DEPARTMENT OF INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Bachelor's Thesis in Engineering Science

Graph-Based Hindsight Goal Generation for Robotic Object Manipulation with Sparse-Reward Deep Reinforcement Learning

Matthias Brucker

DEPARTMENT OF INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Bachelor's Thesis in Engineering Science

Graph-Based Hindsight Goal Generation for Robotic Object Manipulation with Sparse-Reward Deep Reinforcement Learning

Graph-Based Hindsight Goal Generation: Ein Deep Reinforcement Learning - Algorithmus zur Objekt-Manipulation mit einem Roboter-Greifarm

Author:	Matthias Brucker
Supervisor:	Prof. Dr.-Ing. habil. Alois Knoll
Advisor:	Dr. rer. nat. Zhenshan Bing
Submission Date:	April 6th, 2020

I confirm that this bachelor's thesis in engineering science is my own work and I have documented all sources and material used.

Munich, April 6th, 2020 Matthias Brucker

Abstract

Recent deep reinforcement learning (RL) algorithms such as Hindsight Experience Replay (HER) and Hindsight Goal Generation (HGG) have been successfully applied to challenging robotic object manipulation tasks, where an object shall be moved to a certain target goal position by a robotic gripper arm. Both HER and HGG excel in multi-goal settings with sparse rewards, where the agent, similar to many real-world problems, only receives a non-negative reward when it has reached the desired target goal. While HER accelerates the learning process through hindsight replays of past experience with a heuristic choice of previously achieved goals, HGG additionally guides the exploration process by providing the agent with an implicit curriculum of intermediate hindsight goals. For efficient guided exploration, hindsight goals are carefully chosen so that they are both easy to achieve in the short term and promising to lead towards target goals in the long term. In contrast to HER, HGG can successfully solve challenging tasks in which target goals are far away from the object's initial position. However, HGG is not applicable to object manipulation tasks in environments with obstacles, since the euclidean norm used for hindsight goal generation is not an accurate distance measure in such environments.

In this thesis, we introduce Graph-Based Hindsight Goal Generation (G-HGG), an algorithm extending HGG to environments with obstacles by replacing the euclidean norm with a graph-based distance measure. Hindsight goals are selected based on shortest distances in an obstacle-avoiding graph that is created as a discrete representation of the environment. G-HGG is evaluated on four challenging object manipulation tasks in environments with obstacles, where significant improvement in both sample efficiency and overall success rate is demonstrated over HGG and HER.

Contents

Abstract iii

List of Figures vi

Abbreviations vii

1. Introduction 1

2. Background 3
- 2.1. Deep Reinforcement Learning (RL) . . . 3
- 2.2. Deep Deterministic Policy Gradients (DDPG) . . . 4
- 2.3. Universal Value Function Approximators (UVFA) . . . 5
- 2.4. Hindsight Experience Replay (HER) . . . 6
- 2.5. Energy-Based Prioritization (EBP) . . . 8
- 2.6. Hindsight Goal Generation (HGG) . . . 10

3. Methodology 15
- 3.1. Problem Formulation . . . 15
- 3.2. Graph-Based Hindsight Goal Generation (G-HGG) . . . 19
  - 3.2.1. Graph Construction . . . 19
  - 3.2.2. Shortest Distance Computation . . . 23
  - 3.2.3. G-HGG: Algorithm . . . 26
- 3.3. Stop Condition . . . 28

4. Experiments 30
- 4.1. Environments . . . 30
  - 4.1.1. FetchPushLabyrinth . . . 31
  - 4.1.2. FetchPickObstacle . . . 32
  - 4.1.3. FetchPickNoObstacle . . . 33
  - 4.1.4. FetchPickAndThrow . . . 34
- 4.2. Results . . . 35

5. Discussion **39**
5.1. Applicability of G-HGG 39
5.2. Advantages of G-HGG 40
5.3. Drawbacks of G-HGG 41

6. Related Research **45**
6.1. Prioritized Experience Replay 45
6.2. Demonstrations for Improved Exploration 45
6.3. Curriculum Learning 46
6.4. Distance Measure for Environments with Obstacles 48

7. Conclusion & Outlook **50**

A. Experiment Settings **53**
A.1. Environment Settings 53
A.2. Evaluation Details 53
A.3. Overall Computation Time 54

B. Implementation Details **55**
B.1. Hardware and Software 55
B.2. Hyper-Parameters 55
B.3. Details on Data Processing 56

C. Ablation Study on G-HGG Parameters **57**
C.1. Stop Condition Parameter 57
C.2. Number of Vertices 58

Bibliography **60**

List of Figures

3.1. Euclidean distance vs. alternative distance 17
3.2. Vertices in demo obstacle environment 21
3.3. Outgoing edges from one vertex 22
3.4. Graph in demo obstacle environment 23
3.5. Existing graph-based distances in demo obstacle environment 25
3.6. Non-existing graph-based distances in demo obstacle environment . . . 25

4.1. FetchPushLabyrinth environment 32
4.2. FetchPickObstacle environment 33
4.3. FetchPickNoObstacle environment 34
4.4. FetchPickAndThrow environment 35
4.5. Test success rate of G-HGG compared to HGG and HER in different environments 36
4.6. Hindsight goals in FetchPushLabyrinth environment 37

5.1. Computation time per iteration of G-HGG compared to HGG and HER 43

C.1. Ablation study on the stop condition parameter 58
C.2. Ablation study on the number of vertices 59

Abbreviations

DDPG	Deep Deterministic Policy Gradients.
EBP	Energy-Based Prioritization.
G-HGG	Graph-Based Hindsight Goal Generation.
HER	Hindsight Experience Replay.
HGG	Hindsight Goal Generation.
MDP	Markov Decision Process.
RL	Deep Reinforcement Learning.
UVFA	Universal Value Function Approximators.

1. Introduction

In recent years, deep reinforcement learning (RL) has seen significant progress, with its concepts being successfully applied to decision-making problems in robotics, including, but not limited to, helicopter control (Ng et al. 2006), hitting a baseball (Peters and Schaal 2008), door opening (Chebotar et al. 2017), screwing a cap onto a bottle (Levine et al. 2016), and object manipulation (Andrychowicz et al. 2017). To train a meaningful policy for such tasks, neural networks are often used as function approximators for learning a value function to optimize a long-term expected return. Estimation of the expected return is based on a reward function, which must be thoroughly shaped for policy optimization and reflect the task under consideration at the same time. In most real-world applications, where a concrete representation of admissible behavior is unknown, time-consuming, and challenging, reward function engineering limits the applicability of RL.

Consequently, algorithms which support learning from sparse rewards, e.g. a binary signal indicating successful task completion, are a big advance since sparse rewards are easy to derive from the task definition without further engineering. Andrychowicz et al. (2017) have provided an algorithm called Hindsight Experience Replay (HER), which improves the success of off-policy deep Q-learning (Watkins 1989, Watkins and Dayan 1992) in multi-goal RL problems with sparse rewards. The idea behind HER is to first learn with intermediate goals that are easy to achieve and then to continue with more difficult goals. More concretely, HER constructs hindsight goals from previously achieved states, replays known trajectories with these hindsight goals, and trains the goal-dependent value function based on the results. While HER has proven to work very well in object manipulation environments where target goals are distributed uniformly around the initial position of the manipulable object (Plappert, Andrychowicz, et al. 2018), it fails to solve tasks where target goals are generally far away from the initial object position. Due to random exploration and the heuristic choice of hindsight goals from achieved states, hindsight goals keep being distributed around the initial object position, far away from the target goals, which will never be reached since no meaningful reward signal is obtained.

Hindsight Goal Generation (HGG) (Ren et al. 2019) tackles this problem by using intermediate hindsight goals as an implicit curriculum to guide exploration towards target goals. For efficient exploration, hindsight goals should be 1) easy to reach and 2) challenging enough to help the function approximators learn how to achieve target goals, eventually. In HGG, the choice of hindsight goals is based on two criteria: the current value function (as much knowledge as possible about how to reach the hindsight goals) and the Wasserstein distance between the target goal distribution and the distribution of potential hindsight goals (goals as close as possible to the target goal distribution). The resulting Wasserstein-Barycenter problem is discretely solved using the euclidean norm as a distances measure between two goals sampled from the potential hindsight goal distribution and the target goal distribution. While HGG demonstrates higher sample efficiency than HER in environments where the euclidean norm is applicable, it fails in environments with obstacles, where the shortest obstacle-avoiding distance between two goals cannot be computed with the euclidean norm.

In this thesis, we introduce Graph-Based Hindsight Goal Generation (G-HGG), an extension of HGG designed for sparse-reward RL in robotic object manipulation environments with obstacles. In order to make G-HGG applicable to these environments, the euclidean norm is replaced by a pre-computable graph-based distance measure: the environment's goal space is represented by a graph consisting of discrete goals as vertices and obstacle-avoiding connections between the vertices as edges. Distances between two goals are then approximated by the shortest paths on the graph between the two vertices that are closest to the two goals. Note that our graph-based shortest distances are inexact approximations of real euclidean shortest distance: since computing exact euclidean shortest distances in 3-D space is an NP-hard problem that is computationally expensive to solve, using such distances in HGG, where $5 \cdot 10^5$ distance must be calculated in each iteration, is not feasible. Despite their inexactness, our approximate and pre-computable graph-based shortest distances lead to great training results within reasonable computation time, as our experiments show.

We evaluate G-HGG on four challenging object manipulation tasks in simulated environments, most of them including obstacles. With the graph-based distance measure, a significant improvement in both sample efficiency and overall success rate is demonstrated over HGG and HER. We show that G-HGG can solve robotic object manipulation tasks in environments with obstacles, where HGG fails. After discussing applicability, advantages, and limitations of our algorithm, we provide future research proposals ranging from strategies to improve G-HGG performance to deploying G-HGG to a physical robot.

2. Background

2.1. Deep Reinforcement Learning (RL)

(Deep) Reinforcement learning (RL) is, together with supervised and unsupervised learning, one major class of machine learning. In RL, an agent interacts with its environment in discrete time steps and maximizes its expected cumulative reward.The environment is modeled as a Markov Decision Process (MDP) with a state space $\mathcal{S}$, an action space $\mathcal{A}$, a (potentially stochastic) transition function $p : S \times \mathcal{A} \to \mathcal{S}$, and a reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. Furthermore, an MDP contains an initial state distribution $\mathcal{S}_0 : \mathcal{S} \to \mathbb{R}$ and a discount factor $\gamma \in \mathbb{R}$. A RL agent's behavior is defined by a deterministic policy $\pi : \mathcal{S} \to \mathcal{A}$, which maps states to actions (potentially using a probability distribution).

Each iteration of a basic RL training follows the process. In the beginning, an initial state s_0 is sampled from $\mathcal{S}_0$. At every time step t, the agent chooses an action based on its current state and the underlying policy: $a_t = \pi(s_t)$. Based on the current state and the selected action, the agent receives a reward $r_t = r(s_t, a_t)$ and the environment's next state is obtained from the transition function $s_{t+1} = p(s_t, a_t)$.

The state-value function of an MDP is defined as the expected cumulative and discounted reward starting from state s, and then following policy π:

$$V^{\pi}(s) = \mathbb{E}_{s_0=s,\, a_t\sim\pi(s_t),\, s_{t+1}\sim p(s_t,a_t),\, a_{t+1}\sim\pi(s_{t+1})}\left[\sum_{t=0}^{\infty}\gamma^t r(s_t, a_t)\right] . \tag{2.1}$$

In a similar way, the action-value function or Q-function is defined as the expected cumulative and discounted reward starting from state s, taking action a, and then following policy π:

$$Q^{\pi}(s,a) = \mathbb{E}_{s_0=s,\, a_0=a,\, s_{t+1}\sim p(s_t,a_t),\, a_{t+1}\sim\pi(s_{t+1})}\left[\sum_{t=0}^{\infty}\gamma^t r(s_t, a_t)\right] . \tag{2.2}$$

An optimal policy π^* is defined in a way such that $Q^{\pi^*}(s,a) \geq Q^{\pi}(s,a)$ and $V^{\pi^*}(s) \geq V^{\pi}(s)$ for every $s \in \mathcal{S}$, $a \in \mathcal{A}$, and any other policy π.

All policies related to one RL training have the same optimal state-value and action-value functions, denoted as V^* and Q^*, respectively. It can be shown that these value functions satisfy their respective formulations of the Bellman Optimality Equation (2.3) (2.4).

$$V^*(s) = \mathbb{E}_{s' \sim p(s,a)} \left[\max_{a \in \mathcal{A}} r(s,a) + \gamma V^*(s') \right] \tag{2.3}$$

$$Q^*(s,a) = \mathbb{E}_{s' \sim p(s,a)} \left[r(s,a) + \gamma \max_{a' \in \mathcal{A}} Q^*(s',a') \right] \tag{2.4}$$

The goal of a RL agent is to find a policy π as close as possible to π^*, maximizing the expected cumulative reward.

2.2. Deep Deterministic Policy Gradients (DDPG)

Deep Deterministic Policy Gradients (DDPG) (Lillicrap et al. 2016) is a model-free, off-policy, actor-critic RL algorithm for continuous action spaces. DDPG uses two neural networks as function approximators, one as the actor, one as the critic. The actor $\pi(s \mid \theta^\pi)$ represents the current policy by deterministically mapping states to specific actions, while the critic $Q(s,a \mid \theta^Q)$ approximates the actor's action-value function Q^π. θ^π and θ^Q denote the weights of the corresponding neural networks.

In the beginning of DDPG, the actor network $\pi(s \mid \theta^\pi)$ and the critic network $Q(s,a \mid \theta^Q)$ are initialized with weights θ^π and θ^Q. So-called target networks $\pi'(s \mid \theta^{\pi'})$ and $Q'(s,a \mid \theta^{Q'})$ are initialized as copies of π and Q with $\theta^{\pi'} \leftarrow \theta^\pi$ and $\theta^{Q'} \leftarrow \theta^Q$.

In each iteration of DDPG, the following steps are performed:

- An initial state s_0 is sampled from $\mathcal{S}_0$.
- In each time step t, an action $a_t = \pi(s_t) + \mathcal{N}_t$ is selected according to the current policy π and some exploration noise $\mathcal{N}_t$, which could be sampled from e.g. $\mathcal{N}(0,1)$. After executing the action a_t, the agent obtains a reward $r_t = r(s_t, a_t)$ and observes a new state $s_{t+1} \sim p(s_t, a_t)$. Since DDPG is an off-policy RL algorithm, this so-called transition (s_t, a_t, r_t, s_{t+1}) is now stored in a replay buffer R, together with transitions encountered in previous time steps and episodes.
- Now, a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) is sampled from R. The

critic's weights θ^Q are updated by minimizing the loss

$$L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i \mid \theta^Q))^2 , \tag{2.5}$$

where

$$y_i = r_i + \gamma Q'(s_{i+1}, \pi'(s_{i+1} \mid \theta^{\pi'}) \mid \theta^{Q'}) . \tag{2.6}$$

Next, the actor's weights θ^π are updated using the sampled policy gradient

$$\nabla_{\theta^\pi} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a \mid \theta^Q) \mid_{s=s_i, a=\pi(s_i)} \nabla_{\theta^\pi} \pi(s \mid \theta^\pi) \mid_{s_i} . \tag{2.7}$$

- Finally, the critic and actor target networks are updated according to (2.8) and (2.9), respectively, with $\tau \in \mathbb{R}$, $\tau \ll 1$.

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'} \tag{2.8}$$

$$\theta^{\pi'} \leftarrow \tau\theta^\pi + (1-\tau)\theta^{\pi'} \tag{2.9}$$

A more detailed description of the DDPG algorithm is provided by Lillicrap et al. 2016.

2.3. Universal Value Function Approximators (UVFA)

Universal Value Function Approximators (UVFA) (Schaul, Horgan, et al. 2015) extend the idea of value function approximators (as used in DDPG) to both states s and goals g. UVFA is therefore particularly relevant in setups where more than one goal shall be achieved. The overall goal space $\mathcal{G}$ is defined as the space of all possible goals. In such a scenario, every goal $g \in \mathcal{G}$ can be linked to some reward function $r_g : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$. In the beginning of an episode of UVFA, a goal g is sampled from a target goal distribution $\mathcal{G}_T : \mathcal{G} \rightarrow \mathbb{R}$ in addition to an initial state $s_0 \sim \mathcal{S}_0$. This results in an initial state - goal pair (s_0, g), in which the goal does not change throughout the episode. Since the agent's behavior should depend not only on the given state, but also on the goal, the policy is redefined as $\pi : \mathcal{S} \times \mathcal{G} \rightarrow \mathcal{A}$. For convenience, we use $\|$ as a symbol for concatenation. Since goals can be interpreted as an extension of the state space, s_t can simply be replaced by $s_t \| g$ in multi-goal scenarios with UVFA. Consequently, policies, state-value functions, and action-value function transform to $\pi(s_t \| g)$, $V^\pi(s_t \| g)$, and $Q^\pi(s_t \| g,\ a_t)$. (2.1), (2.2), (2.3), and (2.4) remain valid despite these changes.

2.4. Hindsight Experience Replay (HER)

Hindsight Experience Replay (HER) (Andrychowicz et al. 2017) is a RL technique building upon off-policy RL algorithms such as DDPG. It is especially useful in multi-goal UVFA set-ups with sparse rewards, on which we focus in this thesis.

