Value-based Subgoal Discovery and Path Planning for Reaching Long-Horizon Goals

Shubham Pateria, Budhitama Subagdja, Ah-Hwee Tan, *Senior Member, IEEE* and Chai Quek, *Senior Member, IEEE*

***Abstract*—Learning to reach long-horizon goals in spatial traversal tasks is a significant challenge for autonomous agents. Recent subgoal graph-based planning methods address this challenge by decomposing a goal into a sequence of shorter-horizon subgoals. These methods, however, use arbitrary heuristics for sampling or discovering subgoals, which may not conform to the cumulative reward distribution. Moreover, they are prone to learning erroneous connections (edges) between subgoals, especially those lying across obstacles. To address these issues, this paper proposes a novel subgoal graph-based planning method called Learning Subgoal Graph using Value-based Subgoal Discovery and Automatic Pruning (LSGVP). The proposed method uses a subgoal discovery heuristic that is based on a cumulative reward (value) measure and yields sparse subgoals, including those lying on the higher cumulative reward paths. Moreover, LSGVP guides the agent to automatically prune the learnt subgoal graph to remove the erroneous edges. The combination of these novel features helps the LSGVP agent to achieve higher cumulative positive rewards than other subgoal sampling or discovery heuristics, as well as higher goal-reaching success rates than other state-of-the-art subgoal graph-based planning methods.**

***Index Terms*—Long-horizon Goal-reaching, Subgoal Discovery, Subgoal Graph, Reinforcement Learning, Motion Planning, Path Planning**

I. Introduction

Optimally performing long-horizon goal-reaching tasks is a significant challenge for autonomous agents. Such a task requires an agent to execute a long sequence of actions in a large state space to reach a desired goal. In this paper, we focus on the tasks involving spatial traversal while avoiding obstacles to reach long-horizon goals. Model-free Reinforcement Learning (RL) methods [1], [2] perform poorly on such tasks due to the complexity of finding the optimal policy over long timescales. Model-free Hierarchical Reinforcement Learning (HRL) [3] enables long-horizon goal-reaching by learning a hierarchy of policies to decompose a goal into a sequence of shorter-horizon subgoals that are easier to reach [4], [5]. Here, a subgoal might be an original state or an abstract state. This approach is effective for learning in an unknown and stochastic environment, but requires a large amount of data for training the agent to reach different goals.

S. Pateria, B. Subagdja and A.H. Tan are with the School of Computing and Information Systems, Singapore Management University, Singapore, 178902 SG (e-mail: shubhamp@smu.edu.sg, budhitamas@smu.edu.sg, ahtan@smu.edu.sg).

C. Quek is with the School of Computer Science and Engineering, Nanyang Technological University, Singapore, 639798 SG (e-mail: ashcquek@ntu.edu.sg).

Recent existing subgoal graph-based planning methods [6]–[9] provide a more data efficient approach by learning a higher-level model of the environment in the form of a subgoal graph, in which the nodes and edges represent subgoals and distances between them, respectively. Planning the optimal sequences of subgoals can then be conducted over this graph to reach long-horizon goals, while the primitive actions to reach subgoals can be selected by model-free Reinforcement Learning (RL) policy. Those methods, however, suffer from one or both of the following two important issues:

Firstly, they might not be suitable for environments with non-uniform distribution of rewards across different regions of the state space as they rely on reward-agnostic heuristics for discovering important subgoals. Such heuristics fail to discriminate the subgoals lying on higher cumulative reward paths (to various goals) from other candidate subgoals. *Secondly*, they may plan sequences of subgoals that are infeasible to achieve due to erroneous edges without any rigorous procedure to update or remove them. This issue might occur especially when erroneous edges are formed between pairs of subgoals lying across the obstacles.

This paper proposes a novel subgoal graph-based planning method, called **L**earning **S**ubgoal **G**raph using **V**alue-based Subgoal Discovery and Automatic **P**runing (LSGVP), to address the two specific issues above. The key contributions of this work are as follows:

- LSGVP is based on a novel subgoal discovery heuristic proposed in this paper (subsection IV-B), which discriminates between different states as candidate subgoals by using a measure called *path value*. The *path value* of a candidate subgoal is calculated with respect to a pair of start and goal states, and it is proportional to the *predicted* average cumulative reward if the agent traverses from the start state to the goal through the candidate subgoal. This helps in discriminating between more valuable candidate subgoals and less rewarding ones. The discovered subgoals become the nodes of the LSGVP subgoal graph.
- We also propose an automatic graph pruning procedure as a key component of LSGVP (subsection IV-E). This procedure guides the agent to perform back-and-forth edge traversals across the subgoal graph to automatically prune the erroneous edges formed due to mis-predicted distances, thereby reducing the infeasible plans.

We compare LSGVP with other subgoal discovery heuristics and various state-of-the-art subgoal graph-based planning

methods [6]–[9] in both two-dimensional maze navigation [6] and high-dimensional visual navigation (VizDoom [10]) domains. In the experiments, LSGVP achieves higher average cumulative rewards and higher goal-reaching success rates compared to other methods, validating the benefits of value-based subgoal discovery and graph pruning. LSGVP is also compared with a state-of-the-art model-free HRL method, called Hierarchical Actor Critic (HAC) [4], in the MuJoCo continuous control domains [11]. The results show that LSGVP learns to reach long-horizon goals (via planning) in a more data efficient manner than HAC. This result is discussed in Appendix C of the Supplementary Document.

The organization of the rest of this paper is as follows: The related work is reviewed in Section II. The essential preliminary concepts, problem scope, and problem definition are provided in Section III. The proposed method is described in Section IV. Section V provides the details, results, and analysis related to the experiments outlined above. Finally, the conclusion of the paper is provided in Section VI.

II. Related Work

Our work is related to subgoal graph-based planning. Various methods have recently been introduced in this category [6]–[9]. Among these methods, Search on the Replay Buffer (SoRB) [6] constructs a graph that connects subgoal states which are uniformly sampled from a replay buffer filled during the agent's training. This graph is used to find the shortest path to reach long-horizon goals. Huang et al. [8] proposed an approach to learn sparse subgoal graphs by using the Farthest Point Sampling (FPS) heuristic [12]. FPS samples subgoal states that are situated farther from each other, thereby improving the coverage of the state space. Semi Parametric Topological Memory (SPTM) [7] constructs a subgoal graph based on the predicted temporal proximity of different subgoal states. These subgoal states are sampled at uniform intervals from human demonstration trajectories. Recently, Hu et al. [13] proposed a method called episodic memory-based topological mapping (*e*-TM), to learn sparse subgoal graphs using fusion adaptive resonance theory (ART) networks [14].

In the visual navigation domains, various methods have been proposed to learn topological or semantic maps for planning. Among such methods, MapNet [15] is a deep learning based Simultaneous Localization and Mapping (SLAM) method that learns *allocentric* spatial memory in a 2.5D representation space, in which any information related to the vertical dimension is implicitly encoded in a dense 2D field representing the ground. Chaplot et al. [16] proposed Active Neural SLAM (ANS) which uses a hierarchical structure for planning. It learns a free-space geometric map including the agent's estimated pose. A global policy takes this map and learns to exploit structural regularities to produce long-term goals, which are then used to generate short-term subgoals for a local action-selection policy using a geometric path-planner. Neural Topological SLAM [17] learns topological graph in which the nodes denote areas in a map and the edges denote spatial relations. It leverages learnt semantic features to localize visual observations to graph nodes. If an observation is not localized to any node, a graph update module adds a new node into the topological memory. Lv et al. [18] introduced an approach to integrate 3D knowledge graph with Deep Reinforcement Learning (DRL). Their method extracts 3D spatial relationships between objects observed during agent's exploration to form graphs, then apply graph convolutional networks to obtain node features for the established graph. This graph is utilized to navigate to various target locations based on the learned inter-object relations and to adapt to various variations in novel or unfamiliar environments.

Planning can also be done using learnt parametric models of state-action transitions [19]–[21]. In such method, subgoals are directly sampled from the transition model. Nair et al. [19] proposed a method for planning over a *latent encoding of visual state space* at multiple levels. This method is called Hierarchical Visual Foresight (HVF). In HVF, an agent learns a *parametric state-action transition model* and samples a sequence of subgoals using this model such that the cost of planning from a start state to a goal, over those subgoals, is minimized. Pertsch et al. [21] proposed a similar method to find optimal sequence of subgoal states, which they call *keyframes*. They use a two-level probabilistic prediction model, where the higher-level model predicts a sequence of keyframes (subgoals) and the lower-level model fills the sequence of states from one keyframe to another (called *infilling*), up to a goal state.

In contrast to LSGVP, the above-mentioned methods do not use value-based subgoal discovery and lack a mechanism for graph pruning. Sparse Graphical Memory (SGM) [9] is a subgoal graph-based planning method which is more closely related to LSGVP. It is a state abstraction method which estimates the dissimilarity of different states as start or goal nodes (not subgoals) based on their goal-conditioned Q-values. On the other hand, LSGVP identifies salient states by directly considering them as candidate subgoals between multiple start-goal pairs and discriminating based on their *path values* (discussed in subsection IV-B). SGM also uses an automatic graph pruning heuristic based on traversals to random goals by following the edges of the subgoal graph (along shortest path). If the agent cannot traverse a particular edge, it is treated as erroneous edge and removed. In contrast, LSGVP uses a more rigorous back-and-forth edge traversal to prune the erroneous edges, discussed later.

