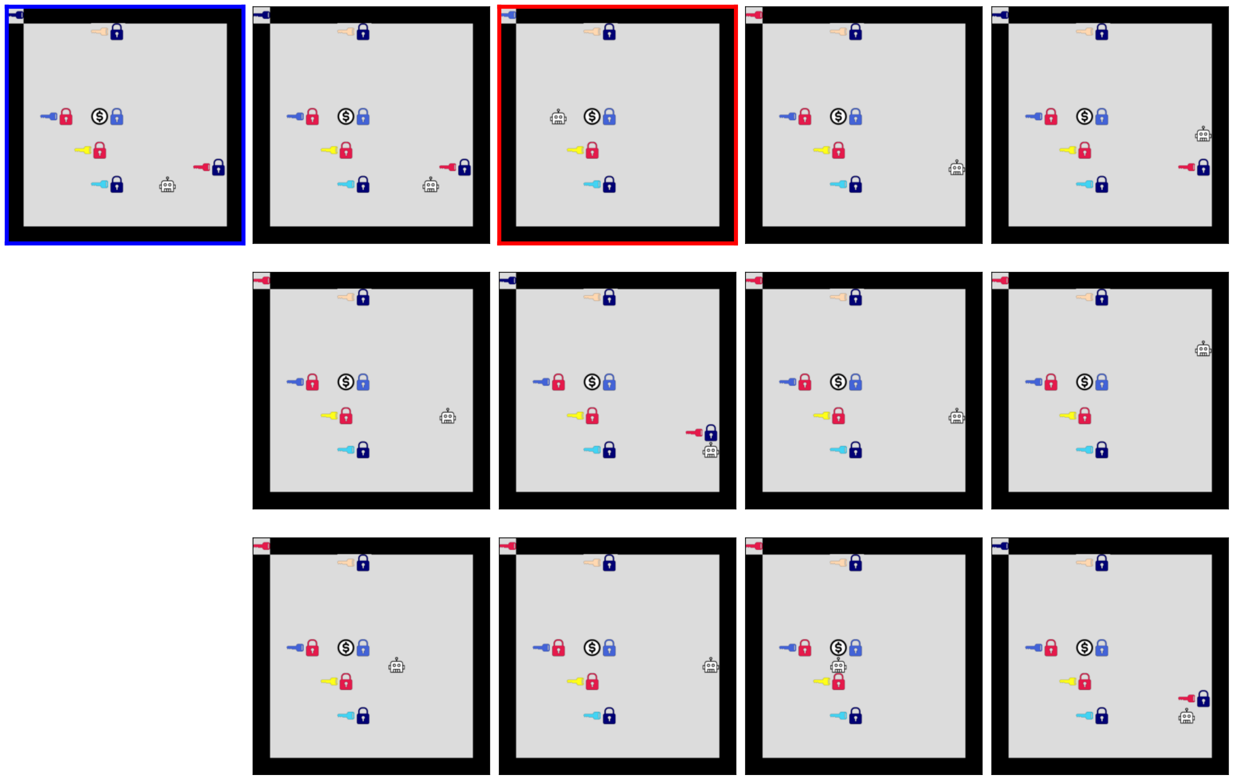
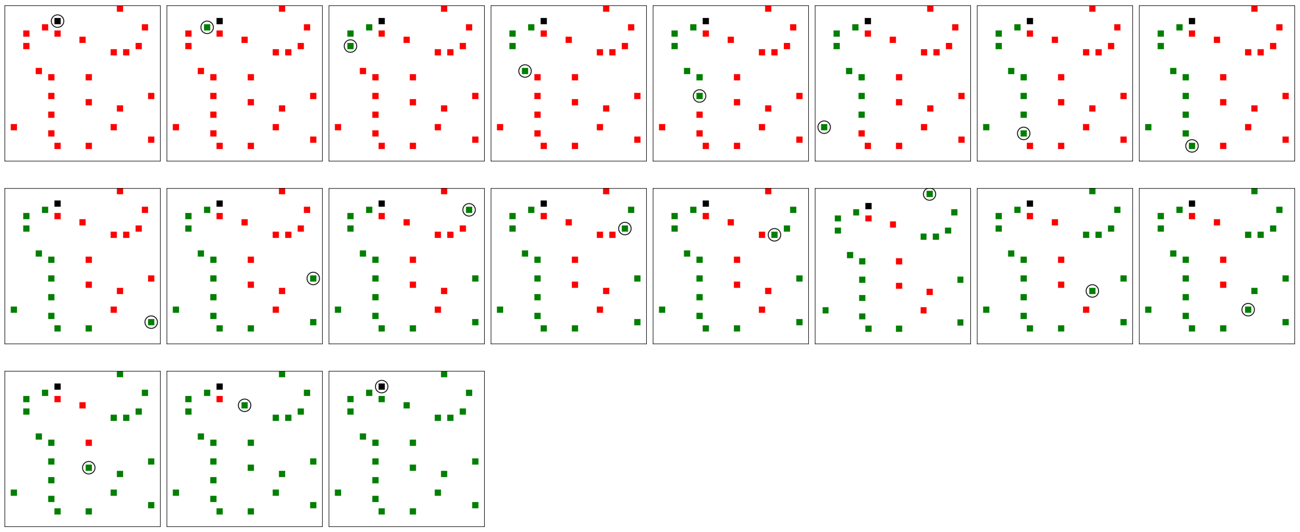