For such tasks, Andrychowicz et al. (2017) assume that every goal $g \in \mathcal{G}$ corresponds to some predicate $f_g : \mathcal{S} \rightarrow \{0, 1\}$. A goal is considered as reached, once the agent achieves any state s that satisfies $f_g(s) = 1$. In an example case, where $\mathcal{G} = \mathcal{S}$, one could define

$$f_g(s) := \begin{cases} 1, & \text{if } g = s \\ 0, & \text{otherwise} \end{cases}, \tag{2.10}$$

when the agent should learn to reach certain goal states s. It is furthermore assumed that for each state s, there is at least one goal g that is achieved in that state, i.e. there is a given mapping $m : \mathcal{S} \rightarrow \mathcal{G}$ such that $f_{m(s)}(s) = 1 \; \forall s \in \mathcal{S}$. Finally, a sparse reward function is defined as

$$r_g(s, a) := \begin{cases} 0, & \text{if } f_g(s) = 1 \\ -1, & \text{otherwise} \end{cases}, \tag{2.11}$$

such that the agent constantly receives negative rewards as long as it has not reached the goal. Only when the goal is reached, zero reward can be observed. The main motivation behind HER can be summarized as follows. When applying a standard off-policy RL algorithm such as DDPG to multi-goal scenarios with sparse rewards, the agent does very rarely encounter informative (non-negative) rewards, leading to a highly inefficient learning process. HER, on the other hand, uses experience replay to provide more meaningful reward feedback to the agent. After each episode consisting of a sequence of reached states $s_0, s_1, ..., s_T$, every transition $(s_t \parallel g,\ a_t,\ r_t,\ s_{t+1} \parallel g)$ is not only stored in the replay buffer with the original goal g used for the episode, but also with some hindsight goals g', i.e. $(s_t \parallel g',\ a_t,\ r_t,\ s_{t+1} \parallel g')$. A possible strategy for choosing such hindsight goals for replay would be selecting the latest state achieved in an episode, i.e. $g' = \mathbb{S}(s_0, ..., s_T) = m(s_T)$. Andrychowicz et al. (2017) call this replay strategy `final` and propose the following alternative replay strategies $\mathbb{S}$:

- `future` - replay k random but future states from the same episode as the transition being replayed
- `episode` - replay k random states from the same episode as the transition being replayed,
- `random` - replay with k random states encountered so far in the entire training process.

The hyper-parameter $k \in \mathbb{N}$ controls the ratio of hindsight replays to normal experience replays in the replay buffer. Generally, the `future` strategy proved to be most successful, which is why it is used in all experiments conducted in this thesis. When replaying hindsight transitions $(s_t \parallel g',\ a_t,\ r_t,\ s_{t+1} \parallel g')$ with goals that have already been achieved, the agent is more likely to encounter non-negative, informative rewards, thus learning progresses. The complete HER algorithm 1 is taken from Andrychowicz et al. (2017), experimental results can be found in Plappert, Andrychowicz, et al. (2018).

Algorithm 1 Hindsight Experience Replay (HER)

```
1: Given:
     • An off-policy RL algorithm $\mathbb{A}$,                          ▷ e.g. DDPG
     • A strategy $\mathbb{S}$ for sampling goals for replay,            ▷ e.g. $\mathbb{S}(s_0, ..., s_T) = m(s_T)$
     • A set of reward functions $r_g : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$   ▷ e.g. $r_g(s,a) = - \mid f_g(s) == 0 \mid$
2: Initialize $\mathbb{A}$
3: Initialize replay buffer $R$
4: for iteration do
5:     Construct a set of $M$ tasks $\{(s_0^i, g^i)\}_{i=1}^{M}$, where $g^i \sim \mathcal{G}_T$ and $s_0^i \sim \mathcal{S}_0$
6:     for $episode = 1, M$ do
7:         $(s_0, g) \leftarrow (\hat{s}_0^i, g^i)$
8:         for $t = 0, T-1$ do
9:             Sample an action $a_t$ using the policy from $\mathbb{A}$ with noise:
                   $$a_t \leftarrow \pi(s_t \parallel g) + \mathcal{N}_t \quad (2.12)$$
10:            Execute the action $a_t$ and observe a new state $s_{t+1}$
11:        for $t = 0, T-1$ do
12:            $r_t := r_g(s_t, a_t)$
13:            Store transition $(s_t \parallel g,\ a_t,\ r_t,\ s_{t+1} \parallel g)$ in $R$   ▷ DDPG experience replay
14:            Sample a set of additional goals for replay $G := \mathbb{S}(current\ episode)$
15:            for $g' \in G$ do
16:                $r' := r_{g'}(s_t, a_t)$
17:                Store the transition $(s_t \parallel g',\ a_t,\ r',\ s_{t+1} \parallel g')$ in $R$   ▷ HER
18:    for $t = 1, N$ do
19:        Sample a minibatch $B$ from the replay buffer $R$
20:        Perform one step of optimization using $\mathbb{A}$ and minibatch $B$   ▷ DDPG:
           (2.5), (2.7), (2.8), (2.9)
```

2.5. Energy-Based Prioritization (EBP)

Energy-Based Prioritization (EBP) (Zhao and Tresp 2018) is a concept to improve the selection of hindsight experience for replay in HER. Instead of randomly sampling transitions from the replay buffer R, Zhao and Tresp (2018) came up with a strategy to prioritize transitions stored in the replay buffer based on the energy of their trajectory. The underlying assumption is simple and valid in most robotic scenarios: when $E_{traj}(\tau)$ (2.13) denotes the total change of energy of a trajectory τ, trajectories with a higher $E_{traj}(\tau)$ are more meaningful for replay than trajectories with lower change of energy. Therefore, in EBP, $E_{traj}(\tau)$ is calculated for each trajectory achieved during exploration based on the cumulative transition energy change $E_{trans}(s_{t-1}, s_t)$ of all time steps t in a trajectory.

$$E_{traj}(\tau) = E_{traj}(s_0, s_1, ..., s_T) = \sum_{t=1}^{T} E_{trans}(s_{t-1}, s_t) \tag{2.13}$$

Furthermore, a priority

$$P(\tau_i) = \frac{E_{traj}(\tau_i)}{\sum_{n=1}^{N_R} E_{traj}(\tau_n)} \tag{2.14}$$

is assigned to each trajectory τ_i, where $\{\tau_n\}_{n=1}^{N_R}$ are the N_R trajectories stored in the replay buffer R.

For a more detailed description and a definition of E_{trans}, see algorithm 2 and Zhao and Tresp (2018). Note that the main difference between HER (algorithm 1) and HER with EBP (algorithm 2) is in lines 11 to 17. Zhao and Tresp (2018) have successfully deployed their EBP strategy to multiple robotic environments, where EBP's performance was significantly better than vanilla HER.

Algorithm 2 HER with Energy-Based Prioritization (EBP)

```
1:  Given:
      • An off-policy RL algorithm $\mathbb{A}$,                                   ▷ e.g. DDPG
      • A strategy $\mathbb{S}$ for sampling goals for replay,          ▷ e.g. $\mathbb{S}(s_0, ..., s_T) = m(s_T)$
      • A set of reward functions $r_g : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$     ▷ e.g. $r_g(s, a) = - \mid f_g(s) == 0 \mid$
2:  Initialize $\mathbb{A}$
3:  Initialize replay buffer $R$
4:  for $iteration$ do
5:      Construct a set of $M$ tasks $\{(s_0^i, g^i)\}_{i=1}^{M}$, where $g^i \sim \mathcal{G}_T$ and $s_0^i \sim \mathcal{S}_0$
6:      for $episode = 1, M$ do
7:          $(s_0, g) \leftarrow (\hat{s}_0^i, g^i)$
8:          for $t = 0, T-1$ do
9:              Sample an action $a_t$ using the policy from $\mathbb{A}$ with noise:
                    $$a_t \leftarrow \pi(s_t \parallel g) + \mathcal{N}_t \quad (2.15)$$
10:             Execute the action $a_t$ and observe a new state $s_{t+1}$
11:         Calculate trajectory energy $E_{traj}(s_0, s_1, ..., s_T) = E_{traj}(\tau)$ (2.13)
12:         Calculate priority $P(\tau)$ (2.14)
13:         for $t = 0, T-1$ do
14:             $r_t := r_g(s_t, a_t)$
15:             Store transition $(s_t \parallel g,\ a_t,\ r_t,\ s_{t+1} \parallel g,\ P,\ E_{traj})$ in $R$
16:             Sample trajectory $\tau$ for replay from $R$ based on priority $P(\tau)$      ▷ EBP
17:             Sample transition $(s_t, a_t, s_{t+1})$ from $\tau$
18:             Sample a set $G := \mathbb{S}(s_{t+1}, ..., s_T)$ from future time steps in $\tau$
19:             for $g' \in G$ do
20:                 $r' := r_{g'}(s_t, a_t)$
21:                 Store the transition $(s_t \parallel g',\ a_t,\ r',\ s_{t+1} \parallel g',\ P,\ E_{traj})$ in $R$     ▷ HER
22:     for $t = 1, N$ do
23:         Sample a minibatch $B$ from the replay buffer $R$
24:         Perform one step of optimization using $\mathbb{A}$ and minibatch $B$          ▷ DDPG:
            (2.5), (2.7), (2.8), (2.9)
```

2.6. Hindsight Goal Generation (HGG)

Hindsight Goal Generation (HGG) (Ren et al. 2019) is an extension of HER to scenarios in which the target goal distribution $\mathcal{G}_T$ is very different from the goal representation of the initial state distribution $m(\mathcal{S}_0)$. An example for such a set-up is the FetchPickNoObstacle environment (Figure 4.3), where the target goal distribution is uniform in the square between the blue balls, while initial states of the object are uniformly sampled from the square between the green balls. Goals sampled from the target goal distribution are generally far away from sampled initial positions of the object, which is why the two distribution are considered as being different from one another. Consequently, when randomly moving the object around starting from its initial position, it is improbable that the task is completed successfully "by accident". Instead, an elaborate policy is necessary for reaching any target goal. If, however, target goals were uniformly sampled from the surface of the table (as in the FetchPush benchmark task provided by Andrychowicz et al. 2017), some sampled target goals would be very close, if not identical, to the sampled initial positions of the object. Hence, the target goal distribution would similar to the goal representation of the initial state distribution in that case. The core idea of HGG, namely using intermediate hindsight goals as an implicit curriculum to guide exploration towards target goals, is motivated as follows.

Remember from section 2.3, that each episode starts with sampling an initial state - goal pair (s_0, g) from an initial state distribution $\mathcal{S}_0$ and a target goal distribution $\mathcal{G}_T$. A state s can be mapped to a goal g_s by $g_s = m(s)$ (see section 2.4). In the beginning of the HER exploration process, exploration is random since no meaningful policy has been developed yet. Exploration naturally starts from $s_0 \sim \mathcal{S}_0$, thus goals that are close to $m(s_0)$ are reached more easily.

Note that Andrychowicz et al. (2017) have only applied HER to scenarios where 1) the target goal distribution $\mathcal{G}_T$ is similar to the goal representation of the initial state distribution $m(\mathcal{S}_0)$, and 2) the non-zero part of the target goal distribution $\mathcal{G}_T$ is continuous (e.g. uniform). When $\mathcal{G}_T$ is similar to $m(\mathcal{S}_0)$, the probability that a sampled goal g is close to $m(s_0)$ is relatively high, thus the agent will, sooner or later, reach a goal and receive an informative, non-negative reward. Due to generalization capabilities of neural networks, the agent can generalize from such experience to goals close to the ones it has already managed to reach. Replaying states from previous trajectories as virtual goals in HER accelerates this procedure: virtual non-negative rewards make the agent learn how to reach states close to s_0 even faster. Due to continuity of $\mathcal{G}_T$, the agent is regularly subjected to goals that are close yet different from the ones he knows how to reach. Through repeated experience and generalization, the agent learns to reach more and more difficult goals, for example those who are far from the initial state goals $m(s_0)$.

When $\mathcal{G}_T$ differs a lot from $m(\mathcal{S}_0)$, all sampled goals g are far from $m(s_0)$ and the agent will never reach a (real) goal through random exploration. Thus, the agent never encounters real positive rewards. In HER, virtual goals that are close to $m(s_0)$ are replayed, since these are the states that are reached first in random exploration. As a consequence of encountered virtual non-negative rewards, the agent learns how to reach these virtual goals, which are, however, still far from the target goals sampled from $\mathcal{G}_T$. Due to this large distance, the agent is unable to generalize from this virtual experience to reaching the target goals. Instead, the agent behaves repeatedly random, and, because of the large difference between $\mathcal{G}_T$ and $m(\mathcal{S}_0)$, does not encounter any real non-negative rewards. As this process repeats itself, the agent keeps learning to reach goals close to sampled s_0, but does not learn more, which is why HER fails in such scenarios. EBP may choose more energetic (and therefore more meaningful) goals for replay, but can only select from previously encountered states generated through random experience. Hence, when $\mathcal{G}_T$ is very far from $m(\mathcal{S}_0)$, EBP fails as well.

A solution to this problem is Hindsight Goal Generation (HGG). The idea behind this approach is to guide exploration by replacing target goals with more meaningful intermediate goals. On the one hand, such intermediate goals should not be too far from goals that the agent already knows to reach, allowing the neural networks to generalize from this experience. On the other hand, they should not be too close similar to already known goals, such that the agent does not learn anything new.

Ren et al. (2019) derive this concept more formally:

- A value function of a policy π for a specific goal g is assumed to have some generalizability to another goal g' close to g. This assumption is mathematically characterized via Lipschitz continuity and has been used by Asadi, Misra, and Littman (2018) as well as Luo et al. (2019):

  $$|\ V^{\pi}(s \parallel g) - V^{\pi}(s' \parallel g')\ | \leq\ L \cdot d(s \parallel g,\ s' \parallel g')\ , \tag{2.16}$$

  where $d(s \parallel g,\ s' \parallel g')$ is a metric defined as

  $$d((s \parallel g),\ (s' \parallel g')) := c \parallel m(s) - m(s') \parallel + \parallel g - g' \parallel . \tag{2.17}$$

  The hyper-parameter $c > 0$ provides a trade-off between 1) the distance between target goals and 2) the distance between the goal representation of the initial states. Even though Lipschitz continuity cannot be guaranteed for every $s, s' \in \mathcal{S}$ and $g, g' \in \mathcal{G}$, it is only required over some specific region. In the considered environments, the generalizability condition (2.16) is satisfied by most of the $s \parallel g$ and $s' \parallel g'$ when $d(s \parallel g,\ s' \parallel g')$ is not too large, according to Ren et al. (2019).

- Given that the generalizability condition (2.16) holds for two distributions $\mathcal{T} : \mathcal{S} \times \mathcal{G} \to \mathbb{R}$ and $\mathcal{T}' : \mathcal{S} \times \mathcal{G} \to \mathbb{R}$, where $s \parallel g \sim \mathcal{T}$ and $s' \parallel g' \sim \mathcal{T}'$, Ren et al. (2019) have proven that

$$V^{\pi}(\mathcal{T}') \geq V^{\pi}(\mathcal{T}) - L \cdot \mathcal{D}(\mathcal{T}, \mathcal{T}') , \tag{2.18}$$

  where $\mathcal{D}(\cdot, \cdot)$ is the Wasserstein distance based on $d(\cdot, \cdot)$, defined as

$$\mathcal{D}(\mathcal{T}^{(1)}, \mathcal{T}^{(2)}) := \inf_{\mu \in \Gamma(\mathcal{T}^{(1)}, \mathcal{T}^{(2)})} \left(\mathbb{E}_{\mu} \left[d(s_0^{(1)} \parallel g^{(1)},\ s_0^{(2)} \parallel g^{(2)}) \right] \right) . \tag{2.19}$$

  $\Gamma(\mathcal{T}^{(1)}, \mathcal{T}^{(2)})$ denotes the collection of all joint distributions $\mu(s_0^{(1)}, g^{(1)}, s_0^{(2)}, g^{(2)})$ whose marginal probabilities are $\mathcal{T}^{(1)}$, $\mathcal{T}^{(2)}$, respectively.

- When $\mathcal{T}^* : \mathcal{S} \times \mathcal{G} \to \mathbb{R}$ denotes the joint distribution over initial states $s_0 \sim \mathcal{S}_0$ and goals $g \sim \mathcal{G}_T$, the agent tries to find a policy $\pi : \mathcal{S} \times \mathcal{G} \to \mathcal{A}$ maximizing the expectation of the discounted cumulative reward based on the state value function defined in (2.1):

$$V^{\pi}(\mathcal{T}^*) := \mathbb{E}_{s_0 \parallel g \sim \mathcal{T}^*} \left[V^{\pi}(s_0 \parallel g) \right] . \tag{2.20}$$

- From (2.18), it can be derived that optimizing the expected cumulative reward (2.20) can be relaxed into the substitute problem

$$\max_{\mathcal{T}, \pi} \ V^{\pi}(\mathcal{T}) - L \cdot \mathcal{D}(\mathcal{T}, \mathcal{T}^*) . \tag{2.21}$$

  This step is crucial: instead of optimizing V^{π} with the difficult target goal - initial state distribution $\mathcal{T}^*$, which carries the risk of being too far away from known goals (such that the neural networks are unable to generalize), Ren et al. (2019) try to optimize with a set of intermediate goals sampled from $\mathcal{T}$. On the one hand, the goals contained in $\mathcal{T}$ should be easy to optimize and generalize from, which requires a high $V^{\pi}(\mathcal{T})$ term. On the other hand, goals within $\mathcal{T}$ should be close enough to $\mathcal{T}^*$ to be challenging for the agent, requiring a low $L \cdot \mathcal{D}(\mathcal{T}, \mathcal{T}^*)$ term.