III. Preliminaries

A. Universal Markov Decision Process

The basic problem considered in this paper is *episodic goal-reaching*, in which an agent must learn to reach different goals in different episodes but within the same environment. This problem can be formally defined in terms of a Universal Markov Decision Process (UMDP) [4], [6], [22]. A UMDP consists of a state space $\mathbb{S}$, an action space $\mathbb{A}$, a set of goals $\mathbb{G}$, and a reward $R(s, a|G)$ for taking an action $a \in \mathbb{A}$ in a state $s \in \mathbb{S}$ conditioned on a goal $G \in \mathbb{G}$. The expected cumulative

reward after taking action a in state s is represented by the following goal-conditioned Q-value function [22],

$$Q^{\pi}(s,a|G) = \mathbb{E}_{a_t \sim \pi(a_t|s_t,G)}[\sum_{t=0}^{t=T_G^s} \gamma^t R(s_t,a_t|G)| \quad s_0 = s, a_0 = a] \tag{1}$$

Here, $\pi(a|s,G)$ is a *goal-conditioned policy* which maps a state and a goal to an action, T_G^s is the time horizon over which the goal G is reached when starting from the state s, and $\gamma \in [0,1)$ is a reward discount factor. The objective of the agent is to learn a goal-conditioned policy that maximizes the goal-conditioned Q-value for all state-action pairs.

1) Problem Scope: In the context of this paper, we further refine the problem scope by applying the following constraints:

- The set of goals is a subset of the state space, that is, $\mathbb{G} \subset \mathbb{S}$.
- The reward $R(s,a|G)$ is equal to -1 by default, in addition to which other positive or zero reward values can be defined. This means that the maximization of the Q-value defined in equation 1 is equivalent to finding the *shortest path* to the goal G, in the default case.
- The UMDP is stationary.

B. Planning for Long-horizon Goal Reaching

Learning a goal-conditioned policy becomes challenging in the UMDPs with large state spaces if the time horizon for reaching the goals is very long [23], [24]; this is due to the complexity of finding the optimal policy over long timescales without special exploration techniques [25]. We approach the long-horizon goal-reaching problem from the perspective of decomposing the long-horizon goal into simpler shorter-horizon ones. This involves the training of a goal-conditioned policy using various shorter-horizon goals, treating a few shorter-horizon goals as *subgoals*, learning an inter-subgoal transition model (subgoal graph), and planning over that model to reach various long-horizon goals after the training phase. Henceforth, we use different notations to avoid confusion among state, subgoal, and goal, as follows: s is used as the notation for a state $s \in \mathbb{S}$, g is used as the notation for a subgoal $g \in \mathbb{S}$, and G is used as the notation for a goal $G \in \mathbb{G} \subset \mathbb{S}$.

1) Reward-based Distance Function: For planning to reach long-horizon goals, we have to first define a distance function between pairs of states. We define a distance function that is *inversely* proportional to the maximum Q-value when traversing from one state to another [6], as follows,

$$\mathcal{D}(s_i,s_j) = Q^{+} - \max_{\pi} Q^{\pi}(s_i,a|s_j)\Big| a = \pi(a|s_i,s_j) \tag{2}$$

Here, Q^{+} is a domain-specific positive upper limit of the Q-value, to avoid negative distances. It is used as a hyper-parameter for LSGVP. This distance function has the following key properties:

- In the problem scope defined in subsection III-A1, the predicted distance $\mathcal{D}(s_i,s_j)$ is *proportional* to the average predicted number of steps required for traversing from s_i to s_j since the default reward is -1 at each step.
- Additionally, if there are positive rewards on the traversal path, the predicted distance might be reduced due to a higher predicted Q-value. This implies that higher the cumulative reward on the path, the lower the distance.

2) Planning using Subgoals for Reaching Goals: We now provide the specific problem definition that guides the development of our method, presented in section IV.

Problem Definition: Given a set of goals $\mathbb{G} \subset \mathbb{S}$, a learnt goal-conditioned policy π, and a learnt goal-conditioned Q-value function Q^{π}, find a set of subgoals $\mathbb{G}^{sub} \subset \mathbb{S}$ and plan a subgoal sequence $g_1, g_2, ..., g_k, ..., g_K \subset \mathbb{G}^{sub}$ so as to get the shortest path of traversal from any start state $s \in \mathbb{S}$ to any goal $G \in \mathbb{G}$, that is

$$\min\Big(\mathcal{D}(s,g_1) + ... + \mathcal{D}(g_{k-1},g_k) + ... + \mathcal{D}(g_K,G)\Big). \tag{3}$$

This can be treated as a shortest-path motion (traversal) planning problem in terms of the reward-based distances.

IV. Proposed Method

This section presents our method, LSGVP, for finding subgoals and planning to reach long-horizon goals (Problem Definition, subsection III-B2). The LSGVP agent starts with a phase of *exploration* of the UMDP followed by the *training* of the goal-conditioned policy and Q-value function (subsection IV-A). It collects the observed states into a memory during this phase. It then uses a value-based subgoal discovery heuristic to find useful subgoals out of the states collected in the memory (subsection IV-B). Those subgoals become the nodes of a subgoal graph, in which the edge weights are equal to the predicted distances $\mathcal{D}$ (subsection IV-C). The learnt subgoal graph can be used for planning the shortest paths (sequences of subgoals) to various long-horizon goals encountered during the testing phase. The planning procedure is discussed in subsection IV-D. However, the LSGVP agent also performs automatic graph pruning before the testing phase to avoid infeasible plans (subsection IV-E).

A. Exploration and Training

In the training phase, the LSGVP agent explores the given UMDP in an *episodic* manner. In each episode, the agent is given a randomly sampled *training goal*. If the agent does not reach the original training goal in an episode, the terminal state (s_T) in that episode is treated as the training goal when learning the goal-conditioned policy $\pi(a|s,s_T)$ and the goal-conditioned Q-value function $Q(s,a|s_T)$. If the agent reaches the original training goal, s_T is naturally the same as that goal. This is loosely based on the concept of Hindsight Experience Replay (HER) [26]. As per the approach outlined in subsection III-B, we keep the lengths of the training episodes short to ensure that the training goals have a shorter horizon than the long-horizon goals that might be encountered during the testing phase.

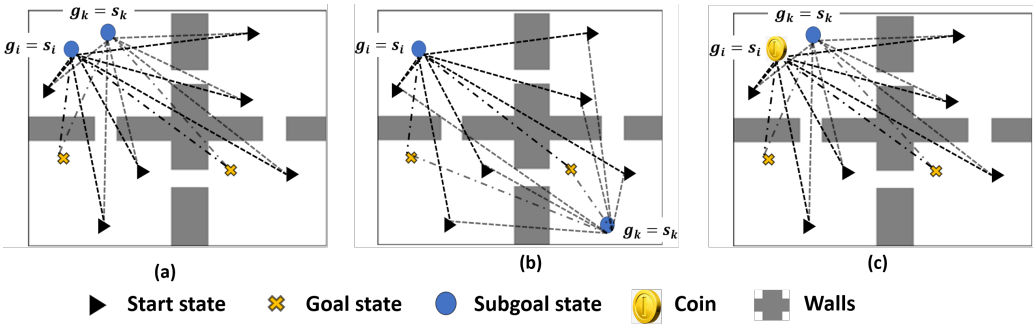


Fig. 1: Examples of (a) similar subgoals due to spatial proximity, (b) dissimilar subgoals due to spatial separations, and (c) dissimilar subgoals due to different rewards obtained, despite spatial proximity. *Coin* represents a state in which the agent can receive higher reward. The dashed lines represent the spatial distances between different states in terms of the number of steps required for traversal.

After each episode, the agent adds the observed states into an experience memory $\mathcal{B}$. Thus, $\mathcal{B} \subset \mathbb{S}$. After the training phase, the learnt policy π is treated as the stationary *primitive action policy* that is used to traverse to various subgoals in the subgoal graph (discussed later) without retraining. The learnt Q-value function is used to estimate the distances as per equation 2.

B. Subgoal Discovery

After the training phase, the agent uses the data present in $\mathcal{B}$ for subgoal discovery and graph construction. The sufficiency of this data is based on the following assumption:

Assumption 1: *The agent extensively explores the state space $\mathbb{S}$ to sufficiently populate memory $\mathcal{B}$, so that it contains the states from all the regions of the state space from which the start states and the test goals might be sampled during the test phase. Essentially, the set of long-horizon test goals $\mathbb{G}$ is a subset of $\mathcal{B}$.*

Under this assumption, we use the memory $\mathcal{B}$ to sample goals that are used for subgoal discovery. The agent populates a set containing *pairs of start and goal states* randomly sampled from $\mathcal{B}$, denoted as $\mathcal{SG} = \{(s, G) \,|\, s \in \mathcal{B}, G \in \mathcal{B}\}$. Then, the agent needs to discover a set of subgoals that satisfy the following two criteria,

- C1: The subgoals should be sparsely distributed over the state space so that planning over the subgoal graph incurs low complexity.
- C2: For consistency with the problem definition (equation 3), the subgoal discovery heuristic should discriminate between the subgoals that lie on higher cumulative reward paths (lower sum of predicted distances) and other candidate subgoals.