(b) Examples of generated subgoals.

Figure 8: Visualization of the solution found by HIPS for a BW problem. (a): A subgoal-level plan found by HIPS. (b): Subgoals proposed for an intermediate state (marked with blue boundaries). The subgoals have been sorted according to the prior probabilities. The subgoal selected for the final plan is marked with red boundaries.


(a) A subgoal-level plan.

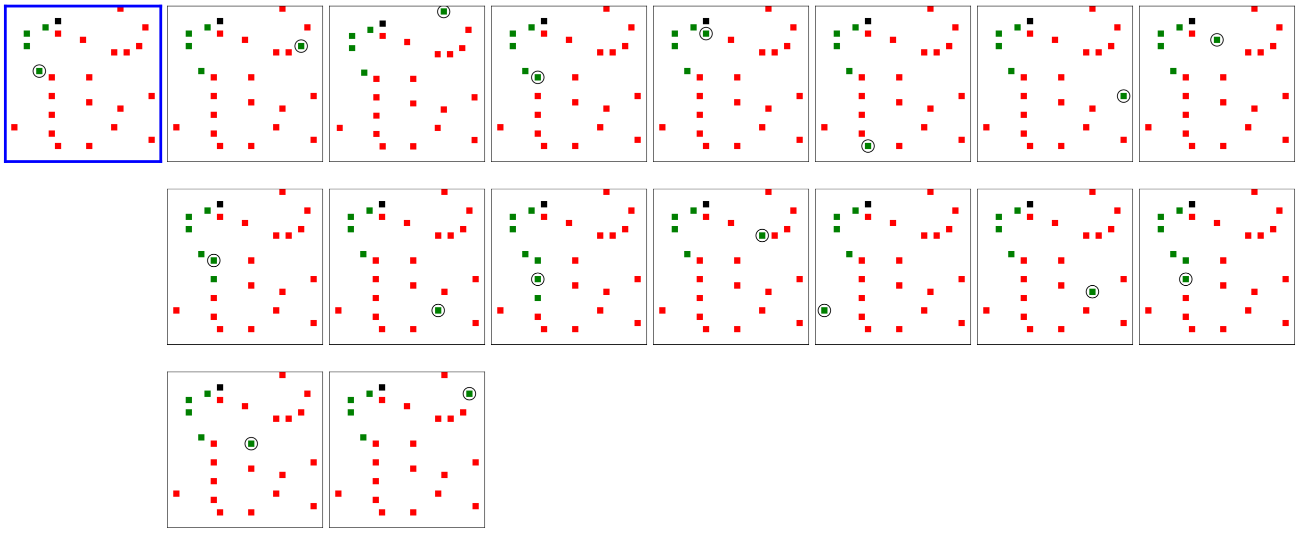

(b) Examples of generated subgoals.

Figure 9: Visualization of the solution found by HIPS for a TSP instance. (a): A subgoal-level plan found by HIPS. (b): Subgoals proposed for an intermediate state (marked with blue boundaries).