Joint optimization of π and $\mathcal{T}$ (2.21) is non-trivial. Therefore, Ren et al. (2019) have chosen to adopt the idea of using hindsight goals from HER: $\mathcal{T}$ is enforced to be a finite set of K initial state - goal pairs (s_0, g) belonging to trajectories τ that are already stored in the replay buffer R. The optimization problem is finally solved in a two-stage iterative algorithm. As a first step, standard policy optimization as in DDPG and HER maximizes the value function based on experience generated from the intermediate task set $\mathcal{T}$. In the second step, the intermediate task set $\mathcal{T}$ is optimized while the policy

π is kept constant. Ren et al. (2019) realized that this second optimization step is a variant of the Wasserstein Barycenter problem with the value function as bias term for each initial state - goal pair, which can be solved as a bipartite matching problem (Duan and Su 2012). For this to work, $\mathcal{T}^*$ is approximated by K sampled initial state - goal pairs, resulting in the set $\hat{\mathcal{T}}^* = \{(\hat{s}_0^i, \hat{g}^i)\}_{i=1}^K$. Now, for every $(\hat{s}_0^i, \hat{g}^i) \in \hat{\mathcal{T}}^*$, a trajectory from the replay buffer $\tau^i = \{s_t^i\}_{t=1}^T \in R$ is identified that minimizes

$$w((\hat{s}_0^i, \hat{g}^i), \tau^i) := c \parallel m(\hat{s}_0^i) - m(s_0^i) \parallel + \min_t \left(\parallel \hat{g}^i - m(s_t^i) \parallel - \frac{1}{L} V^\pi(s_0^i \parallel m(s_t^i)) \right) . \tag{2.22}$$

where the unknown Lipschitz constant L is treated as a hyper-parameter. All together, these K trajectories τ^i minimize the sum

$$\sum_{(\hat{s}_0^i, \hat{g}^i) \in \hat{\mathcal{T}}^*} w((\hat{s}_0^i, \hat{g}^i), \tau^i) . \tag{2.23}$$

Finally, for each of the K selected trajectories τ^i, the hindsight goal g^i is derived from the state $s_t^i \in \tau^i$, that minimized (2.22). More formally,

$$g^i = m\left(\operatorname*{arg\,min}_{s_t^i \in \tau_i} \left(\parallel \hat{g}^i - m(s_t^i) \parallel - \frac{1}{L} V^\pi(s_0^i \parallel m(s_t^i)) \right) \right) . \tag{2.24}$$

In summary, HGG uses previously achieved virtual goals stored in the replay buffer to generate a curriculum of meaningful hindsight goals, guiding exploration towards target goals. Hindsight goals are selected based on a trade-off between two main criteria. The first criterion is the distance between the potential hindsight goal $m(s_t^i)$ and the target goal $\hat{g}^i$ represented by the term $\parallel \hat{g}^i - m(s_t^i) \parallel$ in (2.22) and (2.24), which should be as small as possible for the hindsight goal to be meaningful. The second criterion is the expected return obtained with a potential hindsight goal $m(s_t^i)$, represented by the term $V^\pi(s_0^i \parallel m(s_t^i))$ in (2.22) and (2.24), which should be as high as possible to allow the underlying neural networks to learn. The HGG algorithm 3 is taken from Ren et al. (2019). Note that the main difference between HER (algorithm 1) and HGG is in step 5 and 7. Instead of exploring with target goals, hindsight goals are generated and used for exploration in every iteration. Since EBP changes HER from step 11 onward, HGG is compatible to EBP.

Algorithm 3 Hindsight Goal Generation (HGG)

```
1: Given:
     • An off-policy RL algorithm $\mathbb{A}$,                                   ▷ e.g. DDPG
     • A strategy $\mathbb{S}$ for sampling goals for replay,        ▷ e.g. $\mathbb{S}(s_0, ..., s_T) = m(s_T)$
     • A set of reward functions $r_g : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$    ▷ e.g. $r_g(s,a) = - \mid f_g(s) == 0 \mid$
2: Initialize $\mathbb{A}$
3: Initialize replay buffer $R$
4: for iteration do
5:     Construct a set of $M$ intermediate tasks $\{(\hat{s}_0^i, g^i)\}_{i=1}^M$:                ▷ HGG
         • Sample target tasks $\{(\hat{s}_0^i, \hat{g}^i)\}_{i=1}^K \sim \mathcal{T}^*$
         • Find $K$ distinct trajectories $\{\tau^i\}_{i=1}^K$ that together minimize (2.23)   ▷ weighted
           bipartite matching
         • Find $M$ intermediate tasks $(\hat{s}_0^i, g^i)$ by selecting and intermediate goal $g^i$ from
           each $\tau^i$ (2.24)
6:     for episode = 1, $M$ do
7:         $(s_0, g) \leftarrow (\hat{s}_0^i, g^i)$                        ▷ hindsight goal-oriented exploration
8:         for $t = 0, T-1$ do
9:             Sample an action $a_t$ using the policy from $\mathbb{A}$ with noise:
                           $a_t \leftarrow \pi(s_t \parallel g) + \mathcal{N}_t$                         (2.25)
10:            Execute the action $a_t$ and observe a new state $s_{t+1}$
11:        for $t = 0, T-1$ do
12:            $r_t := r_g(s_t, a_t)$
13:            Store transition $(s_t \parallel g,\ a_t,\ r_t,\ s_{t+1} \parallel g)$ in $R$   ▷ DDPG experience replay
14:            Sample a set of additional goals for replay $G := \mathbb{S}(current\ episode)$
15:            for $g' \in G$ do
16:                $r' := r_{g'}(s_t, a_t)$
17:                Store the transition $(s_t \parallel g',\ a_t,\ r',\ s_{t+1} \parallel g')$ in $R$          ▷ HER
18:    for $t = 1, N$ do
19:        Sample a minibatch $B$ from the replay buffer $R$                  ▷ HER or EBP
20:        Perform one step of optimization using $\mathbb{A}$ and minibatch $B$          ▷ DDPG:
           (2.5), (2.7), (2.8), (2.9)
```

3. Methodology

In this chapter, we first define the concrete problem of sparse-reward object manipulation RL in environments with obstacles, where HGG reaches its limits (section 3.1). We then introduce G-HGG as a solution, which is an extension of HGG using a graph-based distance measure (section 3.2) instead of the euclidean norm.

3.1. Problem Formulation

In this thesis, we focus on solving robotic object manipulation tasks with sparse-reward RL, where the goal is to move an object to a certain point in 3-D space with a robotic gripper arm. Such scenarios show the following characteristics:

1. A multidimensional action space $\mathcal{A} \subset \mathbb{R}^l$, $l \in \mathbb{N}$. The action space can include parameters to drive actuators, for example. In our experiments (chapter 4), it consists of the end effector's position in 3-D space and a gripper's opening control parameter that is to be applied in the next time step via inverse kinematics (see chapter 4).
2. A multidimensional state space $\mathcal{S} \subset \mathbb{R}^n$, $n \in \mathbb{N}$ with $n \geq 3$. This state space can include the current location of the end effector and the object as well as joint positions, velocities, or sensor data.
3. An initial state distribution $\mathcal{S}_0 : \mathcal{S} \to \mathbb{R}$. Initial states should be reasonable starting conditions for the robot to start the task.
4. A three-dimensional goal space $\mathcal{G} \subset \mathbb{R}^3$. A goal is defined as a point in 3-D space.
5. A target goal distribution $\mathcal{G}_T : \mathcal{G} \to \mathbb{R}$. Depending on this target goal distribution, goals $g \sim \mathcal{G}_T$ can be close to or far from the initial state's goal representation $m(s_0)$, where $s_0 \sim \mathcal{S}_0$.
6. A goal predicate $f_g : \mathcal{S} \to \{0, 1\}$, $g \in \mathcal{G}$ defined as

$$f_g(s) := \begin{cases} 1, & \text{if } \parallel g - m(s) \parallel \leq \delta_g \\ 0, & \text{otherwise} \end{cases} \tag{3.1}$$

with distance threshold δ_g and a known and traceable mapping $m : \mathcal{S} \rightarrow \mathcal{G}$ defining the goal representation of states.

7. A sparse reward function $r_g : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$, $g \in \mathcal{G}$ defined as

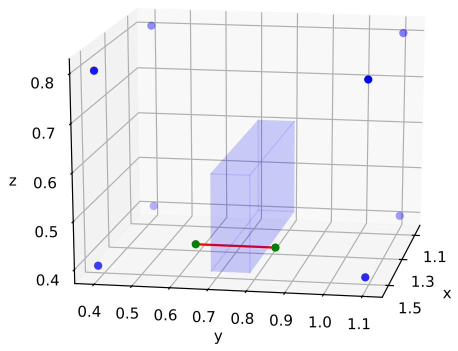
$$r_g(s,a) = \begin{cases} 0, & \text{if } f_g(s) = 1 \\ -1, & \text{otherwise} \end{cases} . \tag{3.2}$$

   This means that the agent constantly receives negative rewards as long as it has not reached the goal. In our experiments (chapter 4), a goal is considered as reached, when the manipulated position of he object is close to the goal (3.1). $m(s)$ trivially extracts the object's position in 3-D space from the current state s.

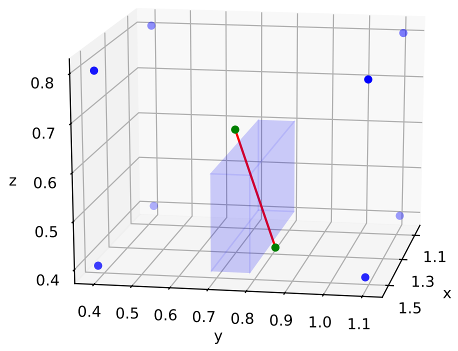
8. The environment is known. In particular, the exact locations of all obstacles in goal space $\mathcal{G}$, i.e. all goals that are not reachable due to material constraints, can be identified. We only allow obstacles that can be represented as a continuous, bounded and convex set of goals $g_{obstacle} \in \mathcal{G}$.

The most important characteristics from this list are the three-dimensional goal space and the fact, that every state $s \in \mathcal{S}$ can be mapped to a point in that 3-D goal space, which can then be used as a virtual goal. Even though the general principle of HGG is applicable to set-ups with above characteristics, there is one severe limitation. Solving the Wasserstein Barycenter problem in (2.22), (2.23), and (2.24) includes the computation of distances between a goal $g \in \mathcal{G}$ and the goal representation $m(s)$ of a state $s \in \mathcal{S}$. As long as the euclidean norm is a reasonable measure for this distance, HGG can be applied as is. If, however, the euclidean norm is not applicable, a new distance measure needs to be found.

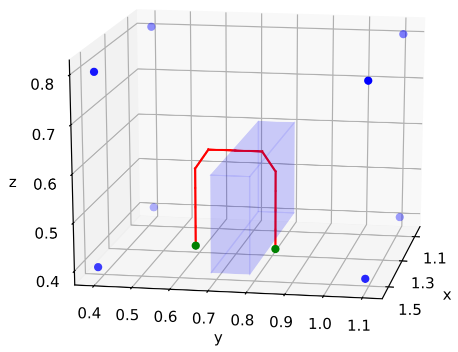
In this thesis, we will focus on robotic object manipulation in 3-D space with obstacles. In such a setting, the euclidean norm used in HGG is generally not applicable. Consider an environment, for example, where the goal is to pick up an object, lift it over an obstacle, and place it at the target goal position. Such an environment is illustrated in Figure 3.1, where the obstacle is represented by a blue box. Let us consider two cases A (Figures 3.1a and 3.1c) and B (Figures 3.1b and 3.1d) in order to compare distances between two points in space (marked in green). In Figures 3.1a and 3.1b, the euclidean norm is applied to calculate the distance $d_{euc} = \parallel g_2 - g_1 \parallel$, where g_1 and g_2 are two points (goals) in space. Since this distance measure completely ignores the obstacle, $d_{euc}^A < d_{euc}^B$. If the object manipulation task had to be completed in an environment without obstacles, this would be an accurate distance measure applicable to hindsight goal evaluation in HGG. In environments with obstacles, however, the object to be manipulated cannot go through these obstacles, which is why the euclidean norm is not a suitable distance measure. Intuitively, the path an object has to take, in order to get from one point to the other in the presence of the obstacle, is longer in case A than in case B. Therefore, a suitable alternative, obstacle-avoiding shortest path based distance measure similar to the one visualized in Figures 3.1c and 3.1d is required in such a scenario, assigning a smaller distance $d^B_{alt} < d^A_{alt}$ to case B.



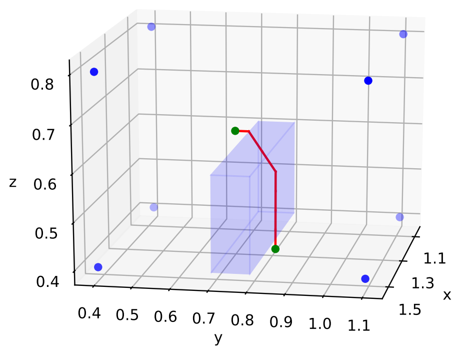
(a) Case A with euclidean norm based distance: the shortest path goes through the obstacle, distance $d^A_{euc} = 0.210$.



(b) Case B with euclidean norm based distance: the shortest path goes through the obstacle, distance $d^B_{euc} = 0.283$.



(c) Case A with alternative distance: the shortest path avoids the obstacle, distance $d^A_{alt} = 0.566$.



(d) Case B with alternative distance: the shortest path avoids the obstacle, distance $d^B_{alt} = 0.301$.

Figure 3.1.: Shortest distances between two points (green) in 3-D space with obstacle.

As this brief example suggests, using a distance measure based on the shortest obstacle-avoiding path between two points would allow for an application of HGG to object manipulation tasks in 3-D space with obstacles. The problem of finding the shortest path between two points without intersecting any obstacle is known as the Euclidean Shortest Path problem, which is a very important problem in robotic path planning. While the Euclidean Shortest Path problem is NP (non-deterministic polynomial time) - hard in the general case for three or more dimension, it can be solved in polynomial time with relaxed constraints, for example when obstacles have polyhedral shape (Canny and Reif 1987). In the 2-D case, optimal solutions have been found, for example by Hershberger and Suri (1999). Gasparetto, Boscariol, and Lanzutti (2015) provide an overview of common shortest path planning algorithms, which can be categorized into road-map techniques, cell decomposition algorithms, and artificial potential methods. Many of these algorithms, for example visibility graph based road-map algorithms or certain cell decomposition algorithms, reduce the problem to a graph, on which shortest paths can easily be computed via e.g. Dijkstra's Algorithm.

At first, one could think that using for example 3-D visibility graphs (Bygi and Ghodsi 2007) to calculate the shortest path between two points and then utilizing its length as a distance measure would be a suitable solution for extending HGG to environments with obstacles. However, this approach is not feasible in reality. Calculating one shortest path based distance requires the creation of an at least partly new graph as well as an entire run of Dijkstra's (or any other) shortest path algorithm. Even for simple environments with few polyhedral obstacles resulting in small graphs, this approach is computationally expensive and not feasible for computing $5 \cdot 10^6$ distances in each iteration of HGG.

An alternative to computing optimal shortest path based distances online is pre-computation of approximate shortest path distances. We have developed an algorithm called Graph-based Hindsight Goal Generation (G-HGG), in which approximate shortest distances between discrete points in 3-D space are pre-computed and used to compare distances between potential hindsight goals. Of course, using approximate shortest path based distances bears the risk that some of these comparisons are inexact. However, sampling goals and initial states as well as random and noisy exploration already introduce a large amount of uncertainty to the algorithm, which is why a certain error in distance calculation can be tolerated. In chapter 4, we show that G-HGG can successfully solve RL object manipulation tasks in environments with obstacles, whereas both vanilla HGG and HER fail. Moreover, G-HGG performs reasonably well in an environment without obstacles solvable by HGG.

3.2. Graph-Based Hindsight Goal Generation (G-HGG)

Graph-based Hindsight Goal Generation (G-HGG) is an extension of HGG to environments with obstacles, where the euclidean norm is not applicable as a distance measure. Hence, we reformulate (2.22) and (2.24), replacing the euclidean norm with a graph-based distance measure d.

With this new formulation, for every $(\hat{s}_0^i, \hat{g}^i) \sim \hat{\mathcal{T}}^*$, $i \in \{1, 2, ..., K\}$, we find a trajectory from the replay buffer $\tau^i = \{s_t^i\}_{t=1}^T \in R$ that minimizes

$$w((\hat{s}_0^i, \hat{g}^i), \tau^i) := c \parallel m(\hat{s}_0^i) - m(s_0^i) \parallel + \min_t \left(d(\hat{g}^i - m(s_t^i)) - \frac{1}{L} V^\pi(s_0^i \parallel m(s_t^i)) \right), \tag{3.3}$$

where the unknown Lipschitz constant L is treated as a hyper-parameter. All together, these K trajectories τ^i minimize the sum

$$\sum_{(\hat{s}_0^i, \hat{g}^i) \in \hat{\mathcal{T}}^*} w((\hat{s}_0^i, \hat{g}^i), \tau^i) . \tag{3.4}$$

Finally, from each of the K selected trajectories τ^i, a hindsight goal g^i is chosen based on the state $s_t^i \in \tau^i$ minimizing (3.3):

$$g^i = m\left(\underset{s_t^i \in \tau_i}{\arg\min} \left(d(\hat{g}^i - m(s_t^i)) - \frac{1}{L} V^\pi(s_0^i \parallel m(s_t^i)) \right) \right) . \tag{3.5}$$

As previously stated, the distance measure d (3.15) is based on shortest paths in a graph with a discrete representation of the goal space $\mathcal{G}$ as vertices. The computation of these shortest path based distances is done before training and consists of constructing a graph representing the environment (subsection 3.2.1), and pre-computing shortest distances between all graph vertices (subsection 3.2.2).

3.2.1. Graph Construction

As a first step of G-HGG, we want to represent our goal space by a graph. A graph $G := (V, E)$ consists of a set of vertices V and a set of weighted edges E, where each edge is characterized by a pair of connected vertices (v_1, v_2) and an assigned weight w (3.6). In this thesis, all graphs are undirected.

$$E \subset \{(v_1, v_2, w) \mid (v_1, v_2) \in V^2 \text{ and } v_1 \neq v_2 \text{ and } w \in \mathbb{R}\} \tag{3.6}$$

For the graph to be an accurate representation of the goal space, we define the set of vertices V as a discrete subset of the continuous goal space $V \subset \mathcal{G}$. Since $\mathcal{G}$ is not bounded and contains infinitely many goals g, we choose a continuous but bounded subset of the goal space $\mathcal{G}_A \subset \mathcal{G}$ called the accessible goal space $\mathcal{G}_A$. $\mathcal{G}_A$ contains all potential goals that the object can reach when it is manipulated, regardless of whether these goals are desired or not. If the object achieves any goal outside of $\mathcal{G}_A$, this goal is irrelevant to us and ignored in further learning. Of course, the target goal distribution $\mathcal{G}_T : \mathcal{G} \to \mathbb{R}$ is zero for all goals $g \notin \mathcal{G}_A$, which is why it can be restricted to $\mathcal{G}_T : \mathcal{G}_A \to \mathbb{R}$. In environments with obstacles, which are of high interest to us, goals $g_{obstacle} \in \mathcal{G}$ lying within an obstacle that are blocked from being reached are explicitly excluded from the accessible goal space $g_{obstacle} \notin \mathcal{G}_A$.