This subsection describes the novel subgoal discovery heuristic used in LSGVP to satisfy both criteria. This heuristic discriminates between different states in $\mathcal{B}$ (candidate subgoals) based on a measure named as *path value*. The path value of a state s_i with respect to a start state s and a goal G is defined as

$$\mathcal{PV}(s, s_i, G)_{g=s_i} = -\Big(\mathcal{D}(s, s_i) + \mathcal{D}(s_i, G)\Big). \quad (4)$$

Here, $g = s_i$ implies that s_i is treated as a subgoal. The relation between the predicted distance $\mathcal{D}$ and the Q-value in equation 2 suggests that the path value $\mathcal{PV}(s, s_i, G)$ is proportional to the predicted cumulative reward the agent might obtain when traversing[1] from the state s to the goal G using s_i as the subgoal. Therefore, *the path value is a reward-based measure of the utility of a state as a subgoal* with respect to a certain pair of start and goal states.

Then, the agent must discriminate between various states to discover those which can be treated as salient subgoals, while satisfying criteria C1 and C2. For this, we define a *dissimilarity* measure between two states $s_i \in \mathcal{B}$ and $s_k \in \mathcal{B}$ based on their path values, as follows

$$dsim(s_i, s_k) = \max_{(s,G)\in\mathcal{SG}} \Big|\mathcal{PV}(s, s_i, G) - \mathcal{PV}(s, s_k, G)\Big| \quad (5)$$

The $dsim$ measure is equal to the maximum absolute difference between the path values of two states (candidate subgoals), where the maximum is taken over all the start and goal pairs in $\mathcal{PV}$. This measure is used to discriminate between different states and discover the salient subgoal states. We discuss the motivation for using $dsim$ below.

Let us consider the three examples shown in Figure 1. In the first two examples (sub-figures 1(a) and (b)), there is only a default reward of -1 received at each step (as per the problem scope outlined in subsection III-A1). Therefore, the *lengths* of the dashed lines connecting any pair of start and goal states, that are $\mathcal{D}(s, s_i) + \mathcal{D}(s_i, G)$ and $\mathcal{D}(s, s_k) + \mathcal{D}(s_k, G)$, simply represent the predicted number of steps required to traverse between that pair through the subgoals s_i and s_k, respectively. In the first example (sub-figure 1(a)), s_i and s_k are situated closer to each other. Therefore, all predicted distances are ideally similar for these two subgoals and, as per equation 4, the two subgoals have similar path values for all pairs of s and G. Hence, $dsim(s_i, s_k)$ is small.

In contrast, as the subgoals become distant (sub-figure 1(b)), their path values also become different from each other. This means that $dsim(s_i, s_k)$ becomes larger. Due to such a relationship between the $dsim$ and the spatial separation between two subgoals, we can discover sparsely distributed

[1]Refers to hypothetical traversal path in this case since the function $\mathcal{D}$ provides the *predicted estimate* of distance. This should not be taken as a comment on feasibility of traversal.

subgoals for which $dsim$ is above a certain threshold, thereby satisfying criterion C1.

Although it is also possible to obtain the sparse distribution of subgoals by other means of reward-free measures such as Euclidean distances, the use of $dsim$ is necessary in this case as the reward-based path values must be considered according to criterion C2. Such a case is shown in the third example (subfigure 1(c)). In this example, although s_i and s_k are spatially closer to each other, the agent can receive a positive reward when traversing through s_i. The lengths of the start-to-goal paths (number of steps) are similar for both subgoals, but their path values ($\mathcal{PV}$) are different due to the different rewards accumulated. Therefore, the agent must discriminate between the two subgoals on the basis of their path values instead of path lengths or Euclidean distances. This is satisfied by using the $dsim(s_i, s_k)$ measure for dissimilarity.

In short, we use $dsim$ to discriminate between various states during subgoal discovery to find subgoals that are generally farther from each other (for sparsity criterion C1) but also include subgoals lying on higher cumulative reward paths even if they are closer to other discovered subgoals (criterion C2). Based on this motivation, we define the subgoal discovery heuristic of LSGVP as follows.

Subgoal Discovery Heuristic: At the beginning of subgoal discovery, the LSGVP agent creates a subgoal set $\mathbb{G}^{sub}$ and fills it with one randomly sampled state from the experience memory $\mathcal{B}$, treated as a random subgoal. Then, the following rules are incrementally applied on each $s_i \in \mathcal{B}$ to discover rest of the subgoals,

- RULE 1: If
$$dsim(s_i, g) \leq \epsilon \tag{6}$$
for at least one $g \in \mathbb{G}^{sub}$, then s_i is similar to g and it is not considered a subgoal.
- RULE 2: If
$$dsim(s_i, g) > \epsilon \tag{7}$$
for each $g \in \mathbb{G}^{sub}$, then s_i is dissimilar from all the existing subgoals and it is added to $\mathbb{G}^{sub}$ as a new subgoal.

The hyper-parameter $\epsilon \in [0, \infty)$ in equations 6 and 7 is the *threshold of dissimilarity* beyond which two states are considered unique as subgoals. Its value controls the sparsity of the discovered subgoals. A smaller value of ϵ results in a denser distribution of subgoals, and vice versa.

A state s_i can only satisfy one of the rules. Given the nature of the distance function (equation 2), RULE 1 is generally satisfied by a state which is spatially closer to at least one existing subgoal (in $\mathbb{G}^{sub}$). On the other hand, RULE 2 is generally satisfied by a state which is spatially farther from all the exiting subgoals (in $\mathbb{G}^{sub}$) or it lies on a higher cumulative reward path.

Relating these rules to the examples given in Figure 1, the states s_i and s_k in the first example satisfy RULE 1, hence only one of them can be added to $\mathbb{G}^{sub}$, whereas the states in the second example satisfy RULE 2 and both can be added to $\mathbb{G}^{sub}$. The states in the third example also satisfy RULE 2 despite being spatially closer to each other because of the higher path value difference due to the different rewards obtained upon traversing through those states. The complete subgoal discovery heuristic is summarized in Algorithm 1. The time complexity of the heuristic is $O(|\mathcal{B}|\times|\mathcal{SG}|\times|\mathbb{G}^{sub}|)$. Here, $|\mathbb{G}^{sub}|$ is not predetermined but depends on the number of subgoals discovered.

Algorithm 1 Subgoal Discovery

```
INPUT: Experience memory B ⊂ 𝕊
OUTPUT: Set of subgoals G^sub
Fill a set SG with |SG| randomly sampled pairs of states (s ∈ B) and goals (G ∈ B)
Fill a set G^sub with one random subgoal sampled from B
Initialize ϵ
foreach s_i ∈ B do
    rule2 ← 1
    foreach g ∈ G^sub do
        rule1 ← 1
        foreach (s, G) ∈ SG do
            if |PV(s, s_i, G) − PV(s, g, G)| ≤ ϵ then
                do nothing
            else
                rule1 ← 0
                break
        if rule1 = 1 then
                                   ▷ RULE 1 satisfied for at least one g
            rule2 ← 0
            break
    if rule2 = 1 then
                                   ▷ RULE 2 satisfied
        G^sub ← G^sub + {s_i}      ▷ Subgoal Discovery
```

C. Subgoal Graph Construction

After subgoal discovery, LSGVP adds the subgoals from $\mathbb{G}^{sub}$ as the nodes in a subgoal graph $\mathcal{GR}$. An edge $\mathcal{E}(g_i, g_k)$ is created from a node g_i to another node g_k if the predicted distance $\mathcal{D}(g_i, g_k) \leq MAXDIST$. The predicted distance $\mathcal{D}(g_i, g_k)$ is set as the weight of the edge $\mathcal{E}(g_i, g_k)$. The hyper-parameter $MAXDIST \in [1, \infty)$ is the maximum predicted distance from a node (subgoal) g_i to another node (subgoal) g_k up to which an edge can be created from the former to the latter. Its value controls the number of edges created, that is, $|\mathcal{E}|$.

The $MAXDIST$ limit is set to avoid a case in which a direct edge is formed from a subgoal g_i to another *long-horizon* subgoal g_j (large $\mathcal{D}(g_i, g_j)$), since the agent might fail to reach the long-horizon subgoal g_j using the primitive action policy $\pi(a|s, g_j)$ (g_j treated as a goal) trained to reach shorter-horizon goals (subsection IV-A). The time complexity of graph construction is $O(|\mathbb{G}^{sub}|^2)$.

D. Planning over Subgoal Graph

After the construction of the subgoal graph $\mathcal{GR}$, planning over the subgoals to reach a long-horizon goal is performed as follows (see Algorithm 2). Given a start state s and a long-horizon test goal $G \in \mathbb{G}$, they are firstly added as temporary nodes in $\mathcal{GR}$. Temporary edges are created from s to every subgoal node $g_i \in \mathbb{G}^{sub}$ for which $\mathcal{D}(s, g_i) \leq$ MAXDIST. Temporary edges are also created to the test goal G from every

subgoal node g_i for which $\mathcal{D}(g_i, G) \leq$ MAXDIST. Then, the shortest path from s to G is found using Dijkstra's algorithm.