In order to keep the number of vertices in V reasonably low, the set of vertices V is defined as a subset of the accessible goal space $\mathcal{G}_A$:

$$V \subset \mathcal{G}_A \subset \mathcal{G} \; . \tag{3.7}$$

Since $\mathcal{G}_A$ is bounded, it can be characterized by values or functions x_{min}, x_{max}, y_{min}, y_{max}, z_{min}, $z_{max} \in \mathbb{R}$, describing the finite infimum and supremum values in each coordinate direction, potentially depending on the other coordinate directions. In our MuJoCo environments (Figures 4.1, 4.2, 4.3, and 4.4), $\mathcal{G}_A$ is depicted by a transparent blue box.

In order to create the set of vertices $V \subset \mathcal{G}_A$, we choose $n_x, n_y, n_z \in \mathbb{N}$, such that the number of vertices in V is $n = n_x \cdot n_y \cdot n_z$. Based on these choices, we can now define the distance between two adjacent vertices in each coordinate direction $\Delta_x, \Delta_y, \Delta_z \in \mathbb{R}$ as

$$\Delta_x := \frac{x_{max} - x_{min}}{n_x - 1}, \;\; \Delta_y := \frac{y_{max} - y_{min}}{n_y - 1}, \;\; \Delta_z := \frac{z_{max} - z_{min}}{n_z - 1} \; . \tag{3.8}$$

Finally, we define the set of vertices V as

$$V := \hat{V} \cap \mathcal{G}_A \; , \tag{3.9}$$

where

$$\begin{aligned} \hat{V} := \{(x_{min} + \Delta_x \cdot i,\ y_{min} + \Delta_y \cdot j,\ z_{min} + \Delta_z \cdot k) \;\mid\; & i \in \{0, n_x - 1\}, \\ & j \in \{0, n_y - 1\}, \\ & k \in \{0, n_z - 1\}\} \; . \end{aligned} \tag{3.10}$$

Let us visualize this set of vertices in a concrete demo environment with an obstacle. In Figure 3.2, $\mathcal{G}_A$ is defined as the cuboid space marked by the blue balls, without the obstacle depicted by the blue box. $n_x = n_y = n_z = 4$. It can be seen that vertices are evenly distributed in $\mathcal{G}_A$, such that no vertex lies inside an obstacle. In the following, we use $g = (x, y, z) \in \mathcal{G}_A$ to denote elements of the continuous accessible goal space and $v = (\hat{x}, \hat{y}, \hat{z}) \in V$ for discrete vertices, that are, per definition, contained in $\mathcal{G}_A$ as well.

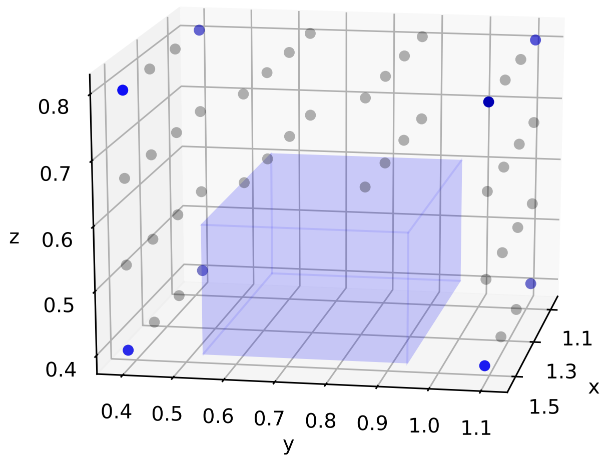


Figure 3.2.: Vertices in demo obstacle environment

As a next step, we connect two vertices $v_1 = (\hat{x}_1, \hat{y}_1, \hat{z}_1) \in V$, $v_2 = (\hat{x}_2, \hat{y}_2, \hat{z}_2) \in V$ with an edge of weight w if the following holds:

$$\begin{aligned}(v_1, v_2, w) \in E \iff & \mid \hat{x}_2 - \hat{x}_1 \mid \leq \Delta_x \text{ and} \\ & \mid \hat{y}_2 - \hat{y}_1 \mid \leq \Delta_y \text{ and} \\ & \mid \hat{z}_2 - \hat{z}_1 \mid \leq \Delta_z\end{aligned} \tag{3.11}$$

with

$$w := \sqrt{(\hat{x}_2 - \hat{x}_1)^2 + (\hat{y}_2 - \hat{y}_1)^2 + (\hat{z}_2 - \hat{z}_1)^2}\,. \tag{3.12}$$

In environments with obstacles, it is important to make sure that obstacles are not overlooked. To ensure that, we require every obstacle to be detected by at least one potential vertex $v \in \hat{V}$ but $v \notin \mathcal{G}_A$. Since we only consider continuous, bounded, and convex sets of goals not included in $\mathcal{G}_A$ as obstacles, every obstacle can be characterized by values x^{obs}_{min}, x^{obs}_{max}, y^{obs}_{min}, y^{obs}_{max}, z^{obs}_{min}, $z^{obs}_{max} \in \mathbb{R}$, describing the finite infimum and supremum values of the obstacle in each coordinate direction. To guarantee the detection of each obstacle, we require the graph to satisfy the vertex density criterion (3.13) for every obstacle, i.e. continuous, bounded, and convex set of goals not included in $\mathcal{G}_{\mathcal{A}}$.

$$\begin{aligned} \Delta_x &< x_{max}^{obs} - x_{min}^{obs} \\ \Delta_y &< y_{max}^{obs} - y_{min}^{obs} \\ \Delta_z &< z_{max}^{obs} - z_{min}^{obs} \end{aligned} \tag{3.13}$$

When edges are defined according to (3.11), every vertex is connected to at most 26 adjacent vertices. In Figure 3.3, one can see that the black vertex is connected to nine vertices with a higher z-coordinate (red), eight vertices with the same z-coordinate (pink), and nine vertices with a lower z-coordinate (blue). However, since the set of vertices V only contains vertices $v \in \mathcal{G}_A$ (3.9), the number of outgoing edges of a vertex is often less than 26 in environments with obstacles as long as the vertex density criterion (3.13) is satisfied. The final graph including edges and vertices in our demo environment is visualized in Figure 3.4.

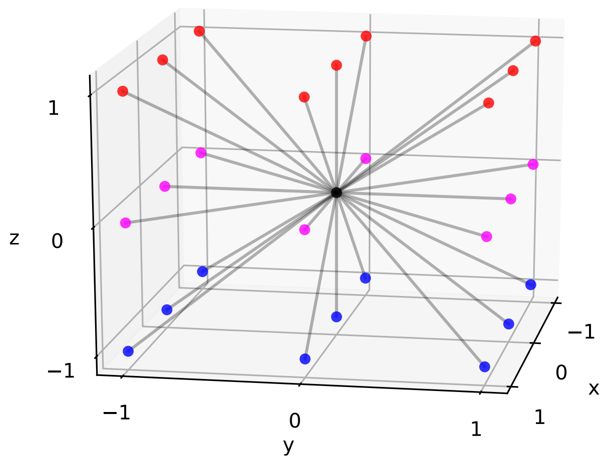


Figure 3.3.: Outgoing edges from one vertex

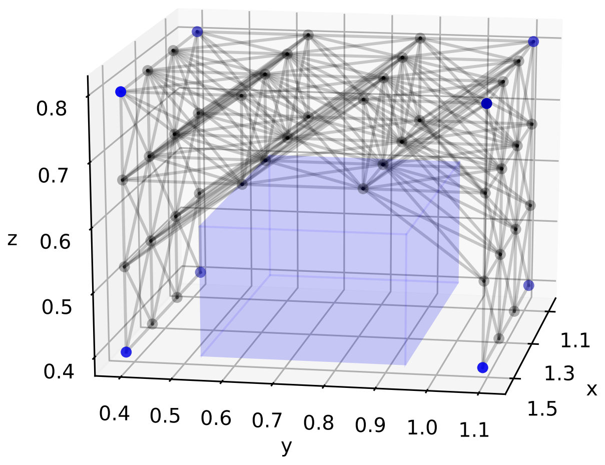


Figure 3.4.: Graph in demo obstacle environment

3.2.2. Shortest Distance Computation

Now that we have found a graph that represents our environment, we can use a shortest path algorithm such as Dijkstra's algorithm (Dijkstra 1959) to calculate shortest paths and shortest distances $\hat{d}_G : V^2 \to \mathbb{R}$ between every possible pair of vertices $(v_1, v_2) = \big((\hat{x}_1, \hat{y}_1, \hat{z}_1),\ (\hat{x}_2, \hat{y}_2, \hat{z}_2)\big) \in V^2$ in a graph $G = (V, E)$. Note that all values of the shortest distance function $\hat{d}_G(v_1, v_2)\ \forall\ v_1 \in V,\ v_2 \in V$ can be efficiently pre-computed with Dijkstra and stored in an $n \times n$ table, where n denotes the number of vertices in V.

More formally, $\hat{d}_G : V^2 \to \mathbb{R}$ on a graph $G = (V, E)$ is defined such that

$$\hat{d}_G(v_1, v_2) := \text{ShortestDistance}_G(v_1, v_2) = \text{DijkstraDistance}_G(v_1, v_2)\ . \tag{3.14}$$

Bringing everything together, the graph-based distance d between two goals used in (3.3) and (3.5) is approximated by the shortest distance between the two vertices that are closest to each of the two goals. In case at least one of the two goals is not contained in the accessible goal space $\mathcal{G}_A$, it is considered as not reachable or irrelevant, and the graph-based distance between the two goals is set to infinity. A more formal description follows in (3.15) and (3.16).

Given two goals $g_1 = (x_1, y_1, z_1) \in \mathcal{G}$ and $g_2 = (x_2, y_2, z_2) \in \mathcal{G}$ and a graph $G = (V, E)$ representing the approximate goal space $\mathcal{G}_A \subset \mathcal{G}$ with x_{min}, x_{max}, y_{min}, y_{max}, z_{min}, z_{max}, Δ_x, Δ_y, and Δ_z, the graph-based distance $d : \mathcal{G}^2 \to \mathbb{R}$ is defined such that

$$d(g_1, g_2) := \begin{cases} \hat{d}_G\Big(\nu(g_1),\ \nu(g_2)\Big), & \text{if } g_1 \in \mathcal{G}_A \text{ and } g_2 \in \mathcal{G}_A \\ \infty, & \text{otherwise} \end{cases}, \tag{3.15}$$

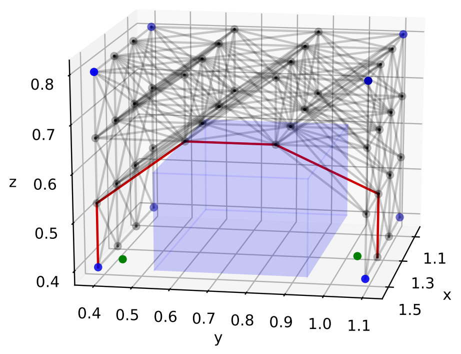
where $\nu : \mathcal{G}_A \to V$ maps goals in $\mathcal{G}_A$ to the closest vertex in V:

$$\begin{aligned} &\nu(g) = \nu(x, y, z) = (\hat{x}, \hat{y}, \hat{z}) := \\ &\left(x_{min} + \Delta_x \cdot \left\lfloor \frac{x - x_{min}}{\Delta_x} \right\rceil, \ y_{min} + \Delta_y \cdot \left\lfloor \frac{y - y_{min}}{\Delta_y} \right\rceil, \ z_{min} + \Delta_z \cdot \left\lfloor \frac{z - z_{min}}{\Delta_z} \right\rceil \right), \end{aligned} \tag{3.16}$$

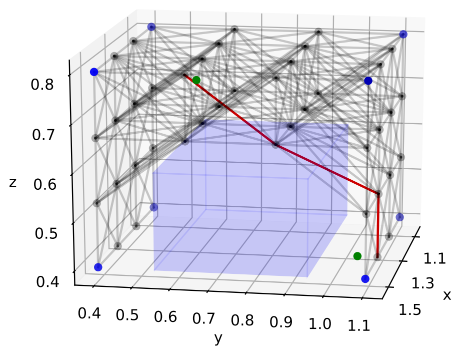
with $\lfloor a \rceil$ rounding any $a \in \mathbb{R}$ to the closest integer value.

In Figure 3.5, a visualization of graph-based goal distance computation between goals $g_1 \in \mathcal{G}_A$ and $g_2 \in \mathcal{G}_A$ is provided. In cases A (Figure 3.5a) and B (Figure 3.5b), both goals are contained in the accessible goal space $\mathcal{G}_A$ since they lie inside the cuboid defined by the blue balls and outside the blue obstacle box. Therefore, the graph-based distance is approximated by the shortest distance between the two vertices $\nu(g_1)$ and $\nu(g_2)$ that are closest to the goals g_1 and g_2, respectively. In mathematical terms, $d(g_1, g_2) = \hat{d}_G(\nu(g_1),\ \nu(g_2))$.

In Figure 3.6, however, goal $g_1 \in \mathcal{G}$ is not contained in the accessible goal space $g_1 \notin \mathcal{G}_A$. In case C (Figure 3.6a), this is due to the location of g_1 inside the obstacle (blue box), whereas in case D, g_1 is located outside the defined boundaries of $\mathcal{G}_A$ (blue balls). In both cases, no shortest path can be computed and the graph-based distance is set to $d(g_1, g_2) = \infty$.

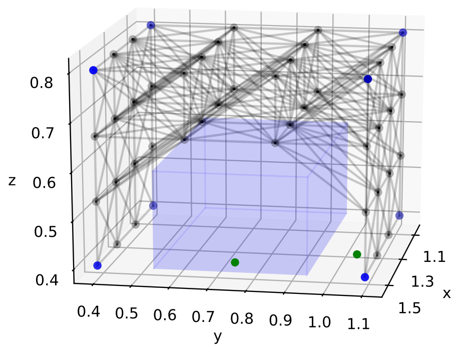


(a) Case A: both goals $g_1, g_2 \in \mathcal{G}_A$. Therefore, $d(g_1, g_2) = \hat{d}_G(v(g_1),\ v(g_2)) = 1.085$.

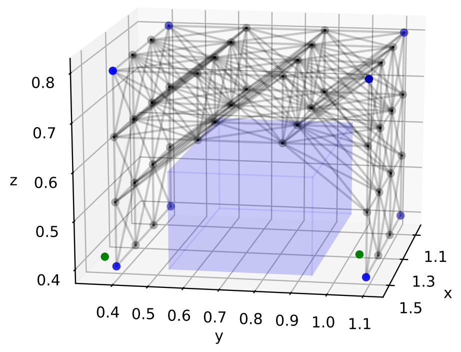


(b) Case B: both goals $g_1, g_2 \in \mathcal{G}_A$. Therefore, $d(g_1, g_2) = \hat{d}_G(v(g_1),\ v(g_2)) = 0.718$.

Figure 3.5.: Graph-based distances and shortest paths (red) between two goals (green) $g_1 \in \mathcal{G}_A$, $g_2 \in \mathcal{G}_A$ in demo obstacle environment.



(a) Case C: goal $g_1 \notin \mathcal{G}_A$ since it lies inside the obstacle. Therefore, $d(g_1, g_2) = \infty$.



(b) Case D: goal $g_1 \notin \mathcal{G}_A$ since it lies outside of the defined boundaries of $\mathcal{G}_A$ (cuboid space marked by the blue balls). Therefore, $d(g_1, g_2) = \infty$.

Figure 3.6.: Graph-based distances between two goals (green) $g_1 \notin \mathcal{G}_A$, $g_2 \in \mathcal{G}_A$ in demo obstacle environment. Shortest paths do not exist.

3.2.3. G-HGG: Algorithm

The entire G-HGG algorithm is described in algorithm 4. Note that the main difference between HGG (algorithm 3) and G-HGG (algorithm 4) is in steps 2, 3, and 7. Step 2 (graph construction, subsection 3.2.1) and step 3 (shortest distance computation, subsection 3.2.2) are performed pre-training. The pre-computed table of shortest distances $\hat{d}_G$ (3.14) is required in step 7, where (3.3), (3.4), and (3.5) use $d \approx \hat{d}_G$ (3.15) for hindsight goal generation. Just like HGG, G-HGG is complementary and therefore compatible to other improvements of HER such as EBP, which focus on optimizing the replay process.

Algorithm 4 Graph-Based Hindsight Goal Generation (G-HGG)

```
1:  Given:
      • An off-policy RL algorithm $\mathbb{A}$,                          ▷ e.g. DDPG
      • A strategy $\mathbb{S}$ for sampling goals for replay,   ▷ e.g. $\mathbb{S}(s_0, ..., s_T) = m(s_T)$
      • A set of reward functions $r_g : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$   ▷ e.g. $r_g(s,a) = - \mid f_g(s) == 0 \mid$
2:  Construct a graph $G = (V, E)$ as a discrete representation of a pre-defined accessible
    goal space $\mathcal{G}_A \subset \mathcal{G}$                                         ▷ section 3.2.1
3:  Pre-compute shortest distances $\hat{d}_G$ between every pair of vertices $(v_1, v_2) \in V^2$ with
    Dijkstra                                                                ▷ section 3.2.2
4:  Initialize $\mathbb{A}$
5:  Initialize replay buffer $R$
6:  for iteration do
7:      Construct a set of $M$ intermediate tasks $\{(\hat{s}_0^i, g^i)\}_{i=1}^M$:              ▷ HGG
      • Sample target tasks $\{(\hat{s}_0^i, \hat{g}^i)\}_{i=1}^K \sim \mathcal{T}^*$
      • Find $K$ distinct trajectories $\{\tau^i\}_{i=1}^K$ that together minimize (3.4)   ▷ weighted
        bipartite matching, based on $d \approx \hat{d}_G$
      • Find $M$ intermediate tasks $(\hat{s}_0^i, g^i)$ by selecting and intermediate goal $g^i$ from
        each $\tau^i$ according to (3.5)                              ▷ based on $d \approx \hat{d}_G$
8:      for episode = $1, M$ do
9:          $(s_0, g) \leftarrow (\hat{s}_0^i, g^i)$                     ▷ hindsight goal-oriented exploration
10:         for $t = 0, T-1$ do
11:             Sample an action $a_t$ using the policy from $\mathbb{A}$ with noise:
                        $a_t \leftarrow \pi(s_t \parallel g) + \mathcal{N}_t$                        (3.17)
12:             Execute the action $a_t$ and observe a new state $s_{t+1}$
13:         for $t = 0, T-1$ do
14:             $r_t := r_g(s_t, a_t)$
15:             Store transition $(s_t \parallel g,\ a_t,\ r_t,\ s_{t+1} \parallel g)$ in $R$   ▷ DDPG experience replay
16:             Sample a set of additional goals for replay $G := \mathbb{S}(current\ episode)$
17:             for $g' \in G$ do
18:                 $r' := r_{g'}(s_t, a_t)$
19:                 Store the transition $(s_t \parallel g',\ a_t,\ r',\ s_{t+1} \parallel g')$ in $R$        ▷ HER
20:     for $t = 1, N$ do
21:         Sample a minibatch $B$ from the replay buffer $R$                 ▷ HER or EBP
22:         Perform one step of optimization using $\mathbb{A}$ and minibatch $B$         ▷ DDPG:
            (2.5), (2.7), (2.8), (2.9)
```

3.3. Stop Condition

The basic idea of HGG / G-HGG is to guide exploration by means of hindsight goals. This is especially useful in the beginning of training, where hindsight replay goals (derived from previously achieved states) are far away from target goals. When training progresses and a certain fraction $\delta_{stop} \in [0,1]$ of the hindsight goal candidates for exploration are very close to sampled target goals $g \sim \mathcal{G}_T$, stopping hindsight goal generation and continuing training with vanilla HER often leads to faster learning and higher success rates.