Algorithm 2 Planning and Traversal to a Goal

INPUT: Subgoal graph $\mathcal{GR}$
OUTPUT: Sequence of subgoals ('path') for traversal to Goal
def *SubgoalPlan(s, G)*:
    foreach $s_i^g \in \mathcal{GR}$ **do**
        if $\mathcal{D}(s, s_i^g) \leq$ *MAXDIST* **then**
            Add a temporary edge $\mathcal{E}(s, s_i^g)$ to $\mathcal{GR}$ with weight equal to $\mathcal{D}(s, s_i^g)$
        if $\mathcal{D}(s_i^g, G) \leq$ *MAXDIST* **then**
            Add a temporary edge $\mathcal{E}(s_i^g, G)$ to $\mathcal{GR}$ with weight equal to $\mathcal{D}(s_i^g, G)$
    path ← DIJKSTRA($\mathcal{GR}$, source = s, target = G)
    if *path* = $\emptyset$ **then**
        path ← s, G
    return path
$G \leftarrow Goal$, s← current state, path ← **SubgoalPlan**(s, G), steps ← 0
while *episode not terminated* **do**
    $g \leftarrow$ path{1}, $s \leftarrow$ current state, $a \leftarrow \pi(s, g)$. Apply action a
    steps ← steps + 1, $s \leftarrow$ current state
    if $\mathcal{D}(s, G) \leq \eta$ **then**
        break
    if $\mathcal{D}(s, g) \leq \eta$ **then**
        path ← path{1 : end} ▷ Remove the node at index 0, steps ← 0
    if *steps % ρ = 0* **then**
        $s \leftarrow$ current state
        path ← **SubgoalPlan**(s, G) ▷ Replan

Since the weight of each edge in $\mathcal{GR}$ is equal to the predicted distance $\mathcal{D}$ from one node to another (subsection IV-C), the shortest path over the graph satisfies the problem definition (equation 3). The agent traverses the planned path using the primitive action policy π, with the subgoal nodes used as the intermediate shorter-horizon goals, until it reaches the long-horizon goal G.

To detect if the agent has reached a subgoal, we use state localization as follows. A state $s \in \mathbb{S}$ is *localized* to a subgoal node g_i if $\mathcal{D}(s, g_i) \leq \eta$ or $\mathcal{D}(g_i, s) \leq \eta$. The hyper-parameter $\eta \in [0, \infty)$ is the maximum predicted distance from one state to another (state or subgoal) up to which they are considered close to each other. The value of η is smaller than the value of $MAXDIST$. The agent *re-plans* the path after a fixed number of steps if it has not reached a target subgoal node (determined by localization). We set the step-limit as a hyper-parameter ρ, which is equal to $MAXDIST$ - Q^+ when the default reward is -1 at each step.

If the agent does not reach a target subgoal node after ρ steps, it assigns the *current* state as the start state, removes the previous temporary edges, creates new temporary edges, and replans. If a path to G over the subgoal graph cannot be found, the agent simply follows the policy $\pi(a|s, G)$ using the long-horizon goal G as input instead of a subgoal. The time complexity of planning (using Dijkstra's algorithm) is $\Theta(|\mathcal{E}|$ + $|\mathbb{G}^{sub}|) \times log|\mathbb{G}^{sub}|)$.

Algorithm 3 Subgoal Graph Pruning

INPUT: Non-pruned graph $\mathcal{GR}$
OUTPUT: Pruned graph $\mathcal{GR}$
def *EdgeTraversal($\mathcal{E}$, s, steps)*:
    keep ← 1, $(g_i, g_y) \leftarrow \mathcal{E}$, $R^{\mathcal{E}}(g_i, g_y) \leftarrow 0$
    i ← 1
    while $i \leq \rho$ **do**
        $a \leftarrow \pi(s, g_y)$, Apply action a
        $R^{\mathcal{E}}(g_i, g_y) \leftarrow R^{\mathcal{E}}(g_i, g_y) + R(s, a, g_y)$
        steps ← steps + 1, i ← i + 1, $s_i \leftarrow$ current state
        if $\mathcal{D}(s, g_y) \leq \eta$ **then**
            break ▷ reached g_y
    $\mathcal{D}^{\mathcal{E}}(g_i, g_y) \leftarrow Q^+ - R^{\mathcal{E}}(g_i, g_y)$
    if $\mathcal{D}^{\mathcal{E}}(g_i, g_y) >$ *MAXDIST* **then**
        keep ← 0 ▷ to be pruned
    return keep, steps
Add the edges of $\mathcal{GR}$ to untested_edges_list and nodes to untested_nodes_list
Initialize MAX_PRUNING_EPISODES and MAX_STEPS
ep ← 0
while *ep < MAX_PRUNING_EPISODES* **do**
    Reset initial state
    seek_subgoal ← $\emptyset$, local_subgoal ← $\emptyset$
    if *untested_edges_list* = $\emptyset$ **then**
        break
    steps ← 0
    while *steps < MAX_STEPS* **do**
        $s \leftarrow$ current state
        if $\mathcal{D}(s, g_i) \leq \eta$ *for any* $g_i \in \mathcal{GR}$ **then**
            local_subgoal ← g_i
            if *local_subgoal is not in untested_nodes_list* **then**
                local_subgoal ← $\emptyset$
        if *local_subgoal* ≠ $\emptyset$ **then** ▷ Enter B-F mode
            $g_i \leftarrow$ local_subgoal, seek_subgoal ← $\emptyset$, forth_edges ← $\emptyset$
            foreach $g_y \in \mathcal{GR}.neighbours(g_i)$ **do**
                if $\mathcal{E}(g_i, g_y) \in$ *untested_edges_list* **then**
                    Add $\mathcal{E}(g_i, g_y)$ to forth_edges
            node_done ← 1
            foreach *fe* ∈ *forth_edges* **do**
                keep, steps ← **EdgeTraversal**(fe, s, steps)
                if *keep = 0* **then**
                    Remove fe from $\mathcal{GR}$ ▷ pruning
                Remove fe from untested_edges_list
                $s \leftarrow$ current state, Back edge $be \leftarrow \mathcal{E}(s, g_i)$
                keep, steps ← **EdgeTraversal**(be, s, steps)
                if *keep = 0* **then**
                    node_done ← 0
                    break ▷ did not reach back
            if *node_done = 1* **then**
                Remove g_i from untested_nodes_list
            local_subgoal ← $\emptyset$
        if *local_subgoal* = $\emptyset$ **then** ▷ Enter SEEK mode
            if *seek_subgoal* = $\emptyset$ **then**
                seek_subgoal ← random node from untested_nodes_list
            path ← **SubgoalPlan**(current state, seek_subgoal) ▷ Algorithm 2
            Step towards path1, steps ← steps + 1
    ep ← ep + 1

E. Subgoal Graph Pruning

In subsection IV-C, we introduced the hyper-parameter $MAXDIST$ which limits the formation of edges between distant subgoals. However, it is still possible that a few of the edges added to the subgoal graph $\mathcal{GR}$ are formed due to Q-value prediction errors, while the actual distance between two subgoal nodes is larger than $MAXDIST$.

Such edges become problematic in task domains that contain obstacles (such as the walls in Figure 1) because an erroneous edge might be formed between two subgoal nodes across an obstacle. To remove such erroneous edges, LSGVP is equipped with an autonomous pruning procedure which is applied before the testing phase. The pruning procedure uses the following assumption about state localization,

Assumption 2: The predicted distance $\mathcal{D}$ is accurate or correct for pairs of states or subgoals that are closer than the localization limit η ($< MAXDIST$).

The subgoal graph pruning procedure of LSGVP is described as follows. It is also summarized in Algorithm 3. At the beginning of the pruning procedure, the agent fills a list of *untested nodes and edges* with the nodes and edges of the graph $\mathcal{GR}$. It then toggles between the following two modes - *back-and-forth edge traversal (B-F mode) and traversal to an untested node (SEEK mode)*, until all the edges have been tested or a pruning time limit is reached. These modes are described as follows.

a) B-F mode.: If the current state is localized to an untested subgoal node g_i, the B-F mode is started. In this mode, the agent traverses to one of the neighbour nodes of g_i, say g_y if the edge $\mathcal{E}(g_i, g_y)$ is untested. This traversal is performed using the primitive action policy π (subsection IV-A). The traversal stops when the current state localizes to g_y or after ρ steps (step-limit defined in subsection IV-D).

The edge $\mathcal{E}(g_i, g_y)$ is then removed from the list of untested edges. The cumulative reward obtained on the traversal trajectory from g_i to g_y, denoted as $R^{\mathcal{E}}(g_i, g_y)$, is then used to calculate an edge distance $\mathcal{D}^{\mathcal{E}}(g_i, g_y) = Q^{+} - R^{\mathcal{E}}(g_i, g_y)$. If $\mathcal{D}^{\mathcal{E}}(g_i, g_y) >$ MAXDIST, the edge $\mathcal{E}(g_i, g_y)$ is considered erroneous and it is removed from the graph $\mathcal{GR}$ (*pruning*). This pruning using a *single* traversal along an edge is based on the **condition** that the primitive action policy is stationary after the training phase (subsection IV-A) and the UMDP is also stationary (subsection III-A1).

The agent then traverses back to g_i for at most ρ steps. Upon returning to g_i, the agent chooses another neighbour, say g_z, as the next traversal target if the edge $\mathcal{E}(g_i, g_z)$ is untested. The edge traversal and pruning process is repeated for the edge $\mathcal{E}(g_i, g_z)$, and so on. The B-F mode is terminated when all outgoing edges from g_i have been tested or the agent cannot return to g_i within ρ steps. g_i is removed from the list of untested nodes if all of its outgoing edges have been tested.

b) SEEK mode.: Upon termination of B-F mode for one subgoal node, or when the current state cannot be localized to any subgoal node, the agent enters the SEEK mode. In this mode, the agent traverses to a randomly chosen *untested* node by planning the shortest path from the current state to the chosen node (using the planning procedure described in subsection IV-D). Upon reaching an untested node, the B-F mode is resumed for that node.