In order to decide when to stop HGG / G-HGG, we can perform the HGG / G-HGG Stop Condition Check (algorithm 5) after step 5 of HGG (algorithm 3) and step 7 of G-HGG (algorithm 4), respectively. Even though HGG and G-HGG perform reasonably well without stopping ($\delta_{stop} = 1$) and switching to vanilla HER based on the Stop Condition Check, our findings show that a suitable choice of δ_{stop} can increase training performance whilst reducing computation time significantly. An ablation study on δ_{stop} can be found in the appendix (chapter C).

Note that for algorithm 5, the mapping m is not required to be invertible as long as m^{-1} is not computed explicitly. This is the case for goal predicates f_g similar to the one defined in (3.1), which can be easily computed without even using m:

$$f_g(s) = f_g(m^{-1}(g')) = \begin{cases} 1, & \text{if } \parallel g - m(m^{-1}(g')) \parallel = \parallel g - g' \parallel \leq \delta_g \\ 0, & \text{otherwise} \end{cases} . \tag{3.18}$$

Algorithm 5 HGG / G-HGG Stop Condition Check

```
1: Given:
     • A set of sampled target tasks $\{(\hat{s}_0^i, \hat{g}^i)\}_{i=1}^K \sim \mathcal{T}^*$
     • A set of intermediate tasks $\{(\hat{s}_0^j, g^j)\}_{j=1}^M$ selected by HGG / G-HGG
     • A stop condition parameter $\delta_{stop} \in [0, 1]$
     • A goal predicate $f_g$ similar to (3.1) with a traceable mapping $m : \mathcal{S} \to \mathcal{G}$
2: $c \leftarrow 0$
3: for $j = 1, M$ do
4:     $b \leftarrow 0$
5:     for $i = 1, K$ do
6:         if $f_{\hat{g}^i}(m^{-1}(g^j)) == 1$ then
7:             $b \leftarrow 1$
8:     $c += b$
9: if $\frac{c}{M} \geq \delta_{stop}$ then
10:     Stop condition is satisfied $\to$ Continue with vanilla HER.
11: else
12:     Stop condition is not satisfied $\to$ Continue with HGG / G-HGG.
```

4. Experiments

In this chapter, we compare the performance of G-HGG to HGG and vanilla HER in four challenging object-manipulation environments, which are modified versions of the Fetch gripper environments provided by Plappert, Andrychowicz, et al. (2018). Environments follow the structure proposed by OpenAI Gym (Brockman et al. 2016) and use the MuJoCo (Todorov, Erez, and Tassa 2012) physics engine for simulation. The RL problems in each of these environment show the characteristics specified in section 3.1. Three of the environments exhibit obstacles, which makes the underlying object manipulation task especially challenging to solve.

4.1. Environments

All our environments are MuJoCo environments featuring a modeled Fetch robot with a gripper. They are inspired by the environments provided by Plappert, Andrychowicz, et al. (2018) and thus adopt the their control strategy:

- The state space $\mathcal{S}$ contains joint positions and joint velocities for all joints of Fetch, the position of the end effector, as well as the object's position and orientation.
- Depending on whether gripper control is enabled or disabled, the action space is three- or four-dimensional. An action consists of the end effector's position for the next time step (three coordinates), in case of enabled gripper control, the gripper's opening control parameter is added as a forth component.
- The end effector's position of the Fetch robot is controlled via inverse kinematics and motion capturing. While the orientation of the end effector is fixed, the coordinates of its position at a certain time step can be explicitly enforced via input coordinates. Interaction between environment and RL agent takes place every time step $t \in [0, T-1]$ through one iteration of a feedback loop: Based on the observation $s_t \in \mathcal{S}$ at time step t and the current policy, the agent chooses an action $a_t \in \mathcal{A}$. In our case, the action denotes the end effector's desired position coordinates and the gripper control parameter at time step $t+1$. When the action is fed into MuJoCo, the physics simulator sets the new position of the end effector

to the desired value. The movement, joint positions, and joint velocities required to move the end effector from one position to the next are automatically calculated by MuJoCo's inverse kinematics algorithm. At $t + 1$, a new observation s_{t+1} is provided to the RL agent as a new iteration of the feedback loop begins.

Please note that this control strategy was simply adopted from Plappert, Andrychowicz, et al. (2018) and is not subject of this thesis. While it remains an open question to what extent such a control strategy is feasible in real-world robot control, G-HGG is independent of the chosen action and observation space and therefore applicable to real-world robot object manipulation. Parameters used for training in each of the environments can be found in sections A.1 and B.2.

4.1.1. FetchPushLabyrinth

In the FetchPushLabyrinth environment (Figure 4.1), the task is to push the object (black cube) from its initial position (sampled from $\mathcal{S}_0$, which is uniform in the square of length 0.12 between the green balls, and zero otherwise) around the blue obstacles (labyrinth) to a target goal (sampled from the target goal distribution $\mathcal{G}_T$, which is uniform in the square of length 0.12 between the blue balls, and zero otherwise). The gripper remains permanently closed and gripper control for picking up the object is disabled, leading to a three-dimensional action space (end effector coordinates in next time step only). The accessible goal space $\mathcal{G}_A$ is visualized by the semi-transparent blue box of length 0.5, width 0.7, and height 0.2, excluding the blue obstacle labyrinth. Note that it is crucial to define $\mathcal{G}_A$ properly: if the accessible goal space contained space above the labyrinth, the shortest path would require lifting the object over the labyrinth and not pushing it around the obstacles. Since the gripper is permanently closed, such paths are impossible to achieve in this environment and G-HGG would fail.

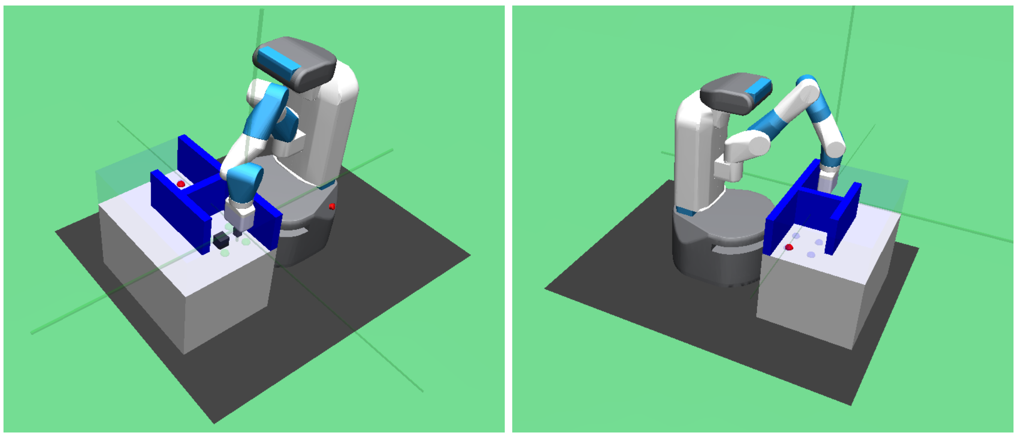


Figure 4.1.: FetchPushLabyrinth environment

4.1.2. FetchPickObstacle

In the FetchPickObstacle environment (Figure 4.2), the task is to pick up the object (black cube) from its initial position (sampled from $\mathcal{S}_0$, which is uniform in the square of length 0.12 between the green balls, and zero otherwise), lift it over the blue obstacle, and place it at a target goal position (sampled from the target goal distribution $\mathcal{G}_T$, which is uniform in the rectangle of length 0.4 and width 0.2 between the blue balls, and zero otherwise). Gripper control is enabled, the gripper can be symmetrically opened and closed via a single actuator, leading to a four-dimensional action space (end effector coordinates in next time step and gripper control parameter). The accessible goal space $\mathcal{G}_A$ is visualized by the semi-transparent blue box of length 0.5, width 0.7, and height 0.4, excluding the blue obstacle.

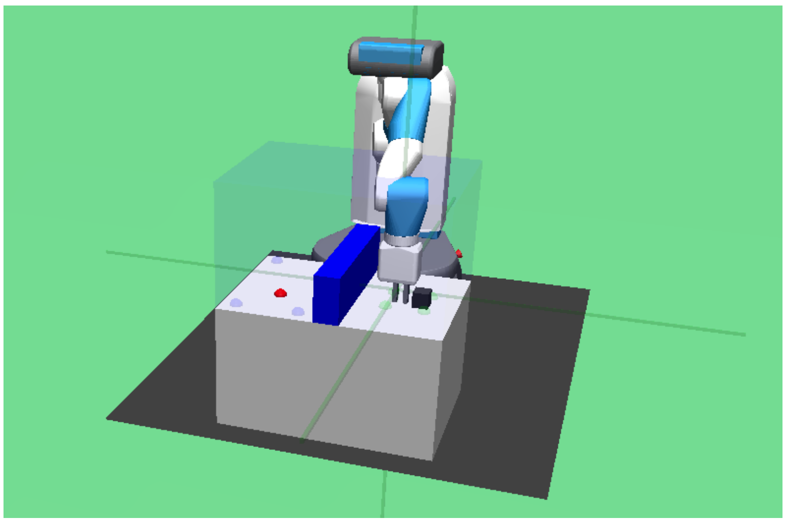


Figure 4.2.: FetchPickObstacle environment

4.1.3. FetchPickNoObstacle

In the FetchPickNoObstacle environment (Figure 4.2), the task is to pick up the object (black cube) from its initial position (sampled from $\mathcal{S}_0$, which is uniform in the square of length 0.12 between the green balls, and zero otherwise), lift it up, and place it at a target goal position (sampled from the target goal distribution $\mathcal{G}_T$, which is uniform in the rectangle of length 0.4 and width 0.2 between the blue balls, and zero otherwise). No obstacle is present in this scenario, but the target goals are located in the air at a height of 0.3 above the table. Gripper control is enabled, the gripper can be symmetrically opened and closed by a single actuator, leading to a four-dimensional action space (end effector coordinates in next time step and gripper control parameter). The accessible goal space $\mathcal{G}_A$ is visualized by the semi-transparent blue box of length 0.5, width 0.7, and height 0.4.

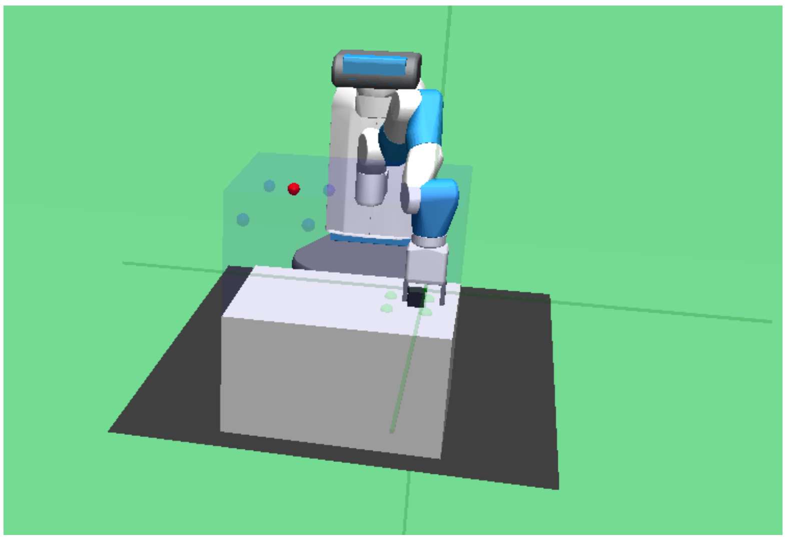


Figure 4.3.: FetchPickNoObstacle environment

4.1.4. FetchPickAndThrow

In the FetchPickAndThrow environment (Figure 4.4), the task is to pick up the object (black cube) from its initial position (sampled from $\mathcal{S}_0$, which is uniform in the square of length 0.12 between the green balls, and zero otherwise), lift it up, and throw it into one of the eight blue boxes (obstacles) (target goals are sampled from the target goal distribution $\mathcal{G}_T$, which is uniform in positions of the eight blue balls). Gripper control is enabled, the gripper can be symmetrically opened and closed by a single actuator, leading to a four-dimensional action space (end effector coordinates in next time step and gripper control parameter). The accessible goal space $\mathcal{G}_A$ is visualized by the semi-transparent blue box of length 0.85, width 0.7, and height 0.4 excluding the the blue walls of the eight obstacle boxes.

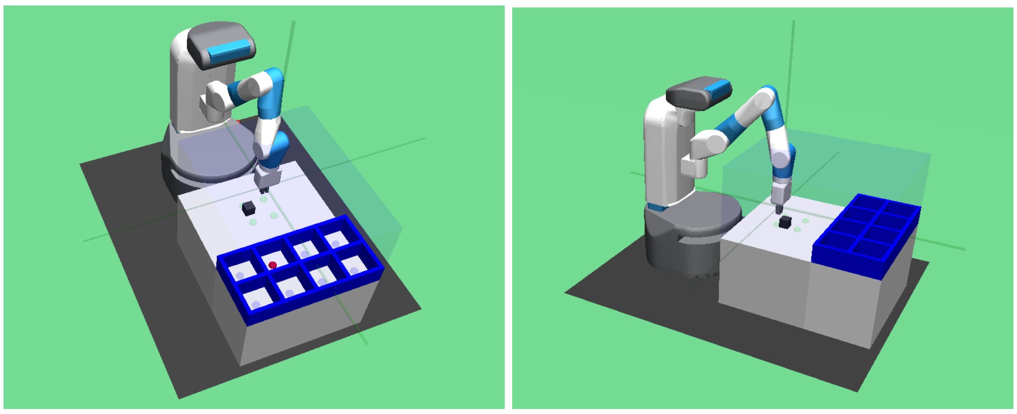
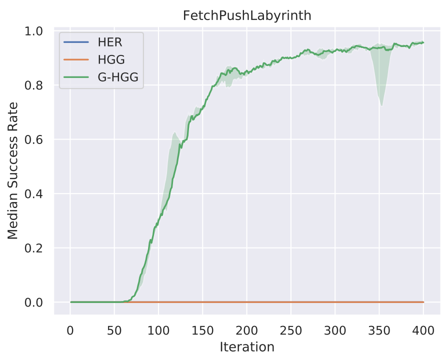


Figure 4.4.: FetchPickAndThrow environment

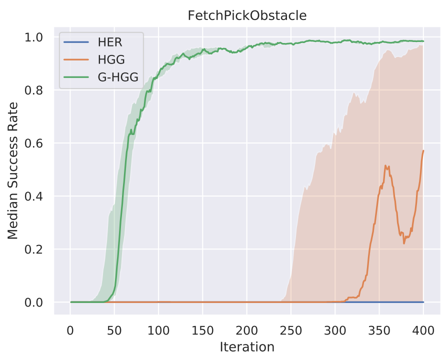
4.2. Results

We have tested G-HGG on the four environments from section 4.1 in order to compare its performance to HGG and HER. DDPG is used as off-policy RL algorithm in all trainings. Since we know from Zhao and Tresp (2018) and Ren et al. (2019) that EBP significantly increases performance in both HER and HGG, we used EBP in all our training runs of HER, HGG and G-HGG. The results presented in Figure 4.5 show that G-HGG outperforms HGG by far in environments with obstacles, both in terms of sample efficiency and maximum success rate. In the environment without obstacles, performance of G-HGG is comparable to HGG. Each plot in Figure 4.5 shows success rates of G-HGG, HGG, and HER (median and interquartile range of five training runs each) plotted over training iterations in our four testing environments. For an overview of computation times, hyper-parameters, and other implementation details, please refer to the appendix (chapters A and B).