The time required to complete the B-F mode for all the edges is $O(|\mathcal{E}|^2 \times \rho)$. Additional time is required during each SEEK mode traversal between consecutive execution of B-F mode for different nodes. That time is difficult to quantify because it depends on the current state and the randomly chosen *untested* node to which the agent traverses during the SEEK mode.

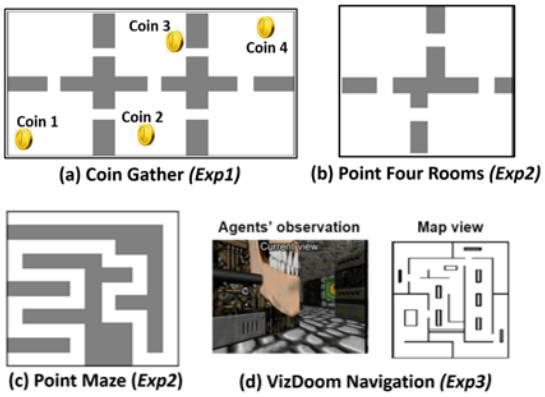


Fig. 2: Task domains used in different experiments.

		Training	Pruning	Testing
Exp1	DDPG	1800 x 300	-	100 x 300
	LSGVP and other methods	1500 x 20	300 x 300	100 x 300
Exp2	DDPG	1800 x 300	-	100 x 300
	SoRB and Map-planner	6000 x 20	-	100 x 300
	LSGVP and SGM	1500 x 20	300 x 300	100 x 300
Exp3	SPTM	30 x 50,000	-	96 x 5000
	LSGVP and SGM	10 x 50,000	200 x 5000	96 x 5000

TABLE I: Composition of different experiments. Each cell value is in terms of 'number of episodes $\times$ number of steps'. Subgoal discovery and graph construction occurs after Training stage.

V. Experiments

This section presents various experiments conducted to evaluate our method LSGVP on task domains requiring navigation to long-horizon goals. The composition of each trial of different experiments is shown in Table I. The task domains used in different experiments are shown in Figure 2. The hyper-parameters of LSGVP are set via a procedure described in *Appendix A* of the Supplementary document. Three types of performance metrics are used in the reported experiments, as follows,

Average Success Rate. Success rate in one trial of an experiment is equal to the *percentage* of testing episodes in which the agent successfully reaches the long-horizon goals. The average is taken across multiple trials.

Average Planning Time. Planning Time is equal to the time required to plan a single path from a start state to a long-horizon goal over the subgoal graph. The average is taken across multiple instances of planning within a testing episode, across multiple testing episodes in a trial, and across multiple trials.

Average Positive Cumulative Reward (APCR). Positive cumulative reward is equal to the total positive reward (including zero) gathered by the agent while traversing from the start state to the long-horizon goal in a testing episode of a trial. The

average is taken across multiple testing episodes in a trial and across multiple trials.

A. Exp1: LSGVP vs Other Subgoal Discovery Methods

LSGVP uses the *path value* measure for subgoal discovery (subsection IV-B, equation 4). In this subsection, we compare LSGVP with a few other *subgoal graph-based ATD* methods which use different subgoal discovery heuristics.

1) Task Domain:

Coin Gather (Figure 2): *Continuous state and action spaces. Navigation + Object Gathering.* This is our custom task domain in which an agent needs to traverse a two-dimensional plane to reach a goal location, while it can gather a Coin (intermediate positive reward) during the traversal. The agent is in the form of a *two*-dimensional point. The state space is *three*-dimensional and each state is denoted as $s = (x, y, hasCoin) \in [0, 55] \times [0, 80] \times (0, 1)$. The state represents the location of the agent and whether it has gathered a Coin or not. The action space is *two*-dimensional and each action is denoted as $a = (dx, dy) \in [-1, 1]^2$, which indicates an incremental change in the location of the agent. Random noise is added to the action with probability 0.1 to introduce stochasticity. There are *four* Coins fixed at different locations. Each Coin physically represents a 2×2 region in the plane. If the agent passes through this region, it gathers the corresponding Coin. Upon gathering a Coin, the agent receives a $+5$ reward. The agent can only gather a Coin if the value of $hasCoin$ is 0. The value of $hasCoin$ becomes 1 when the agent gathers any one of the Coins. Due to this feature, the agent does not loop around a Coin region (exploiting the positive reward).

Other than the sparse positive rewards, the default reward at each time step is -1. The reward upon reaching a goal state is 0. In each testing episode, the agent is provided with two goals, both with the same location features but different $hasCoin$ features. The agent can reach either of the goals for successful completion of the episode. This means that gathering a Coin is not necessary to reach a goal.

The hyper-parameters of LSGVP and other methods in this domain are as follows: ϵ = 5, Q^+ = 5, $MAXDIST$ = Q^+ + 11, η = Q^+ + 3, ρ = $MAXDIST$ - Q^+, $|\mathcal{B}|$ = 2000, $|\mathcal{SG}|$ = 2000.

2) Methods Compared: For a fair comparison, we use the same training procedure (subsection IV-A), graph construction procedure (subsection IV-C), and the distance function (equation 2) as those used in LSGVP for all methods discussed below. Moreover, our pruning procedure is added to all the methods (subsection IV-E). Hence, the only difference between LSGVP and other methods is in terms of the heuristic used for subgoal sampling/discovery. The compared methods are described as follows,

SoRB+Pruning: This method samples a predetermined number of subgoals with *uniform* probability. Here, SoRB stands for Search on the Replay Buffer [6], which is a state-of-the-art subgoal-graph based planning method. In the original SoRB, the primitive policy and Q-values are learnt in the same manner as described in subsection IV-A. Then, a predetermined number of subgoals are randomly sampled from the experience memory $\mathcal{B}$ with uniform probability. SoRB then constructs a graph connecting the sampled subgoals, in which the edge weights are the inverse of the Q-values (the same as the distance function used for LSGVP). However, the original SoRB does not use a pruning technique to reduce erroneous edges.

FPS+Pruning: This method samples a predetermined number of subgoals that widely cover the state space, using the *Farthest Point Sampling (FPS)* algorithm [8], [12]. In FPS, the first subgoal state is randomly sampled from $\mathcal{B}$. Then, a state in $\mathcal{B}$ that is farthest from the first subgoal is sampled as the next subgoal. The distance measure is the same as the function $\mathcal{D}$ defined in equation 2. Subsequently, the next sampled subgoal is farthest from the first two subgoals, the fourth sampled subgoal is farthest from the first three, and so on. The sampling stops after populating the subgoal set $\mathbb{G}^{sub}$ with the predetermined number of subgoals. The sparse subgoal graph is then constructed over these subgoals. This heuristic is used by Huang et al. [8] to learn the sparse subgoal graph for goal-reaching. However, their method does not include pruning.

Bottlenecks+Pruning: This method samples a predetermined number of subgoals lying at the *bottleneck* regions of the state space. Bottlenecks are those regions which become the convergence points for the paths connecting various pairs of start and goal states [27]. In the problem scope of this paper (subsection III-A1), we only use the *shortest paths* connecting various pairs of start and goal states to identify such bottlenecks. Firstly, we construct a dense graph over the states in the experience memory $\mathcal{B}$ using the procedure given in subsection IV-C (same distance function $\mathcal{D}$ and setting MAXDIST = 4). This is called the *base graph*. Then, a set of shortest paths are found between random pairs of start and goal states (nodes), over the base graph. Subsequently, we use the Betweenness Centrality (BC) measure [28] to discover the bottleneck subgoals. BC of a state is equal to the fraction of shortest paths passing through that state. A state with higher BC represents a bottleneck subgoal. We sort the states according to BC values and take the predetermined number of subgoals with higher BC.

SR+Pruning [29]: In this method, subgoals are discovered by first learning Successor Representations (SRs) of the states observed during exploration and then clustering the SRs to extract the cluster centers. Subgoals are the states with largest cosine similarity with the cluster centers. The SRs capture temporally close-by states, therefore, the generated clusters are spread across the state space while each cluster contains densely connected states. Thus, the discovered subgoals tend to optimize the state space coverage. However, there is no involvement of rewards or value-function in subgoal discovery.

Sparse Graphical Memory (SGM) [9]: This method is closer to LSGVP in terms of the value-based subgoal discovery. The basis of SGM is to create a sparse graph structure by dynamically merging similar nodes, where the nodes refer to a subset of the states observed by the agent. The similar nodes are determined using *two-way consistency*,

Methods ↓	APCR	Relative APCR %	Average Success Rate
Non-hierarchical DDPG	0.02 ± 0.03	0.004	17.2 ± 2.4
SoRB+Pruning	2.65 ± 1.22	58.1	90.9 ± 3.2
FPS+Pruning	2.80 ± 1.31	61.4	91.2 ± 2.9
Bottleneck+Pruning	2.47 ± 1.28	54.1	89.6 ± 4.6
SR+Pruning	2.82 ± 1.28	61.9	90.6 ± 2.7
SGM	**4.55** ± 0.22	100	91.2 ± 3.0
LSGVP	**4.56** ± 0.21	100	91.1 ± 3.1

TABLE II: LSGVP vs other subgoal discovery methods. APCR stands for Average Cumulative Positive Reward.