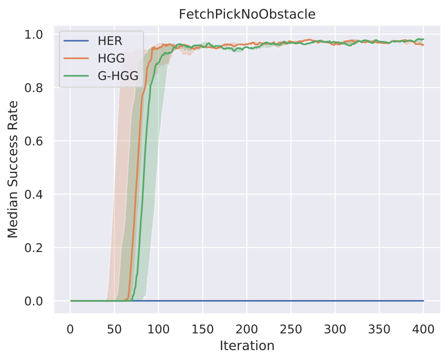
FetchPushLabyrinth: The most remarkable result can be observed in the FetchPush-Labyrinth experiment (see Figure 4.5a). While vanilla HER and HGG display no success over 400 training iterations, G-HGG reaches a success rate of 80% after 170 iterations, increasing to over 90% after 300 iterations. By comparing a sample of hindsight goals from iterations 20, 40, 60, and 80 (Figure 4.6), it becomes obvious why G-HGG outperforms HGG by far. While the graph-based distance measure used in G-HGG leads to a curriculum of hindsight goals guiding the agent around the obstacle towards the target goals (Figure 4.6a), HGG repeatedly chooses goals that are closest to the target goals with respect to the euclidean norm (Figure 4.6b). Since this euclidean norm based shortest path is blocked by an obstacle, the agent gets stuck trying to reach



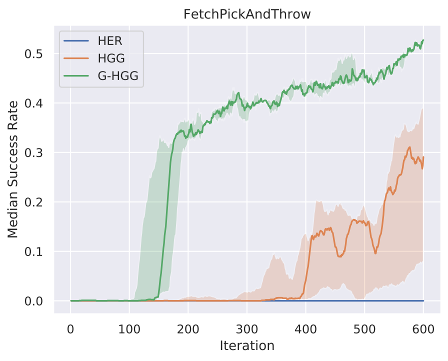
(a) FetchPushLabyrinth



(b) FetchPickObstacle



(c) FetchPickNoObstacle

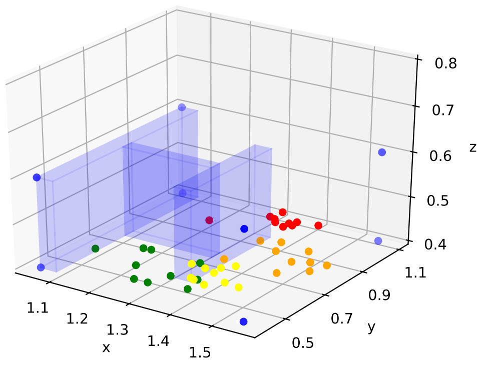


(d) FetchPickAndThrow

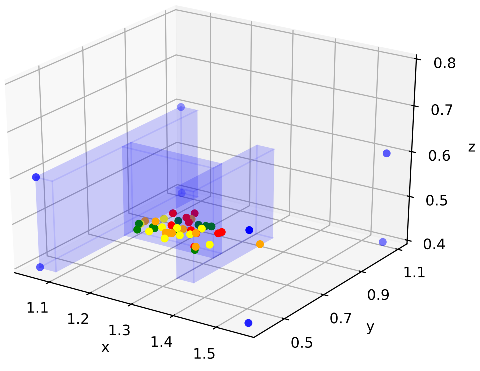
Figure 4.5.: Median test success rate (line) and interquartile range (shaded) of G-HGG, HGG, and HER in different environments. One iteration contains $M = 50$ episodes.

the hindsight goals (pushing the object against the obstacle), never achieving more valuable goals and thus never reaching the target goals. Note that G-HGG meets the stop criterion ($\delta_{stop} = 0.3$) after 70 iterations, thus training continues with vanilla HER from that point on.

FetchPickObstacle: In the FetchPickObstacle task, G-HGG clearly outperforms HGG in terms of sample efficiency (Figure 4.5b), due to graph-based distances generating obstacle-avoiding curricula of hindsight goals. Since the euclidean norm as distance measure is at least partly (in x- and y-direction) valid in this environment, HGG eventually achieves a notable success rate as well. However, the plots show that there is a large variance within the HGG trainings, caused by a sub-optimal curriculum of hindsight goals in HGG that is not able to efficiently guide exploration towards target



(a) G-HGG



(b) HGG

Figure 4.6.: Hindsight goals in FetchPushLabyrinth after 20 (red), 40 (orange), 60 (yellow) and 80 (green) iterations of HGG / G-HGG. One iteration contains $M = 50$ episodes.

goals. Vanilla HER is not able to solve the task. Note that G-HGG meets the stop criterion ($\delta_{stop} = 0.3$) after 50 iterations, thus training continues with vanilla HER from that point on.

FetchPickNoObstacle: FetchPickNoObstacle is an environment where G-HGG has no advantage over HGG. As no obstacle is present, the euclidean norm is applicable to compare distances, allowing HGG to perform well in this task. Taking into account that the euclidean norm used in HGG is more exact than the graph-based distances in G-HGG, it is not surprising that HGG yields slightly better training results than G-HGG. However, the similarity of the curves in Figure 4.5c shows that G-HGG's performance is comparable to HGG in terms of sample efficiency and maximum success rate even in environments without obstacles. Generalizing from these findings, we assume that G-HGG is applicable to all environments where HGG yields good training results, provided that the criteria defined in section 3.1 are met. Just like in the other environments, vanilla HER fails to reach any of the target goals. Note that G-HGG meets the stop criterion ($\delta_{stop} = 0.3$) after 70 iterations, thus training continues with vanilla HER from that point on.

FetchPickAndThrow: FetchPickAndThrow is a special task, since target goals are not uniformly sampled from a continuous target goal distribution $\mathcal{G}_T$, but from a discrete set of eight goals. This makes learning more difficult and a higher value of δ_{stop} is preferable. G-HGG yields better results than HGG in terms of maximum success rate after 600 iterations and is clearly more sample efficient. Both HGG and G-HGG cannot achieve success rates above 60% due to the difficulty of the throwing task, involving picking the object, lifting it up, and dropping it while giving it a well-dosed push in the desired direction. Note that neither G-HGG nor HGG meet the stop criterion ($\delta_{stop} = 0.9$). Since the final policy trained to achieve the task results from guided, but still random exploration, it is no surprise that the agent cannot develop a perfect throwing motion from sparse rewards. When looking more closely at the results and the robot's performance, some aspects can be identified as causes for the relatively low success rate of about 50%. At first, the agent learns very well how to throw the object into the boxes in the first row that are closer to the initial object states, while it struggles hitting boxes in the second row. Understandably, throwing an object further requires more fine-tuning and is therefore more difficult to achieve. Secondly, the agent throws the object at least partly by closing its gripper joint and squeezing the elastic cube out. Even though this strategy works well for hitting close target goals, it becomes impractical when the object must be thrown further. Moreover, this strategy does not work with less elastic objects. A higher throwing success rate could be achieved by changing the task configuration in order to prevent the development of such a sub-optimal strategy. Reduced elasticity of the cube and decelerated joint control (maybe without motion capture and inverse kinematics) could potentially lead to learning a better throwing strategy involving swinging of the entire robotic arm. It would be interesting to investigate, how G-HGG can be applied to robotic throwing tasks more successfully.

5. Discussion

After testing our G-HGG algorithm in four challenging object manipulation environments and comparing its performance to HGG and HER, we have discussed the experimental results in section 4.2. In this chapter, we conduct a more general analysis of our G-HGG algorithm, focusing on applicability, advantages, and limitations of our proposed method.

5.1. Applicability of G-HGG

In this thesis we have demonstrated successful application of G-HGG to object manipulation problems showing the characteristics listed in section 3.1. However, G-HGG stays applicable when some of these characteristics are relaxed or generalized. Most importantly, the use of G-HGG is not limited to object manipulation tasks as long as the following conditions are met:

1. **Action space:** Applicability of G-HGG requires the definition of an action space $\mathcal{A} \subset \mathbb{R}^l$, $l \in \mathbb{N}$.
2. **State space:** Applicability of G-HGG requires the definition of a state space $\mathcal{S} \subset \mathbb{R}^n$, $n \in \mathbb{N}$. In particular, n is not limited to values $n > 3$, G-HGG is even applicable in one-dimensional tasks.
3. **Initial state distribution:** Applicability of G-HGG requires an initial state distribution $\mathcal{S}_0 : \mathcal{S} \to \mathbb{R}$. Initial states should be reasonable starting states for a task.
4. **Goal space:** Applicability of G-HGG requires the definition of a goal space $\mathcal{G} \subset \mathbb{R}^q$, $q \in \mathbb{N}$. In particular, q is not limited to $q = 3$ as in our three-dimensional object manipulation tasks. However, tasks with $q \neq 3$ demand a graph construction procedure different from the one described in section 3.2.1. Especially connecting vertices in high-dimensional goal spaces is very challenging and requires redefinition of (3.11) and related. Further examination of G-HGG's applicability to higher-dimensional goal-spaces could be subject to future research.

5. **Target goal distribution:** Applicability of G-HGG requires the definition of a target goal distribution $\mathcal{G}_T : \mathcal{G} \to \mathbb{R}$.
6. **Goal predicate:** Applicability of G-HGG requires a goal predicate $f_g : \mathcal{S} \to \{0, 1\}$, $g \in \mathcal{G}$ returning one if and only if a goal is reached, as well as a known and traceable mapping between state space and goal space $m : \mathcal{S} \to \mathcal{G}$. The definition of f_g and m can be fundamentally different from our choice (3.1).
7. **Sparse reward function:** Applicability of G-HGG requires a reward function $r_g : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. Even though G-HGG might yield satisfying training results in dense reward settings as well, it was, just as HGG and HER, explicitly designed for sparse-reward RL. It is therefore strongly recommended to define a sparse reward function such that the agent receives a non-negative reward if and only if it has reached the goal, e.g. $r_g(s, a) = -[f_g(s) == 0]$.
8. **Knowledge about the environment:** Applicability of G-HGG requires knowledge about the geometric properties of the environment. In particular, the exact locations of all goals that are not reachable must be known. Constraining so-called obstacles to continuous, bounded, and convex sets of goals $g_{obstacle} \in \mathcal{G}$ that are not physically reachable is only required in our object manipulation tasks. In general, it is sufficient to know which goals are not reachable in order to define an accessible goal space $\mathcal{G}_A \subset \mathcal{G}$. However, a redefinition of the vertex density criterion (3.13) might be necessary.

5.2. Advantages of G-HGG

Extended use-cases compared to HGG: Obviously, the biggest advance of G-HGG over HGG is its applicability to environments with obstacles, where the euclidean norm is not an accurate distance measure. In section 4.2, we demonstrated that G-HGG yields high success rates in environments where HGG cannot learn a reliable policy. Moreover, G-HGG training results on the FetchPickNoObstacle environment (Figure 4.5c) show no significant decrease in learning performance compared to HGG. We therefore assume that G-HGG is applicable to the same environments as HGG under the condition that the criteria defined in 5.1 are met. Just as HGG, G-HGG demonstrates training performance that is largely superior to HER (even with EBP) in environments, where target goals are far from the object's initial position. Our outstanding experimental results support the choice of using a graph-based distance measure in environments with obstacles despite its limitations regarding exactness (see section 5.3).

Straight-forward application: Another significant advantage of G-HGG is its easy and straight-forward application to object manipulation environments. G-HGG inherits its applicability to sparse-reward RL from HER and HGG. Therefore, G-HGG can easily be implemented for different task settings, as no time-consuming shaping of the reward-function is required. Apart from choosing a suitable action space, state space, goals space, reward function, and suitable hyper-parameters, the only important input expected from the user is information about the environment and its potential obstacles. For graph construction, G-HGG requires the definition of the accessible goal space $\mathcal{G}_A \subset \mathcal{G}$, representing the environment of the RL task at hand. As long as the environment is known, a definition of $\mathcal{G}_A$ can be trivially derived from its geometric and material properties. The user does not need to define and hand-craft a suitable distance metric for each new environment. Instead, computation of shortest distances is fully automated as long as $\mathcal{G}_A$ is provided. Moreover, we show in the appendix (chapter C) that G-HGG is, similar to HGG, robust to changes of its hyper-parameters (in particular n_x, n_y, n_z, δ_{stop}). Summarizing all these aspects, G-HGG is easy to apply to new object manipulation environments without fine-tuning or input pre-processing.

Pre-computation: Another strong point of G-HGG is efficient pre-computation of the discrete distance values $\hat{d}_G$. Replacing an easy-to-compute euclidean distance measure with a graph-based shortest path distance measure inevitably comes at the cost of increased computation time (see section 5.3). Since distances between goals are computed $5 \cdot 10^6$ times in each episode of HGG / G-HGG, even little increase in distance computation time has a major effect on overall training time. In G-HGG, this problem is solved efficiently by pre-computing shortest distances $\hat{d}_G$ between every pair of vertices $(v_1, v_2) \in V^2$ on graph $G = (V, E)$ and creating an $n \times n$ table of distance values, where n denotes the number of vertices in the graph. Even for large numbers of vertices at the limit of storage capacity (section 5.3), pre-computation time is negligible compared to overall training time. In our experiments on the FetchPickAndThrow task with a large number of $n = 18207$ vertices, pre-computing $n^2 \approx 331 \cdot 10^6$ shortest distances takes less than three minutes, which is negligible compared to an overall training time of several hours.

5.3. Drawbacks of G-HGG

Inexactness: The most obvious problem of using graph-based distances in G-HGG instead of euclidean norm based distances in HGG is their inexactness. While the euclidean norm is always equal to the shortest distance between two goals in environments without obstacles, this is not the case for the graph-based distance in environments with obstacles. It is easy to see from the definition of the graph and from the given examples, that there is a potential error between the graph-based distance and the real shortest obstacle-avoiding distance. Unfortunately, this error is not bounded by a constant, but is, in the worst case, linearly proportional to the length of the exact obstacle-avoiding shortest path. Consequently, no guarantee whatsoever can be given with respect to the accuracy of the distance measure. However, despite its inexactness, the graph-based distance measure used in G-HGG proved to be very successful in practice, which can be attributed to the fact that the real errors are a lot smaller than the worst-case errors, since small inaccuracies often even out. Moreover, at least the constant part of the error (i.e. the distance between the goal and the closest vertex) converges to zero with an increasing number of vertices n. Finally, it is important to point out that the distance measure is only used to choose valuable hindsight goals, a process involving more parameters and restrictions subject to inexactness. Just like HGG has proven to be robust to changes in its parameters such as Lipschitz constant L and distance weight c (2.22) (Ren et al. 2019), G-HGG is robust to errors in distances between goals. In the worst case, a wrongly calculated distance leads to the replay of a transition with a non-valuable hindsight goal, which, apart from consuming training time, has no negative effect on learning an optimal policy. As long as part of the selected hindsight goals are valuable due to correct distance computation, training progresses and G-HGG yields better success rates than HGG with its euclidean distance.

Distance computation time: One major drawback of G-HGG compared to HGG is the increased distance computation time. While distances between two goals are directly calculated with the euclidean norm in HGG, values of the shortest distance function table $\hat{d}_G$ for each combination of two vertices in the graph are pre-computed in G-HGG. However, even with pre-computation, every graph-based distance calculation requires mapping the goals to the closest vertices in V (3.16) and then accessing the corresponding entries of the shortest distance function table (3.15). Although both processes have very small execution times, our implementation of G-HGG shows that calculating graph-based distances using the pre-computed shortest distance function values takes slightly more time than simply computing the euclidean norm based distances. When running our implementation of G-HGG on our hardware, calculating a graph-based distance takes about $2 \cdot 10^{-5}$ seconds longer than calculating a euclidean norm based distance. That small difference in computation time becomes relevant when looking at how often distances are calculated. In each iteration (containing $M = 50$ episodes) of HGG / G-HGG, $K \cdot T \cdot 10^3 = 50 \cdot 100 \cdot 10^3 = 5 \cdot 10^6$ distances are computed. K denotes the number of hindsight goals, T is the number of time steps per episode and 10^3 the number of potential hindsight goals from the replay buffer. Consequently, on our hardware, each iteration of G-HGG takes roughly 100 seconds longer than the corresponding iteration of HGG. An exemplary visualization of computation time per iteration in the FetchPickAndThrow task with $\delta_{stop} = 0.3$ is provided in Figure 5.1. In the beginning of training, G-HGG takes about 175 seconds per iteration, while HGG and HER take between 60 and 65 seconds. After roughly 110 iterations, G-HGG satisfies the stop criterion (algorithm 5) with $\delta_{stop} = 0.3$, resulting in a drop in computation time to the level of HER, since G-HGG is aborted and training continues with vanilla HER. After iteration 350, a slight drop in HGG computation time can be observed, caused by HGG meeting the stop condition as well. While an increase in computation time is an undeniable drawback of G-HGG, it is overshadowed by G-HGG's improved sample efficiency in many environments. Since the stop condition is often satisfied early in G-HGG, the increase in computation time lasts only for a limited number of training iterations.

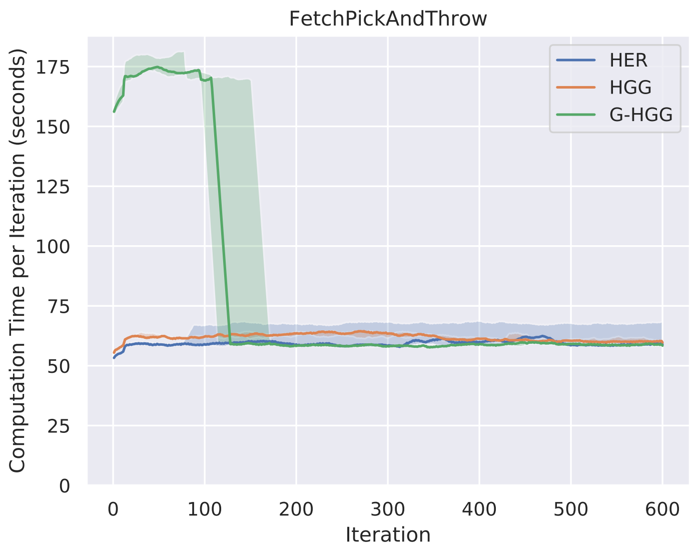


Figure 5.1.: Median (line) and interquartile range (shaded) of computation time per iteration of G-HGG, HGG, and HER in FetchPickAndThrow with $\delta_{stop} = 0.3$. One iteration contains $M = 50$ episodes. G-HGG satisfies the stop criterion at iteration 110, HGG satisfies the stop criterion at iteration 350.

Scalability: Another important issue of G-HGG is its limited scalability due to storage requirements for the pre-computed shortest distance function table $\hat{d}_G$. The size of this table is $n \times n$, depending quadratically on the number of vertices V in graph G. In our environment FetchPickAndThrow, for example, the number of vertices $n = n_x \cdot n_y \cdot n_z = 51 \cdot 51 \cdot 7 = 18207$ results in $n^2 \approx 331 \cdot 10^6$ table entries of eight bytes each, occupying 2.47 GB (gigabytes) of RAM. This example illustrates that as long RAM is limited to e.g. 16 GB, the number of vertices n cannot be arbitrarily increased. This is especially problematic in large environments with small obstacles, where the upper limit on the number of vertices n conflicts with a lower limit on n imposed by the vertex density criterion (3.13).

6. Related Research

Since Andrychowicz et al. (2017) have published Hindsight Experience Replay (HER), a notable amount of research has focused on extending and improving the sparse-reward RL algorithm. HER has been successfully extended to dynamic goal pursuit (Fang, C. Zhou, et al. 2019) and vision-based robotic tasks. In visual tasks, goal states are characterized by the presence of a distinct visual feature, which must be recognized. Nair, Pong, et al. (2018) solve such tasks by combining multi-goal RL with unsupervised representation learning, while Sahni et al. (2019) re-actively hallucinate goals into the agent's observations with a pre-trained generative model. Moreover, it has been shown that HER can be used in combination with hierarchical RL (Levy et al. 2019) and on-policy RL (Rauber et al. 2019). In their Competitive Experience Replay algorithm, Liu et al. (2019) introduce competition between two agents for better exploration.