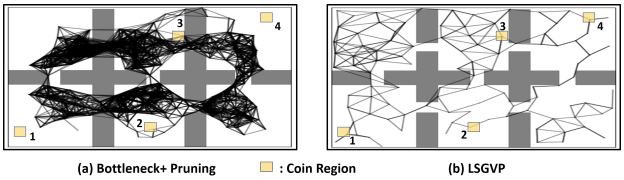


Fig. 3: Subgoal graphs learnt in one of the trials of Exp1. Only the subgoal states with $hasCoin$ = 0 are shown as the graph vertices in sub-figure (b) for clarity.

according to which two states are considered similar if they are both interchangeable as goals and interchangeable as starting states *according to the goal-conditioned value function*. Only the states which are dissimilar are kept as salient subgoals. The dissimilarity threshold is set using a hyper-parameter τ_a. This is comparable to the dissimilarity based subgoal selection in our method (subsection IV-B), however, we determine dissimilarity based on *path values* by directly considering nodes/states as subgoals rather than start or goal states. SGM also includes a pruning procedure based on random goal selection. For this experiment, we replace SGM's pruning procedure with our back-and-forth pruning (subsection IV-E). The dissimilarity threshold τ_a is set as $\tau_a = 5$ and $MAXDIST$ is same as that of LSGVP.

LSGVP discovers 197 ± 4 subgoals in different trials, SGM discovers 198 ± 3, and for other methods the cutoff is set as 200 subgoals. Apart from these subgoal graph-based (hierarchical) methods, we also compare the non-hierarchical Deep Deterministic Policy Gradient (DDPG) method [1] with LSGVP. In this method, a non-hierarchical agent uses the goal-conditioned primitive action policy $\pi(s, a|G)$ to reach a long-horizon goal without any subgoal. The policy and the Q-value function are implemented as DDPG actor and critic, respectively. This is the same as the implementation of the primitive action policy and Q-value function used by other methods, including LSGVP.

3) Results and Analysis: Ten experiment trials are conducted for each method. The composition of each trial is shown in Table I. In each training episode, the start state and the (shorter horizon) training goal are randomly sampled from anywhere in the state space. In each testing episode, the test goal is randomly sampled from anywhere in the state space but the start state is randomly sampled from one of the rooms not containing the goal, to ensure that the goal-reaching horizon is long. As mentioned in subsection V-A1, the test goal is duplicated into two goals with $hasCoin$ equal to 0 and 1, respectively. We run *each* testing episode twice, once for each test goal, and take the best result in terms of APCR (including zero reward).

Table II shows the results comparing the APCR obtained using different methods during the testing phase, as well as their success rates. The non-hierarchical agent performs the worst among all methods and often fails to reach the long-horizon goal because it gets stuck at the obstacles. This shows that subgoals are important for reaching long-horizon goals. Among the subgoal graph-based methods, it is observed that SoRB+Pruning, FPS+Pruning, Bottleneck+Pruning, and SR+Pruning agents take less positively rewarding paths to the test goals on average, in comparison to the LSGVP agent. These results are analyzed below.

In the Coin Gather task domain, an agent does not need to necessarily gather a Coin to reach a test goal. Therefore, various subgoal graph-based methods, including LSGVP, show similar performance in terms of the average success rate. However, the agent must include the subgoal states from the *Coin* regions (Figure 3) as the nodes of the subgoal graph to be able to plan a higher reward path to a test goal. LSGVP discriminates the utility of different states as subgoals based on their respective path values (subsection IV-B). Hence, it can identify that a state lying in a Coin region, including Coin 1 and Coin 4, has a higher path value compared to other states in the proximity. In this way, the LSGVP agent consistently includes a Coin state as one of the subgoals in the graph in various trials. The only method which is comparable to LSGVP in terms of value-based subgoal discovery is SGM, which discriminates different subgoal candidates (states) according to the value function. SGM performs similar to LSGVP in terms of APCR. This method is able to discover the subgoals lying in the higher reward regions (*Coin* regions), similar as LSGVP, due to the value-based *two way consistency* heuristic used for subgoal discovery.

SoRB+Pruning, FPS+Pruning, and SR+Pruning agents often fail to include the subgoal states from the *Coin* regions because their subgoal discovery heuristics do not discriminate between the subgoals lying on the higher reward paths and other subgoals. Therefore, these two agents gather about 40% less APCR compared to LSGVP. Intuitively, the *Bottleneck+Pruning* agent should include the subgoals from the Coin regions in the set of discovered bottlenecks. This is so because several *shortest* paths over the *base graph* must pass through the Coin states since the edge weights of that graph (distance function $\mathcal{D}$) are inversely proportional to the cumulative reward (equation 2). However, the *fraction* of the shortest paths passing through the Coin regions *1 and 4* at the two corners (Figure 3) is lower than the fraction of shortest paths passing through the central area. Therefore, most of the bottleneck subgoals lie in the central area, including the Coin regions *2 and 3* (Figure 3). This leads to a few cases in which the agent should pass through Coin 1 or Coin 4 to gather more rewards but it does not do so because the subgoal graph does not lead the agent through those Coin regions. Hence, the

APCR of Bottleneck+Pruning is also about 50% worse than that of LSGVP.

B. Exp2 and 3: Effect of Pruning

In this subsection, we compare LSGVP with state-art-of-the-art methods that also use subgoal graphs for planning but without automatic pruning or using different pruning heuristic. We conduct two experiments, one in two-dimensional maze navigation domains and the other in high-dimensional visual navigation domain, discussed as follows.

1) Exp2: Two-dimensional Navigation:

a) Task Domains: We use simple two-dimensional navigation domains for this experiment (Figure 2), taken from the SoRB source code[2], described as follows.

Point Four Rooms [6]: *Continuous state and action spaces. Navigation.* This is a simple *two*-dimensional navigation domain provided by Eysenbach et al. [6]. Each state in a *two*-dimensional space is denoted as $s = (x, y) \in [0, 55]^2$, representing the location of the agent. The action space is also continuous and each action is denoted as $a = (dx, dy) \in [-1, 1]^2$, which indicates an incremental change in the state. Random noise is added to the action with probability 0.1 to introduce stochasticity. The environment consists of four rooms separated by walls (*obstacles*). The agent needs to traverse across these rooms through connecting passages when seeking a goal state. The default reward is -1 at each time step and 0 only near the goal.

Point Maze [6]: Similar to Point Four Rooms except for the placement of the obstacles (walls), as shown in Figure 2.

The hyper-parameters of LSGVP in both domains are as follows: ϵ = 5, Q^+ = 0, $MAXDIST$ = 11, η = 3, ρ = $MAXDIST$, $|\mathcal{B}|$ = 2000, $|\mathcal{SG}|$ = 2000.

b) Methods Compared: The task domains used for this experiment do not contain intermediate positive rewards, unlike *Coin Gather* (subsection V-A). Hence, the subgoal sampling technique does not matter. The comparison is mainly in terms of subgoal graph *pruning*.

Search on the Replay Buffer (SoRB) [6]: SoRB [6] is previously discussed in subsection V-A. It has the same training (subsection IV-A) and graph construction (subsection IV-C) procedures as used in LSGVP. Moreover, both SoRB and LSGVP use the similar implementation of the goal-conditioned primitive action policy (DDPG actor [1]). The key differences are that SoRB *uniformly* samples a predetermined number of subgoals to construct the graph and does not prune the graph. The sparsity of the graph is controlled by the number of subgoals sampled.

Map-planner [8]: This method extracts salient subgoal states from an experience buffer by using Farthest Point Sampling (FPS) algorithm [12] and then constructs the subgoal graph using distances estimated using a learnt Universal Value Function (UVF) similar to the value function discussed in subsection III-A. There is no graph pruning involved.

Sparse Graphical Memory (SGM) [9]: SGM is described in subsection V-A2. It uses an automatic graph pruning heuristic called *cleanup*. A random goal is selected in each iteration of cleanup and a shortest path to the goal is planned over the subgoal graph. If the agent cannot traverse a particular edge on this path, it is treated as erroneous edge and removed. In contrast, LSGVP uses a more rigorous back-and-forth edge traversal to prune the erroneous edges (see example in Fig. 4). For this experiment, we set the dissimilarity threshold of SGM as $\tau_a = 5$.

We also compare the non-hierarchical Deep Deterministic Policy Gradient (DDPG) method [1] with LSGVP, in which the policy and the Q-value function are implemented as DDPG actor and critic, respectively. This is the same as the implementation of the primitive action policy and Q-value function used by LSGVP.

c) Results and Analysis: Ten experiment trials are conducted for each method. The composition of each trial is shown in Table I. The training time for SoRB and Map-planner are extended for fair comparison since LSGVP and SGM use extra episodes of experience for pruning. In each training episode for both task domains, the start state and the (shorter horizon) training goal are randomly sampled from anywhere in the state space. In each testing episode for *Point Four Rooms*, the test goal is randomly sampled from anywhere in the state space but the start state is randomly sampled from one of the rooms not containing the goal, to ensure that the goal-reaching horizon is long. Similarly, in each testing episode for *Point Maze*, the test goal is randomly sampled from one of the ends of the maze but the start state is randomly sampled from the other end, to ensure that the goal-reaching horizon is long.