6.1. Prioritized Experience Replay

One major drawback of HER has been its rather inefficient random replay of experience. Research has shown that prioritized sampling of transitions from the replay buffer significantly increases sample efficiency. Prioritized sampling can be based on the TD-error (Schaul, Quan, et al. 2016), reward-weighted entropy (Zhao, Sun, and Tresp 2019), transition energy (Zhao and Tresp 2018), and density of achieved goals (Zhao and Tresp 2019). Curriculum-Based Experience Replay (CHER) introduced by Fang, T. Zhou, et al. (2019) prioritizes replay buffer entries according to their diversity with respect to other replay goals and their proximity to target goals. This approach is very similar to HGG and is discussed in detail in section 6.3.

6.2. Demonstrations for Improved Exploration

Exploration is another challenging problem in sparse-reward RL algorithms such as HER. In scenarios with a high-dimensional action space and a large task horizon (e.g. far-away goals), finding a non-negative reward can be very difficult. Nair, McGrew, et al. (2018) use demonstrations to tackle this issue and learn challenging multi-step tasks such as object stacking. More concretely, their approach combines multi-goal RL with imitation learning by introducing a demonstration replay buffer and a behavioral cloning loss. Due to the high complexity of our FetchPickAndThrow task, combining HER with demonstrations might lead to better results and a more effective throwing motion than G-HGG.

6.3. Curriculum Learning

Another way to tackle the exploration problem in sparse-reward multi-goal RL is curriculum learning, which presents problems in a favorable order, a so-called curriculum. The motivation behind curriculum learning is to improve exploration and learning success by solving easier problems first, which then help solving more complex problems later. The most important challenge in curriculum learning is creating goals of appropriate difficulty, which is tackled in different ways.

One curriculum learning approach to improve exploration is augmenting the sparse learning problem with basic, easy-to-learn auxiliary tasks (Riedmiller et al. 2018). The agent actively schedules auxiliary tasks according to a pre-defined or learned curriculum and executes these sub-tasks to explore its observation space whilst searching for sparse rewards of externally defined target tasks. This approach can be interpreted as a hierarchical extension of HER, where, instead of exploring by following random goals, the agent tries to learn the auxiliary goals first. Eventually, the auxiliary tasks enable the agent to learn the complex target tasks. An algorithm for automated, domain-independent generation of such sub-tasks and their corresponding sub-goals was introduced by Eppe, Magg, and Wermter (2019). The authors create sub-goals by masking certain characteristics of a goal. For example, masking the z-coordinate of goals in a pick-and-place task yields a sub-goal only consisting of x- and y-coordinates, which is easier to achieve. Although their approach has shown improved performance compared to HER in the benchmark pick-and-place scenario, it remains questionable whether the use of goal masking is beneficial, since it requires extensive hyper-parameter tuning and is very likely to be outperformed by EBP or HGG.

In the absence of extrinsic motivation due to the sparsity of external rewards, intrinsic motivation can be used to create a curriculum for improved exploration. In intrinsically motivated RL, the agent develops a large, mostly hierarchical set of skills by reacting curiously to interesting events due to an intrinsic reward. Intrinsic rewards can be based on measures such as novelty, predictive information gain, or learning progress (Singh, Barto, and Chentanez 2005, Pathak et al. 2017). Most notably, Forestier, Mollard, and

Oudeyer (2017) use intrinsic motivation to provide agents with a mechanism to sample goals from a space of parametrized goals and thus automatically generate their own curriculum. Péré et al. (2018) go one step further and introduce an algorithm capable of learning a goal space from raw sensor observations. Another idea for automatic curriculum learning was presented by Sukhbaatar et al. (2018), where two agents are generating goals for each other in intrinsically motivated self-play. While intrinsic motivation is particularly useful in solving a family of problems with potentially different or unknown goal spaces and reward functions, it can be applied for curriculum generation in rather low-level multi-goal object manipulation scenarios as well. Colas et al. (2019), for example, have used intrinsically motivated RL to successfully learn four Fetch object manipulation tasks at once with a single policy. Aubret, Matignon, and Hassas (2019) give a more detailed overview about intrinsically motivated reinforcement learning and its use in curriculum learning.

Another way of constructing a meaningful curriculum is to predict high-reward states and generate goals close to these meaningful states (Goyal et al. 2019). However, this methods requires at least some high-reward states to be easily reachable or known.

Florensa, Held, Geng, et al. (2018) have recently proposed a curriculum learning algorithm, in which a generative adversarial network is trained to produce goals of intermediate difficulty (GOID). The authors use the training success rate of previous time steps for the same goal as a measure for difficulty. Ren et al. (2019) have combined GOID with HER and compared its training performance to their own HGG algorithm on a FetchPush task. HGG demonstrated better sample efficiency than GOID+HER, which showed only minimal improvement over vanilla HER. Inverse curriculum generation (Florensa, Held, Wulfmeier, et al. 2017) shows that curriculum learning also works reversely, when the agent gradually learns to reach the goal from a curriculum of start states chosen to be increasingly far from the goal.

All above mentioned curriculum learning algorithms have demonstrated to improve exploration in sparse multi-goal RL. However, they share one significant drawback: exploration is unguided. Consider a scenario with a very large goal space, where target goals are far from the goal representation of the initial states. Even when following a curriculum based on auxiliary tasks, novelty, information gain, previous rewards, or training success, the agent has to gradually explore the entire goal space until a target goal is reached for the first time. When target goals are "hidden", e.g. by obstacles as in FetchPushLabyrinth, this unguided exploration process takes an unreasonably long time.

Ren et al. (2019) have chosen a distance-based approach to tackle the problem of exploration in sparse-reward object manipulation. By generating hindsight goals, HGG provides a curriculum for training that guides the agent towards reaching target goals. In contrast to the other approaches presented in this chapter, HGG creates goals of appropriate difficulty based on their distance to target goals. Thus, the agent does not explore the goal space uniformly, but is directly guided towards the target goals, making exploration very efficient.

In that aspect, Curriculum-guided hindsight experience replay (CHER) (Fang, T. Zhou, et al. 2019) is very similar to HGG. In CHER, hindsight experiences from the replay buffer are selected based on proximity of the hindsight goal to a target goal and the diversity of the hindsight goal compared to other goals selected for replay. To evaluate the proximity of two goals, both CHER and HGG use a distance measure, e.g. the euclidean norm. Despite their similarity, CHER and HGG show two major differences. First, CHER has an increased focus on curiosity-driven exploration, as it uses diversity of hindsight goals as a goal selection criterion, whereas HGG puts more attention to the expected return obtained with potential hindsight goals. Secondly, CHER prioritizes the sampling of transitions from the replay buffer according to proximity and diversity of the replayed hindsight goals, similar to EBP. HGG, on the other hand, uses hindsight goals for exploration and keeps the regular replay strategy of HER or EBP. Interestingly, despite the similarity of their approaches, HGG and CHER could hence be combined. Developing a holistic approach using hindsight goals for both exploration and replay would be an interesting direction for further research.

6.4. Distance Measure for Environments with Obstacles

To our knowledge, G-HGG and CHER should be the only currently available RL algorithms capable of solving our challenging sparse-reward object manipulation tasks in environments with obstacles. For CHER, however, this only holds under the assumption that a valid distance measure applicable to environments with obstacles is available. Since such distances cannot be trivially computed and no feasible obstacle-avoiding distance measure has been proposed by fellow researchers, we consider graph-based shortest distances from G-HGG as the best and probably only currently available distance measure applicable to this problem. For direct comparison of G-HGG to CHER in environments with obstacles, we suggest investigating the use of graph-based shortest distances in CHER as well. Alternative approaches for the computation of shortest path based distances have been investigated in section 3.1. Due to their large computational cost, most algorithms designed to solve the Euclidean Shortest Path problem are not applicable to HGG, where distances must be computed $5 \cdot 10^6$ times in each training iteration. Our pre-computable graph-based distance measure, on the other hand, is computationally inexpensive and has demonstrated great success on a variety of multi-goal object manipulation RL tasks.

A promising alternative to using shortest path based distances for distance measurement in environments with obstacles would be learned distances. Especially distances obtained via representation learning, a method transforming the state or goal space of an environment into a more convenient representation, are very promising. Srinivas et al. (2018) have applied representation learning to imitation learning tasks in environments with obstacles. Their proposed Universal Planning Network (UPN) architecture is able to learn plannable representations of states, which can be used as a task-specific latent distance measure to other states. Most notably, their learned distance metric is obstacle-aware, supports obstacle avoidance, and generalizes to new tasks with both new goals and new obstacle configurations. Even though their representation learning architecture was designed for imitation learning from demonstrations, the authors claim that it could also be combined with reinforcement learning. Another interesting representation learning architecture is Actionable Representations for Control (ARC) proposed by Ghosh, Gupta, and Levine (2019). The authors' method is based on the assumption that two goals are functionally different if and only if achieving them requires different actions. By comparing which actions a goal-conditioned RL policy takes to achieve different goals, ARC provides a state representation that is aware of the dynamic structure of the environment, including potential obstacles. Consequently, ARC can be used as a distance metric accounting for functional proximity and reachability to improve exploration in sparse-reward RL scenarios.

Both UPN and ARC are promising representation learning architectures for generating a distance metric that is applicable to environments with obstacles. Compared to shortest path based distances, distances generated by representation learning would have a significant advantage: no detailed knowledge about geometric properties of the environment is required. Therefore, following Ren et al. (2019), we strongly encourage researchers to incorporate UPN or ARC within HGG and evaluate training performance in our proposed environments.

7. Conclusion & Outlook

In this thesis, we have presented an extension of the HGG algorithm to robotic object manipulation tasks in environments with obstacles, where the euclidean norm is not applicable. In our Graph-Based Hindsight Goal Generation (G-HGG) algorithm, the euclidean distance measure between two goals used for the selection of valuable hindsight goals in HGG is replaced by a graph-based distance measure. Therefore, graph construction and shortest distance computation were introduced as pre-training computation steps. Experiments on four different object manipulation tasks with a MuJoCo-simulated robotic gripper arm demonstrated superior performance of G-HGG over vanilla HGG and HER in terms of both maximum success rate and sample efficiency. Future research could concentrate on the improvement, extension, and real-world deployment of G-HGG.

Improving G-HGG performance: It would be very interesting to investigate how to further increase the performance of G-HGG in sparse-reward robotic object manipulation tasks in environments with obstacles. For task-independent improvement, one could try to tackle the three main limitations of G-HGG: inexactness, increased distance computation time and lack of scalability (see section 5.3). Furthermore, performance could be increased by combining G-HGG with complementary strategies to improve HER. Possible compatible algorithms include, but are not limited to, prioritizing the transitions in the replay buffer (Zhao and Tresp 2018, Schaul, Quan, et al. 2016, Zhao, Sun, and Tresp 2019) or using parameter space noise for exploration (Plappert, Houthooft, et al. 2018). Further improvement of G-HGG could be inspired by work referenced in chapter 6. For more task-specific improvement, extensive parameter studies or changes in robot control may yield considerably better performance. From the list of RL-related hyper-parameters (see chapter B), increasing the number of MPI workers, changing batch size, (approximated) replay buffer size (Zhang and Sutton 2017), noise, or the constitution of the actor and critic networks seem most promising. However, better overall success rate might come at the cost of increasing computation time or resource usage. Finally, a change in task design could lead to better overall training success. We suspect the maximum success rate of FetchPickAndThrow being below 60% can be traced back to unrealistic material properties of the very elastic object and a questionable approach to control the robot via inverse kinematics and motion capturing (see chapter 4). We are well aware that G-HGG might not be the best strategy to solve the FetchPickAndThrow task, especially an approach combining HER with imitation learning and demonstrations (Nair, McGrew, et al. 2018) seems very promising.

Extending G-HGG to more diverse use-cases: In this thesis, G-HGG was designed for robotic object manipulation tasks with a three-dimensional goal space. Although the benefits of a graph-based distance measure are most obvious in 3-D environments with obstacles, we are positive that G-HGG could as well be applied to different tasks. One possible idea for future research would be to extend G-HGG to more general goal spaces $\mathcal{G} \subset \mathbb{R}^m, m > 3$, for example a six-dimensional goal space in object manipulation including both position and rotation of the object. Moreover, it would be very interesting to apply G-HGG to tasks other than object manipulation such as playing certain Atari games or navigating through a labyrinth. As long as the tasks are about reaching a goal from a target distribution over an accessible goal space $\mathcal{G}_A$, G-HGG is applicable. However, for complex scenarios with a high-dimensional goal space, defining an accessible goal space $\mathcal{G}_A$ and constructing a graph for distance computation is very challenging. Especially connecting vertices with weighted edges in higher dimensions is not as straight-forward as in 3-D space and requires a redefinition of equation 3.11. Another possible and less complicated future research topic would be the extension of G-HGG to environments with more difficult obstacle geometries. For example, supporting obstacles that are not convex would require a more sophisticated vertex density criterion (3.13) to make sure that objects are not overlooked.

Applying graph-based shortest distances in reward shaping: Another interesting proposal for future research would be to examine the applicability of graph-based shortest distances in object manipulation RL tasks with dense rewards. Instead of using G-HGG, HGG, or HER, object manipulation tasks could be solved by vanilla DDPG using graph-based shortest distances for reward shaping: A reward function $r_g : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ defined as $r_g(s,a) = -d(m(s),g)$ with $g \in \mathcal{G}$, a traceable mapping $m : \mathcal{S} \to \mathcal{G}$, and a graph-based shortest distance function d defined and computed according to subsections 3.2.1 and 3.2.2 could lead to successful learning of the tasks as well.

Applying graph-based shortest distances to CHER: In section 6.3, we have identified Curriculum-Guided Experience Replay (CHER) (Fang, T. Zhou, et al. 2019) as another RL algorithm capable of solving challenging sparse-reward object manipulation tasks in environments with obstacles, where target goals are far from the goal representation of initial states. Similar to HGG, CHER selects hindsight goals based on distances between achieved and target goals. A direction for future research could be to use graph-based shortest distances as described in sections 3.2.1 and 3.2.2 in CHER and compare training performance of the resulting Graph-Based Curriculum-Guided Experience Replay (G-CHER) algorithm to G-HGG.

Representation learning: As described in section 6.4, representation learning is a promising alternative approach for generating a distance metric that is applicable to environments with obstacles. It would be an interesting research topic to incorporate two representation learning architectures, namely Universal Planning Networks (Srinivas et al. 2018) and Actionable Representations for Control (Ghosh, Gupta, and Levine 2019) within HGG, test them in our proposed environments, and compare their training performance to G-HGG.

Deploying G-HGG to a physical robot: Finally, it would be an important advance to bridge the gap between theory and practice by deploying a policy learned with G-HGG to a physical robot. Successful sim-to-real transfer has been achieved for pick-and-place tasks with HER (Andrychowicz et al. 2017), where a convolutional neural network was separately trained on raw fetch head camera images from the MuJoCo renderer to predict the object's position. Similarly to the experiment performed by Andrychowicz et al. (2017), adding Gaussian noise to the observations for policy training might be required for the algorithm to be robust to small errors of the object's position. Dynamics randomization has proven to be a successful strategy for bridging the gap between simulation and real world (Peng et al. 2018). Despite exclusively training the policy in simulation, the authors were able to maintain a similar level of performance in an object pushing task performed by a real Fetch robot.

A. Experiment Settings

A.1. Environment Settings

For all environments presented in this thesis, the number of episodes per iteration of HER, HGG, or G-HGG is $M = 50$, with $T = 100$ time steps per episode. The distance threshold δ_g for successfully achieving a goal as defined in (3.1) is $\delta_g = 0.06$ in FetchPickAndThrow and $\delta_g = 0.05$ in all other environments. G-HGG parameters for each environment as defined in chapter 3 and used in chapter 4 are listed in table A.1.

Table A.1.: G-HGG parameters for different environments

Environment	n_x	n_y	n_z	δ_{stop}	n	Δ_x	Δ_y	Δ_z
FetchPushLabyrinth	31	31	11	0.3	10571	0.017	0.023	0.020
FetchPickObstacle	31	31	11	0.3	10571	0.017	0.023	0.040
FetchPickNoObstacle	31	31	11	0.3	10571	0.017	0.023	0.040
FetchPickAndThrow	51	51	7	0.9	18207	0.017	0.014	0.067

A.2. Evaluation Details

- All curves in this thesis are plotted from five runs with random task initializations and seeds.
- Shaded regions indicate 50% population around median.
- All curves are plotted using the same parameters described in sections A.1 and B.2, except for the ablation study (chapter C).

A.3. Overall Computation Time

In table A.2, we list the average computation times of our experimental runs from section 4.2. The experiments were run on hardware described in B.1. Note that computation time is highly dependent on hardware, implementation, and other factors. Therefore, the informative value of this table is limited to giving an impression of the order of magnitude and the relationship between computation times of different algorithms.

Table A.2.: Computation time (in hours) of experimental runs of G-HGG, HGG, and HER in different environments. Used parameters are described in chapters 4, A, and B. Hardware is described in section B.1.

Environment	HER	HGG	G-HGG
FetchPushLabyrinth	6.5	7	9
FetchPickObstacle	6.5	7	8
FetchPickNoObstacle	6.5	6.5	10
FetchPickAndThrow	10	11	29

B. Implementation Details

B.1. Hardware and Software

Experiments were performed on an octa-core machine with 16 GB of RAM. G-HGG was implemented in Tensorflow (version 1.14) and run on Ubuntu 16.04 with Python 3.5.2. Our G-HGG implementation is based on the HGG implementation provided by Ren et al. (2019). For source code and detailed information about the environment, please refer to our GitHub repository.[1]

B.2. Hyper-Parameters

Hyper-parameters for DDPG and HER are kept the same as in the training runs of Ren et al. (2019). These parameters are equal to the ones used for generating benchmark results in Andrychowicz et al. (2017) and Plappert, Houthooft, et al. (2018), with exception of the number of MPI workers and the buffer size. Hyper-parameters related to graph generation and training of G-HGG are documented in section A.1.