Table III shows the performance results for non-hierarchical DDPG, LSGVP, SGM, and different versions of SoRB and Map-planner with different number of subgoals (nodes) in the graph. *Less number of subgoals (nodes) imply more sparsity of the subgoal graph*. Only the best result is shown for each method, corresponding to the best value of MAXDIST. It is observed that denser SoRB and Map-planner graphs (e.g., Fig. 5 (a) and (c)) lead to higher success rates but also require more planning time due to longer paths over the graphs. On the other hand, sparser SoRB and Map-planner graphs (e.g., Fig. 5 (b) and (d)) reduce the average planning time but also result in lower success rates. This is because the distance cutoff for edge formation, that is, $MAXDIST$ (subsection IV-C), needs to be increased when the subgoals (nodes) are sparsely distributed. This increases the chance of erroneous edge formation across obstacles due to the mis-predicted distances (equation 2). The SoRB and Map-planner agents do not prune the graph and, therefore, have lower success rates.

In contrast, the LSGVP and SGM agents achieve higher success rates while maintaining similar level of sparsity as SoRB (100 nodes) and Map-planner (100 nodes), by automatically pruning the erroneous edges in the sparse subgoal graph. However, the graph pruning procedure of SGM (called *cleanup*) only prunes the erroneous edges which are found when traversing to random goals. In contrast, LSGVP uses a more rigorous back-and-forth edge pruning procedure (subsection IV-E which covers more edges within same pruning iterations. Therefore, as shown in Fig. 5 (e), the SGM graph

[2]https://github.com/google-research/google-research/tree/master/sorb

	Point Four Rooms			Point Maze			VizDoom
Methods ↓	MAXDIST*	Average Success Rate	Average Planning Time (in sec.)	MAXDIST*	Average Success Rate	Average Planning Time (in sec.)	Average Success Rate
Non-hierarchical DDPG	-	12.4 ± 3.3	-	-	5.6 ± 2.4	-	
SoRB (1000 nodes)	7	**97.5** ± 4.6	9.95 ± 0.05	7	**68.4** ± 6.2	10.1 ± 0.11	-
SoRB (500 nodes)	9	76.3 ± 4.6	5.02 ± 0.07	9	32.5 ± 7.6	5.21 ± 0.06	-
SoRB (100 nodes)	12	52.3 ± 4.7	**1.01** ± 0.08	12	20.7 ± 4.3	**1.10** ± 0.04	-
Map-planner (1000 nodes)	7	**96.3** ± 4.2	9.76 ± 0.04	7	**68.8** ± 5.9	9.8 ± 0.14	-
Map-planner (100 nodes)	12	49.4 ± 4.2	**1.02** ± 0.06	12	18.3 ± 4.3	**1.10** ± 0.06	-
LSGVP (Ours)	11	**96.8** ± 3.6	**1.03** ± 0.04	12	**69.2** ± 5.5	**1.11** ± 0.03	**74.5** ± 9.6
SGM	11	94.7 ± 4.0	**1.08** ± 0.04	12	64.3 ± 4.7	1.17 ± 0.05	68.7 ± 10.2
SPTM	-	-	-	-	-	-	52.4 ± 12.3

TABLE III: Exp2 and Exp3 results, taken as average across ten trials. *MAXDIST is a hyperparameter, not a performance metric.

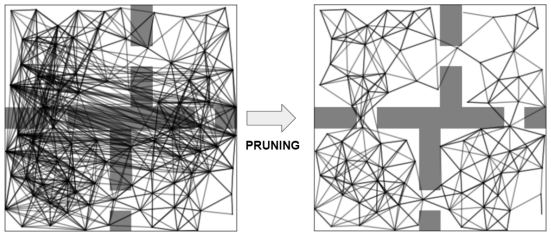
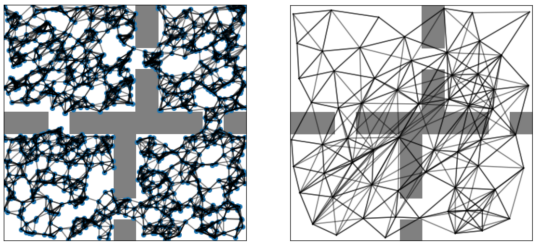
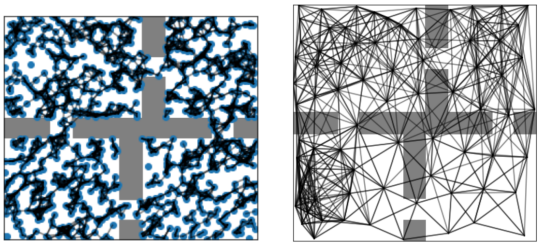


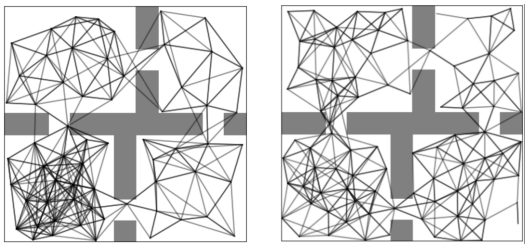
Fig. 4: Example of LSGVP graph pruning.



(a) SoRB (1000 nodes) (b) SoRB (100 nodes)



(c) Map-planner (1000 nodes) (d) Map-planner (100 nodes)



(e) SGM Pruned. (f) LSGVP Pruned.

Fig. 5: Graphs learnt by different methods.

still has a few erroneous edges which bring down the success rate compared to LSGVP.

2) Exp3: High-dimensional Navigation: In this experiment, we compare LSGVP with other subgoal graph-based planning methods in a high-dimensional visual navigation domain VizDoom [10]. The methods compared with LSGVP are discussed below.

Sparse Graphical Memory (SGM) [9], described in subsections V-A2 and V-B1b.

Semi Parametric Topological Memory (SPTM) [7]: This method is designed for long-horizon goal-reaching in environments with high-dimensional state spaces, where each state is a *first person view* image observed by the agent. SPTM uses a *pre-trained* Retrieval network (denoted as RtN in this paper) to estimate the probability that two states are *at most* 20 time steps apart. This means that for a pair of states s_i and s_k, $RtN(s_i, s_k)$ = 1 implies that the two states are at most 20 steps apart and a lower value of RtN implies that the two states are farther than 20 steps. SPTM samples subgoals at a uniform interval from the exploration trajectories.

The original SPTM [7] uses exploration trajectories generated by human demonstrators, whereas in this paper we use random exploration for all methods. After uniform sampling, consecutive subgoals and the subgoals with RtN value above a threshold are connected via edges to form a subgoal graph. The SPTM agent plans a path over this subgoal graph to reach a long-horizon goal (given as an image). The traversal from one subgoal/state to another subgoal/state is performed using a pre-trained policy.

For this experiment, LSGVP and SGM use the same pre-trained Retrieval network and policy as in SPTM[3]. In congruence with the RtN probability estimation, we redefine the distance function used in LSGVP as $\mathcal{D}(s_i, s_k)$ = 100 - 80$\times RtN(s_i, s_k)$. Note that if we assume that the agent receives a -1 reward at each time step, the cumulative reward obtained when traversing from s_i to s_k is higher if the states are closer (that is, higher RtN). Thus, the modified distance is still inversely proportional to the cumulative reward. Apart from this, the two main differences between SPTM and LSGVP are that the subgoals are uniformly sampled in SPTM whereas LSGVP discovers them using the path value-based heuristic, and SPTM does not use automatic pruning. The

[3]https://github.com/nsavinov/SPTM

VizDoom task domain used for this experiment (discussed below) does not include intermediate positive rewards, unlike the Coin Gather domain. Therefore, the subgoal sampling technique does not matter and the performance comparison is mainly based on the effect of graph pruning.

a) Task Domain: **VizDoom Navigation** [7] (Figure 2): *High-dimensional continuous state space. Visual (image) states. Discrete action space. Navigation.* VizDoom is a first-person game [10] with navigation and enemy-shooting tasks. We focus on the navigation task for goal-reaching. The agent is a player which observes its first-person view as a 160×120 visual state (image). In this case, different dimensions corresponding to the image pixel values. The agent can take one of the following *seven* actions: Do nothing, move forward, backward, left, right, turn left, and turn right. It needs to navigate through a maze environment while seeking a goal (specified as an image). Considering the modified distance function $\mathcal{D}(s_i, s_k)$ = 100 - 80$\times RtN(s_i, s_k)$, the LSGVP hyper-parameter values for this domain are set as follows: ϵ = 60, η = 21 (RtN = 0.99), MAXDIST = 32 (RtN = 0.85), ρ = MAXDIST, $|\mathcal{B}|$ = 10,000, and $|\mathcal{SG}|$ = 10,000. For SGM, $\tau_a = 20$ and $MAXDIST = 32$.

b) Results and Analysis: Ten experiment trials are conducted for each method. The composition of each trial is shown in Table I. There is no training phase for this experiment since the retrieval network and policy network are pre-trained. During the testing phase, four predefined test goals are used for each method. Each testing trial consists of 96 episodes, with 24 episodes per goal. The agent is successful if it reaches the goal in an episode within 5000 steps. The testing phase in each trial is preceded by *exploration*, subgoal graph construction, and pruning (only in the case of LSGVP). The trajectories gathered during exploration are subsampled to create experience memory of size $|\mathcal{B}|$ = 10,000 for both SPTM, LSGVP, and SGM. Then, LSGVP discovers around 2500 subgoals with ϵ = 60 as the *threshold of dissimilarity* (similarly, SGM discovers around 2500 subgoals). In contrast, SPTM extracts 2,500 subgoals via uniform interval sampling.