List of important hyper-parameters:

- Number of MPI workers: 1;
- Number of episodes per iteration M: 50;
- Number of time steps per episode T: 100;
- Buffer size: 10^4 trajectories;
- Actor and critic networks: 3 layers with 256 units and ReLU activation;
- Adam optimizer with 10^{-3} learning rate;
- Polyak-averaging coefficient: 0.95;
- Batch size: 256
- Probability of random action: 0.3;

[1] `https://github.com/mbrucker07/HGG-Extended`

- Scale of additive Gaussian noise: 0.2;
- Probability of HER experience replay: 0.8;
- Number of batches to replay after collecting one trajectory: 20;
- Number of batches to replay in each iteration N: 1000;
- HER replay strategy: `future`;
- Lipschitz constant L: 5.0;
- Distance weight c: 3.0;
- Number of hindsight goals K: 50.

B.3. Details on Data Processing

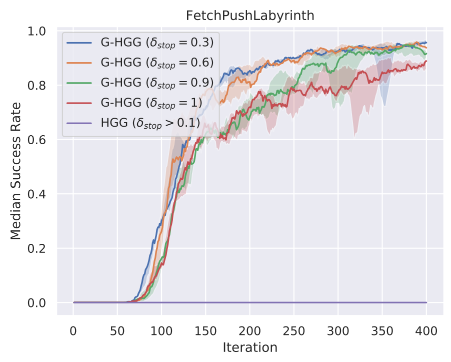
- In policy training of G-HGG and HGG, we sample minibatches using HER.
- EBP is used to prioritize transitions stored in the replay buffer.
- As a normalization step, we use Lipschitz constant $L^* = \frac{L}{(1-\gamma)d^{max}}$ in back-end computation, where d^{max} is the l_2-diameter of the target goal space $\mathcal{G}_T$.
- To reduce computational cost of bipartite matching in HGG / G-HGG, the replay buffer is approximated by a First-In-First-Out queue containing 10^3 recent trajectories.
- An additional Gaussian noise $\mathcal{N}(0, 0.05I)$ is added to goals generated by HGG / G-HGG.

C. Ablation Study on G-HGG Parameters

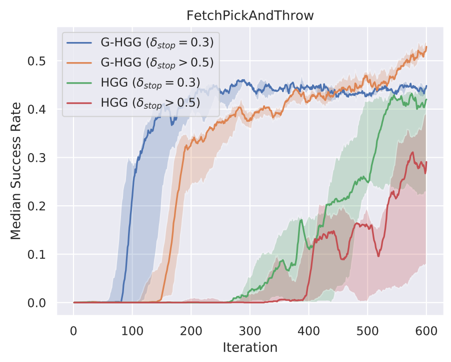
In this chapter, we provide an ablation study on the stop condition parameter δ_{stop} and the number of vertices $n = n_x \cdot n_y \cdot n_z$.

C.1. Stop Condition Parameter

Figure C.1 shows training success rates for different values of the stop condition parameter δ_{stop} of both HGG and G-HGG in the FetchPushLabyrinth and FetchPickAndThrow environments. In alignment with our main results, both plots show that G-HGG outperforms HGG significantly, regardless of the value of δ_{stop}. Furthermore, G-HGG proves to be robust to changes in δ_{stop}. In FetchPushLabyrinth (Figure C.1a), G-HGG performs best with $\delta_{stop} = 0.3$, but other choices of δ_{stop} only show a slight degradation in overall training success and sample efficiency. In FetchPickAndThrow (Figure C.1b), $\delta_{stop} > 0.5$ is the best choice. Despite G-HGG being generally robust to changes in δ_{stop}, we recommend rough parameter tuning on the stop condition parameter. As a rule of thumb, values around $\delta_{stop} = 0.3$ usually yield good success rates and come with the benefit of considerably reduced computation time (see Figure 5.1). Note that all values $\delta_{stop} > 0.5$ yield the same G-HGG training behavior in FetchPickandThrow, since the stop criterion is never met. This is also the case for $\delta_{stop} > 0.5$ in HGG runs of FetchPickAndThrow and for $\delta_{stop} > 0.1$ in HGG runs of FetchPushLabyrinth.



(a) FetchPushLabyrinth



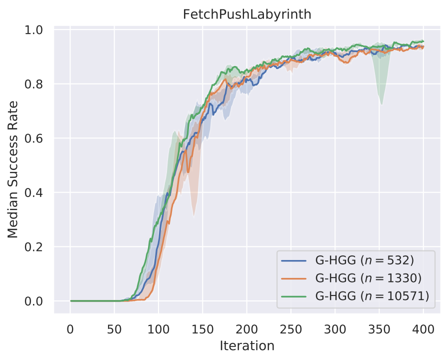
(b) FetchPickAndThrow

Figure C.1.: Ablation study on the stop condition parameter δ_{stop} for G-HGG and HGG in FetchPushLabyrinth and FetchPickAndThrow environments.

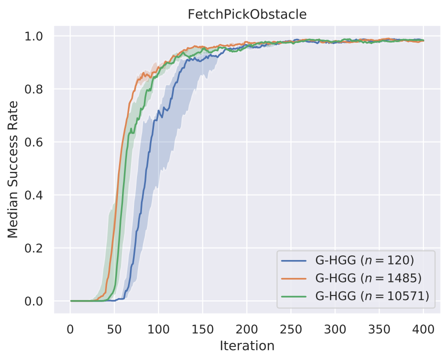
C.2. Number of Vertices

Figure C.2 shows training success rates for different numbers of vertices n in environments FetchPushLabyrinth and FetchPickObstacle. Both plots show that G-HGG is robust to changes in the number of vertices and demonstrates reasonable performance in all chosen configurations.

Note that the number of vertices n cannot be directly specified but depends on the choice of n_x, n_y, and n_z. The same holds for Δ_x, Δ_y, and Δ_z, all of which are directly derived from n_x, n_y, and n_z. An overview of parameters used in this ablation study is provided in table C.1. Even though the choices of n_x, n_y, and n_z might seem arbitrary at first sight, they are not. The blue lines in the plots ($n = 532$ in FetchPushLabyrinth and $n = 120$ in FetchPushObstacle) correspond to the minimal number of vertices required to satisfy the vertex density criterion (3.13) of the respective environment. The orange lines correspond to minimal configurations following a rule of thumb that we recommend for choosing the appropriate number of vertices: n_x, n_y, n_z should be chosen such that 1) the vertex density criterion is met and 2) such that $\Delta_x, \Delta_y, \Delta_z < \delta_g$, where δ_g is the distance threshold for considering a goal as reached (3.1). $\delta_g = 0.05$ in our environments and can be perceived as a kind of critical length of the environment. The green lines were computed with parameter choices from our main results (Figure 4.5).



(a) FetchPushLabyrinth



(b) FetchPickObstacle

Figure C.2.: Ablation study on the number of vertices n for G-HGG in FetchPush-Labyrinth and FetchPickObstacle environments. A detailed overview on used parameters is provided in table C.1.

Table C.1.: G-HGG parameters for the ablation study on n in FetchPushLabyrinth and FetchPickObstacle environments. Note that $\delta_{stop} = 0.3$ and $\delta_g = 0.05$ in all runs.

Environment	n	n_x	n_y	n_z	Δ_x	Δ_y	Δ_z
FetchPushLabyrinth	532	14	19	2	0.038	0.039	0.200
FetchPushLabyrinth	1330	14	19	5	0.038	0.039	0.050
FetchPushLabyrinth	10571	31	31	11	0.017	0.023	0.020
FetchPickObstacle	120	3	10	4	0.250	0.077	0.133
FetchPickObstacle	1485	11	15	9	0.050	0.050	0.050
FetchPickObstacle	10571	31	31	11	0.017	0.023	0.040

Bibliography

Andrychowicz, Marcin et al. (2017). "Hindsight experience replay." In: *Advances in Neural Information Processing Systems*. Vol. 2017-Decem, pp. 5049–5059. arXiv: 1707.01495.

Asadi, Kavosh, Dipendra Misra, and Michael L. Littman (2018). "Lipschitz continuity in model-based reinforcement learning." In: *35th International Conference on Machine Learning, ICML 2018* 1.1, pp. 419–435. arXiv: 1804.07193.

Aubret, A, L Matignon, and S Hassas (2019). "A survey on intrinsic motivation in reinforcement learning."

Brockman, Greg et al. (2016). "OpenAI Gym." URL: http://arxiv.org/abs/1606.01540.

Bygi, Mojtaba Nouri and Mohammad Ghodsi (2007). "3D Visibility Graph."

Canny, John and John Reif (1987). "New Lower Bound Techniques for Robot Motion Planning Problems." In: *Annual Symposium on Foundations of Computer Science (Proceedings)* November, pp. 49–60. ISSN: 02725428. DOI: 10.1109/sfcs.1987.42.

Chebotar, Yevgen et al. (2017). "Path integral guided policy search." In: *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 3381–3388. ISSN: 10504729. DOI: 10.1109/ICRA.2017.7989384. arXiv: 1610.00529.

Colas, Cédric et al. (2019). "CURIOUS: Intrinsically motivated modular multi-goal reinforcement learning." In: *36th International Conference on Machine Learning, ICML 2019*. Vol. 2019-June, pp. 2372–2387. ISBN: 9781510886988. arXiv: 1810.06284.

Dijkstra, E.W. (1959). "A Note on Two Problems in Connexion with Graphs." In: *Numerische Mathematik 1*.

Duan, Ran and Hsin Hao Su (2012). "A scaling algorithm for maximum weight matching in bipartite graphs." In: *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 1413–1424. DOI: 10.1137/1.9781611973099.111.

Eppe, Manfred, Sven Magg, and Stefan Wermter (2019). "Curriculum goal masking for continuous deep reinforcement learning." In: *2019 Joint IEEE 9th International Conference on Development and Learning and Epigenetic Robotics, ICDL-EpiRob 2019*, pp. 183–188. DOI: 10.1109/DEVLRN.2019.8850721. arXiv: 1809.06146.

Fang, Meng, Cheng Zhou, et al. (2019). “DHER: Hindsight experience replay for dynamic goals.” In: *7th International Conference on Learning Representations, ICLR 2019*, pp. 1–12.

Fang, Meng, Tianyi Zhou, et al. (2019). “Curriculum-guided Hindsight Experience Replay.” In: *NeurIPS 2019* 3, pp. 1–12.

Florensa, Carlos, David Held, Xinyang Geng, et al. (2018). “Automatic Goal Generation for Reinforcement Learning Agents.” In: *Proceedings of the 35th International Conference on Machine Learning, PMLR 80*. arXiv: 1705.06366v5.

Florensa, Carlos, David Held, Markus Wulfmeier, et al. (2017). “Reverse Curriculum Generation for Reinforcement Learning.” In: *1st Conference on Robot Learning, CoRL 2017*, pp. 1–14. arXiv: 1707.05300.

Forestier, Sébastien, Yoan Mollard, and Pierre-Yves Oudeyer (2017). “Intrinsically Motivated Goal Exploration Processes with Automatic Curriculum Learning.” URL: http://arxiv.org/abs/1708.02190.

Gasparetto, Alessandro, Paolo Boscariol, and Albano Lanzutti (2015). “Path Planning and Trajectory Planning Algorithms : A General Overview.” In: *Motion and Operation Planning of Robotic Systems*. Springer, pp. 3–27. ISBN: 9783319147055. DOI: 10.1007/978-3-319-14705-5.

Ghosh, Dibya, Abhishek Gupta, and Sergey Levine (2019). “Learning actionable representations with goal-conditioned policies.” In: *7th International Conference on Learning Representations, ICLR 2019*, pp. 1–18. arXiv: 1811.07819.

Goyal, Anirudh et al. (2019). “Recall traces: Backtracking models for efficient reinforcement learning.” In: *7th International Conference on Learning Representations, ICLR 2019*, pp. 1–19. arXiv: 1804.00379.

Hershberger, John and Subhash Suri (1999). “An optimal algorithm for euclidean shortest paths in the plane.” In: *SIAM Journal on Computing* 28.6, pp. 2215–2256. ISSN: 00975397. DOI: 10.1137/S0097539795289604.

Levine, Sergey et al. (2016). “End-to-end training of deep visuomotor policies.” In: *Journal of Machine Learning Research* 17, pp. 1–40. ISSN: 15337928. arXiv: 1504.00702.

Levy, Andrew et al. (2019). “Learning multi-level hierarchies with hindsight.” In: *7th International Conference on Learning Representations, ICLR 2019*, pp. 1–16. arXiv: 1712.00948.

Lillicrap, Timothy P. et al. (2016). “Continuous control with deep reinforcement learning.” In: *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*. arXiv: 1509.02971.

Liu, Hao et al. (2019). “Competitive experience replay.” In: *7th International Conference on Learning Representations, ICLR 2019*, pp. 1–16. arXiv: 1902.00528.

Luo, Yuping et al. (2019). “Algorithmic framework for model-based deep reinforcement learning with theoretical guarantees.” In: *7th International Conference on Learning Representations, ICLR 2019*, pp. 1–27. arXiv: `1807.03858`.

Nair, Ashvin, Bob McGrew, et al. (2018). “Overcoming Exploration in Reinforcement Learning with Demonstrations.” In: *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 6292–6299. ISSN: 10504729. DOI: `10.1109/ICRA.2018.8463162`. arXiv: `1709.10089`.

Nair, Ashvin, Vitchyr Pong, et al. (2018). “Visual reinforcement learning with imagined goals.” In: *Advances in Neural Information Processing Systems* 2018-Decem, pp. 9191–9200. ISSN: 10495258. arXiv: `1807.04742`.

Ng, A. et al. (2006). “Autonomous inverted helicopter flight via reinforcement learning.” In: *Experimental Robotics IX*, pp. 363–372.

Pathak, Deepak et al. (2017). “Curiosity-driven exploration by self-supervised prediction.” In: *34th International Conference on Machine Learning, ICML 2017* 6, pp. 4261–4270. arXiv: `1705.05363`.

Peng, Xue Bin et al. (2018). “Sim-to-Real Transfer of Robotic Control with Dynamics Randomization.” In: *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 3803–3810. ISSN: 10504729. DOI: `10.1109/ICRA.2018.8460528`. arXiv: `1710.06537`.

Péré, Alexandre et al. (2018). “Unsupervised learning of goal spaces for intrinsically motivated goal exploration.” In: *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*, pp. 1–26. arXiv: `1803.00781`.

Peters, Jan and Stefan Schaal (2008). “Reinforcement learning of motor skills with policy gradients.” In: *Neural Networks* 21.4, pp. 682–697. ISSN: 08936080. DOI: `10.1016/j.neunet.2008.02.003`.

Plappert, Matthias, Marcin Andrychowicz, et al. (2018). “Multi-Goal Reinforcement Learning: Challenging Robotics Environments and Request for Research.” In: pp. 1–16. arXiv: `1802.09464`.

Plappert, Matthias, Rein Houthooft, et al. (2018). “Parameter space noise for exploration.” In: *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*, pp. 1–18. arXiv: `1706.01905`.

Rauber, Paulo et al. (2019). “Hindsight policy gradients.” In: *7th International Conference on Learning Representations, ICLR 2019*. arXiv: `1711.06006`.

Ren, Zhizhou et al. (2019). “Exploration via Hindsight Goal Generation.” In: *33rd Conference on Neural Information Processing Systems, NeurIPS 2019*. arXiv: `1906.04279`.

Riedmiller, Martin et al. (2018). “Learning by playing - Solving sparse reward tasks from scratch.” In: *35th International Conference on Machine Learning, ICML 2018* 10.2017, pp. 6910–6919. arXiv: `1802.10567`.

Sahni, Himanshu et al. (2019). "Addressing Sample Complexity in Visual Tasks Using HER and Hallucinatory GANs." In: *Reinforcement Learning for Real Life (RL4RealLife) Workshop in the 36th International Conference on Machine Learning*. arXiv: `1901.11529`.

Schaul, Tom, Dan Horgan, et al. (2015). "Universal Value Function Approximators." In: *Proceedings - IEEE International Conference on Robotics and Automation*. ISBN: 9781538660263. DOI: `10.1109/ICRA.2019.8794206`. arXiv: `1806.06161`.

Schaul, Tom, John Quan, et al. (2016). "Prioritized experience replay." In: *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, pp. 1–21. arXiv: `1511.05952`.

Singh, Satinder, Andrew G. Barto, and Nuttapong Chentanez (2005). "Intrinsically motivated reinforcement learning." In: *Advances in Neural Information Processing Systems*. ISSN: 10495258.

Srinivas, Aravind et al. (2018). "Universal planning networks." In: *35th International Conference on Machine Learning, ICML 2018* 11, pp. 7526–7535. arXiv: `1804.00645`.

Sukhbaatar, Sainbayar et al. (2018). "Intrinsic motivation and automatic curricula via asymmetric self-play." In: *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings* i, pp. 1–16. arXiv: `1703.05407`.

Todorov, Emanuel, Tom Erez, and Yuval Tassa (2012). "MuJoCo: A physics engine for model-based control." In: *IEEE International Conference on Intelligent Robots and Systems*, pp. 5026–5033. ISSN: 21530858. DOI: `10.1109/IROS.2012.6386109`.

Watkins, Christopher J.C.H. (1989). "Learning from Delayed Rewards." PhD thesis.

Watkins, Christopher J.C.H. and Peter Dayan (1992). "Technical Note: Q-Learning." In: *Machine Learning* 8.3, pp. 279–292. ISSN: 15730565. DOI: `10.1023/A:1022676722315`.

Zhang, Shangtong and Richard S. Sutton (2017). "A Deeper Look at Experience Replay." URL: `http://arxiv.org/abs/1712.01275`.

Zhao, Rui, Xudong Sun, and Volker Tresp (2019). "Maximum entropy-regularized multi-goal reinforcement learning." In: *36th International Conference on Machine Learning, ICML 2019*. Vol. 2019-June, pp. 13022–13035. ISBN: 9781510886988. arXiv: `1905.08786`.

Zhao, Rui and Volker Tresp (2018). "Energy-Based Hindsight Experience Prioritization." In: *2nd Conference on Robot Learning, CoRL 2018*. arXiv: `arXiv:1810.01363v4`.

– (2019). "Curiosity-Driven Experience Prioritization via Density Estimation." In: *32nd Conference on Neural Information Processing Systems, NIPS 2018*. arXiv: `1902.08039`.