Since all the methods use a similar number of subgoals, the sizes of their subgoal graphs are also similar. Hence, we do not compare the them based on average planning time but only on the basis of goal-reaching success rate, which is affected by pruning. The results are reported in Table III. In this case, the success rate is equal to the fraction of episodes in which the agent reaches the goal, out of the 96 testing episodes per trial. The average is taken across ten trials.

All three agents learn graphs with several erroneous edges due to the noisy exploration data used for graph construction. Since the SPTM agent does not prune the learnt graph, it plans several infeasible plans for subgoal-to-subgoal traversal across obstacles due to the erroneous edges. This results in its lower success rate compared to the LSGVP and SGM agents. As discussed in subsection V-B1c. LSGVP performs a more thorough back-and-forth edge traversal for graph pruning which leads to cleaner graph compared to SGM which does pruning by traversing to random goals. Thus, SGM performs worse than LSGVP in this domain as well.

Despite better performance of LSGVP, the noisy exploration data becomes a limiting factor on the quality of the learnt graphs and the maximum success rate achieved. Better exploration strategies will be investigated in the future.

VI. Conclusion

This paper presents a method for long-horizon goal-reaching via subgoal graph-based planning, called Learning Subgoal Graph using Value-based Subgoal Discovery and Automatic Pruning (LSGVP). LSGVP addresses two key challenges concerning subgoal graph-based planning, specifically, discriminating subgoals lying on higher cumulative reward paths from other candidate subgoals during discovery and making the subgoal graph robust against erroneously predicted edges between subgoals.

To address the former issue, LSGVP uses a novel subgoal discovery heuristic that finds the salient subgoals based on their *path values* (subsection IV-B). To address the latter issue, LSGVP uses an automatic graph pruning procedure involving back-and-forth edge traversals across the subgoal graph (subsection IV-E). Despite these benefits of LSGVP, there are some *key limitations* that must be acknowledged, as follows:

- LSGVP is not suitable for unbounded or infinite state spaces which need to be continuously explored, or for non-stationary environments because it requires subgoal graph construction to be completed before the graph can be used for goal-reaching via planning.
- LSGVP requires a default reward of -1 at each action step. This essentially converts the goal-reaching problem into a shortest-path learning/finding problem by default. The application of LSGVP is limited to such problems.
- Despite the graph pruning procedure, LSGVP is still susceptible to erroneous connections in the subgoal graph due to sub-optimally learnt distance function (which depends on the value function). More robust distance functions need to be investigated.

Future Work: There are two key directions in which this work can be extended. First, the integration of higher-level planning with Hierarchical Reinforcement Learning (HRL) [4] is important. Building a robust model for planning requires extensive exploration. Integrating planning with HRL will enable an agent to improve the task-execution policy during exploration while simultaneously building the higher-level model. This method will also be applicable to very large state spaces, since the HRL policy can be learned even when the planning model is incomplete.

Secondly, constructing subgoal graphs or other form of planning models for visual navigation problems is challenging and computationally expensive, due to the high-dimensional states/observations (as images) and usually large state spaces. A promising approach to scale robust model-construction to high-dimensional visual spaces is *self-supervised learning of correlations or similarities* among visual states [30]. Such a self-supervised pre-trained model can complement value-based subgoal sampling/discovery and lead to a model-construction

approach that is scalable, has lower computational cost, and is generalizable across various large visual spaces.

ACKNOWLEDGEMENT

the Jubilee Technology Fellowship awarded to Ah-Hwee Tan by Singapore Management University.

REFERENCES

[1] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4*, 2016.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[3] S. Pateria, B. Subagdja, A.-h. Tan, and C. Quek, "Hierarchical reinforcement learning: A comprehensive survey," *ACM Comput. Surv.*, vol. 54, no. 5, jun 2021. [Online]. Available: https://doi.org/10.1145/3453160

[4] A. Levy, G. D. Konidaris, R. P. Jr., and K. Saenko, "Learning multi-level hierarchies with hindsight," in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9*, 2019.

[5] N. Dilokthanakul, C. Kaplanis, N. Pawlowski, and M. Shanahan, "Feature control as intrinsic motivation for hierarchical reinforcement learning," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 11, pp. 3409–3418, 2019.

[6] B. Eysenbach, R. R. Salakhutdinov, and S. Levine, "Search on the replay buffer: Bridging planning and reinforcement learning," in *Advances in Neural Information Processing Systems*, vol. 32, 2019, pp. 15 246–15 257.

[7] N. Savinov, A. Dosovitskiy, and V. Koltun, "Semi-parametric topological memory for navigation," in *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3*, 2018.

[8] Z. Huang, F. Liu, and H. Su, "Mapping state space using landmarks for universal goal reaching," in *Advances in Neural Information Processing Systems*, 2019, pp. 1942–1952.

[9] S. Emmons, A. Jain, M. Laskin, T. Kurutach, P. Abbeel, and D. Pathak, "Sparse graphical memory for robust planning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 5251–5262, 2020.

[10] M. Kempka, M. Wydmuch, G. Runc, J. Toczek, and W. Jaśkowski, "Vizdoom: A doom-based ai research platform for visual reinforcement learning," in *2016 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2016, pp. 1–8.

[11] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 5026–5033.

[12] C. Moenning and N. A. Dodgson, "Fast marching farthest point sampling," University of Cambridge, Computer Laboratory, Tech. Rep. UCAM-CL-TR-562, Apr. 2003.

[13] Y. Hu, B. Subagdja, A.-H. Tan, and Q. Yin, "Vision-based topological mapping and navigation with self-organizing neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–13, 2021.

[14] A. Tan, B. Subagdja, D. Wang, and L. Meng, "Self-organizing neural networks for universal learning and multimodal memory encoding," *Neural Networks*, vol. 120, pp. 58–73, 2019.

[15] J. F. Henriques and A. Vedaldi, "Mapnet: An allocentric spatial memory for mapping environments," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 8476–8484.

[16] D. S. Chaplot, D. Gandhi, S. Gupta, A. Gupta, and R. Salakhutdinov, "Learning to explore using active neural slam," in *International Conference on Learning Representations (ICLR)*, 2020.

[17] D. S. Chaplot, R. Salakhutdinov, A. Gupta, and S. Gupta, "Neural topological SLAM for visual navigation," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*. Computer Vision Foundation / IEEE, 2020, pp. 12 872–12 881. [Online]. Available: https://openaccess.thecvf.com/content_CVPR_2020/html/Chaplot_Neural_Topological_SLAM_for_Visual_Navigation_CVPR_2020_paper.html

[18] Y. Lv, N. Xie, Y. Shi, Z. Wang, and H. T. Shen, "Improving target-driven visual navigation with attention on 3d spatial relationships," *CoRR*, vol. abs/2005.02153, 2020. [Online]. Available: https://arxiv.org/abs/2005.02153

[19] S. Nair and C. Finn, "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.

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

[21] K. Pertsch, O. Rybkin, J. Yang, S. Zhou, K. Derpanis, K. Daniilidis, J. Lim, and A. Jaegle, "Keyframing the future: Keyframe discovery for visual prediction and planning," in *Proceedings of the 2nd Conference on Learning for Dynamics and Control*, ser. Proceedings of Machine Learning Research, vol. 120. PMLR, 10–11 Jun 2020, pp. 969–979.

[22] T. Schaul, D. Horgan, K. Gregor, and D. Silver, "Universal value function approximators," in *Proceedings of the 32nd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 37. Lille, France: PMLR, 07–09 Jul 2015, pp. 1312–1320.

[23] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking deep reinforcement learning for continuous control," in *Proceedings of Machine Learning Research*, vol. 48. New York, New York, USA: PMLR, 20–22 Jun 2016, pp. 1329–1338.

[24] A. S. Vezhnevets, S. Osindero, T. Schaul, N. Heess, M. Jaderberg, D. Silver, and K. Kavukcuoglu, "Feudal networks for hierarchical reinforcement learning," in *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ser. ICML'17, 2017, p. 3540–3549.

[25] O. Nachum, H. Tang, X. Lu, S. Gu, H. Lee, and S. Levine, "Why does hierarchy (sometimes) work so well in reinforcement learning?" *CoRR*, vol. abs/1909.10618, 2019.

[26] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. Pieter Abbeel, and W. Zaremba, "Hindsight experience replay," in *Advances in Neural Information Processing Systems*, vol. 30. Curran Associates, Inc., 2017, pp. 5048–5058.

[27] P.-L. Bacon, *On the Bottleneck Concept for Options Discovery: Theoretical Underpinnings and Extension in Continuous State Spaces*, ser. Master's theses, School of Computer Science. McGill University Libraries, 2014.

[28] O. Şimşek and A. G. Barto, "Skill characterization based on betweenness," in *Proceedings of the 21st International Conference on Neural Information Processing Systems*, ser. NIPS'08. Red Hook, NY, USA: Curran Associates Inc., 2008, p. 1497–1504.

[29] R. Ramesh, M. Tomar, and B. Ravindran, "Successor options: An option discovery framework for reinforcement learning," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, S. Kraus, Ed. ijcai.org, 2019, pp. 3304–3310. [Online]. Available: https://doi.org/10.24963/ijcai.2019/458

[30] D. Yuan, X. Chang, P.-Y. Huang, Q. Liu, and Z. He, "Self-supervised deep correlation tracking," *Trans. Img. Proc.*, vol. 30, p. 976–985, jan 2021. [Online]. Available: https://doi.org/10.1109/TIP.2020.3037518